# Categorizing Short Texts:
# A Comparison of Network Theory and Neural Network Approaches

Stephen Barnes / stbarnes@stanford.edu

## 1. Aims

This project aims to find effective ways of classifying collections of short texts such as Twitter tweets or flashcards into meaningful categories. Specifically, my aim is to classify collections of flashcards in a spaced repetition system, which is a program used to manage large collections of flashcards, widely used to learn and review bodies of knowledge such as foreign languages, medical specializations, or fields of science. Generally they store a database of question-answer pairs created by the user. When the user learns or reviews a card, they are prompted to rate the difficulty of remembering the answer. This data is then used to compute the optimal time to next review the card, according to statistical laws of memory first derived in the PhD. thesis of researcher Piotr Wozniak (TODO cite). Such careful scheduling of reviews optimizes the retention of memories while minimizing the work required.

A flashcard collection is typically thought of as a database or a corpus of short documents. A user will generally create cards about a specific field of study (such as medicine), and within this field there will be various subtopics. One problem with categorizing flashcards is that usually not all cards in a given topic will have any single word in common. Rather, cards about for instance pharmacology will tend to have at least a few words (such as "dose" or "opiate") in common with a few other cards about pharmacology. This suggests that by linking together cards with similar words, topics will arise as modular communities in an underlying network. Therefore we can construct such a network using a measure of text similarity, and then apply community detection algorithms to identify topics.

Meaningful classifications of flashcards would be useful because they help to minimize the context-switching required in the review process. When reviewing we generally want to review similar cards in sequence, as this minimizes work and mirrors the real-world use of knowledge, where related factoids are usually needed at the same time.

I aim to compare the performance of network community detection algorithms with the performance of a neural network approach, which would work as follows. First, we train a neural network to perform some text prediction task. Then we use the neural network's learned weights to transform each card into a many-dimensional vector in $R^n$, i.e. to embed the space of cards in the space $R^n$. Once we have the vectors, we can use any point clustering algorithm, such as k-means, to group nearby vectors into clusters. If our neural network was good, these clusters should correspond to groups of related cards.

## 2. Related prior work

While there is no prior work on categorizing flashcards specifically, there are large literatures on community detection in general networks, and on automatically inferring the topics of text documents.

One commonly-used technique for document classification is called Latent Dirichlet Allocation (LDA). LDA uses a a model analogous to spectral analysis of the term-document matrix. We model each document as a distribution of topics, and each topic as a distribution of words (or n-grams) contained in that topic's documents. We can circularly learn which topic each document has and which words each topic tends to contain using a technique such as expectation maximization, where we first randomly assign topic distributions to documents and then repeatedly use the words in each document to re-infer documents' topic distributions and topics' word distributions until the distributions stop changing.

LDA is generally used for classifying long documents. With short texts, LDA performs poorly because each document contains only a few words and so the term-document matrix becomes sparse. To deal with short texts better, we can use a model such as the Word Network Topic Model[i] (WNTM), originally used for detecting trends in Twitter posts. WNTM is similar to LDA, except that instead of associating documents with topics, we only associate words with topics. Instead of a term-document matrix we construct a word-word co-occurrence matrix to represent the likelihood that each pair of words will appear close to each other in any document. Then we use a method such as expectation-maximization to compute co-occurrence distributions for word-topic combinations. We then classify documents by averaging their individual words' topics.

A more modern machine learning approach would instead construct an embedding of words into a vector space $R^n$. We can do this by for instance constructing a matrix similar to the word-word co-occurrence matrix of WNTM, assigning vectors to words randomly, and then incrementally shifting the words' vectors so that words that occur together are nearby in the vector space, while words that are never seen together are far apart. One open-source implementation of this is the GloVe project[ii], which provides open-source code for constructing such an embedding as well as pre-trained embeddings for words. To use GloVe embeddings to classify short texts, we can assign each text a vector which is the average of its words' vectors, and then apply a standard vector clustering algorithm such as k-means to the documents' vectors.

One problem with this approach is that it discards the context in which the words appear, and does not take into account differences in the importance of words. One technique for solving this is to use a sequential neural network model incorporating LSTM (long-short-term memory) modules. In these networks, each LSTM module receives as input a previous module's state and the next word in the sentence, which it combines to produce the state passed on to the next module. The combination method involves a set of weight parameters trained by doing backpropagation on the loss function defined by some prediction task. These states are high-dimensional vectors encoding all the information necessary to capture the context relevant to whichever prediction task the network is trained to perform.

Google's TensorFlow project contains code for such an LSTM network called lm_1b, trained to guess the next word in a sentence[iii]. The code is open-source and includes pre-trained weights for the network. Since the prediction task is likely to require a lot of information about the words seen so far, I chose to use this network to compute meaningful embeddings of sentences into $R^{1024}$, by extracting the 1,024-dimensional state produced by the final LSTM in the sequence.

I decided to use both the GloVe and LSTM embeddings, checking which produces the best performance.

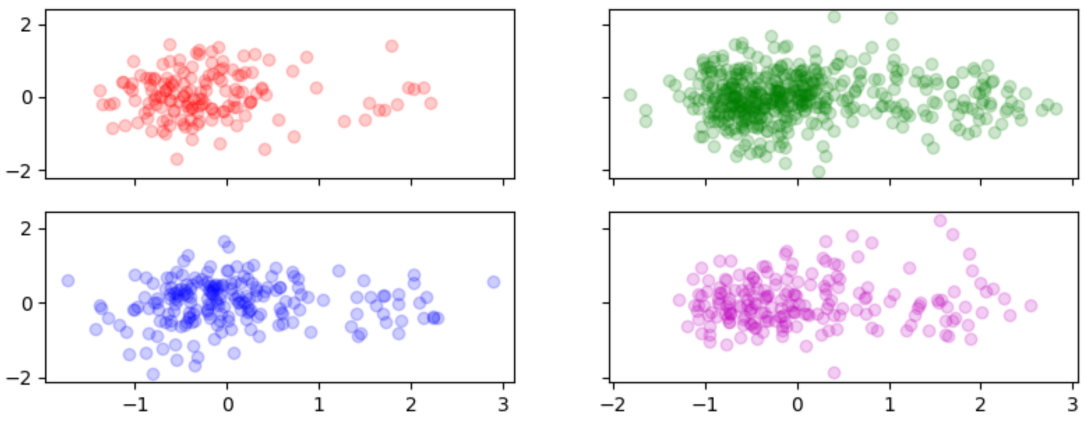
## 3. Dataset

This system is intended to be used on collections of flashcards or other short texts that have not yet been clustered. However, if we test it on such an uncategorized collection, we cannot evaluate the quality of a clustering except by manually inspecting the produced categories, which will either take up a lot of time or be limited to a low number of samples. In light of this, I tested the system on a flashcard deck which had already been classified. The classification algorithms are given no knowledge of this existing clustering.

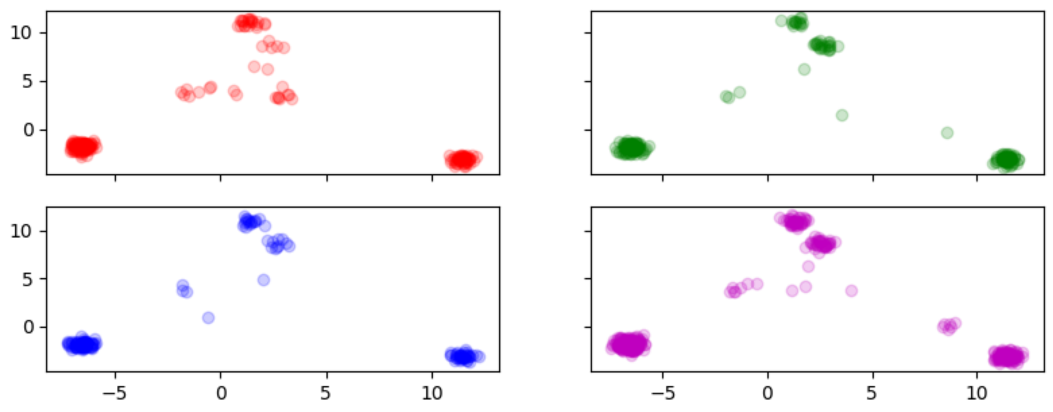
The dataset I used consists of 1,029 flashcards with questions about the history of the United States, intended for SAT preparation; they were uploaded to the Anki website by an anonymous user[iv]. The cards are grouped into 4 separate categories, split into year ranges (1600-1787, 1787-1860, etc.). The categories contain 478, 202, 197, and 152 cards each.

As mentioned above, I am computing an embedding of each card's question into $R^{1024}$ and $R^{100}$ using an LSTM network with pre-trained weights and the average of cards' words' GloVe vectors respectively. Computing the LSTM embeddings requires propagating values through the entire network, which is somewhat computationally expensive; each sentence takes about 10 seconds on an ordinary laptop, so the entire dataset takes 3 hours. To reduce the cost of iteratively improving the code, and to enable you to test the network without downloading the LSTM network and its dependencies (TensorFlow, Bazel, etc.), I have precomputed the LSTM and GloVe embeddings of every card in the dataset and stored them as a NumPy array in a file.

We can try to inspect this dataset by performing principal component analysis (PCA) on the cards' embeddings, both the embeddings generated by GloVe averaging and those generated by the LSTM. In the plot of the GloVe embedding below, each true category is a different color; the x-axes represent the first principal component, and the y-axes represent the second principal component. Notably all true categories seem to be clustered around the same point (0,-0.5), probably because the averaging tends to bring sentences close to the mean by the law of large numbers.



If we instead plot the LSTM embedding's first two principal components in the same way, the cards are definitely concentrated in certain locations, but unfortunately all the true categories' clusters are located in the same places. These clusters seem to represent different types of sentences (e.g. all those starting with "What is..."). A machine learning clustering approach would probably fail dramatically on this dataset, because it would identify each of these clusters as a different category, with little correlation with the true categories.



## 4. Clustering methods and their quality

### 4.1. Evaluating the quality of a clustering

The dataset is already grouped into categories. The question then is: Which method of clustering produces the most "natural" categories – the clustering that is most similar to the natural classification by time period already provided with the dataset? Of course other classifications are possible; for instance, we could classify them by whether they're about slavery, about the First World War, etc. But these other classifications should correlate strongly with certain time periods.

We can then measure the quality of a given clustering as follows. We randomly sample two cards. If they are in the same inferred categories and the same true categories, they are considered to be classified correctly relative to each other; if they are in different inferred categories and different true categories, they are also considered correctly classified. Otherwise, they are considered incorrectly classified. Over many samples, we can estimate the proportion of card-pairs that the algorithm successfully grouped together or apart. This metric obviates any concerns about, for instance, an algorithm that might produce exactly the correct clusters but in a different permutation. It also correctly penalizes algorithms that classify almost all cards in one category. All clustering algorithms used were constrained to produce exactly 4 categories.

Note that despite true categories being known, this is still an unsupervised learning task. The clustering algorithms are given no information about the true categories; they are only used at the end for evaluation.

## 4.2. Baseline approach

When evaluating the network theory and machine learning approaches, we should compare the quality of the clusterings they produce to the quality of some baseline algorithm. A natural choice for this baseline is to simply assign a random category to each card.

We can compute the expected quality of such a random clustering as follows. If we choose two cards at random, with some probability $p$ they will be in the same true category, and then with probability ¼ they will be in the same inferred category. With probability 1-$p$, they will be in different true categories, and then with probability ¾ they will be in different inferred categories. The probability that a pair is correctly classified is thus ¼$p$ + ¾(1-$p$). To find this probability $p$ of two cards being in the same true category, note that it is the sum for each true category that both cards will be in that true category, which is the square of the size of that true category relative to the entire set of cards. If $C$ is the list of category sizes [ 478, 202, 197, 152], and $T$ = 1029 is the total number of cards, then $p$ is:

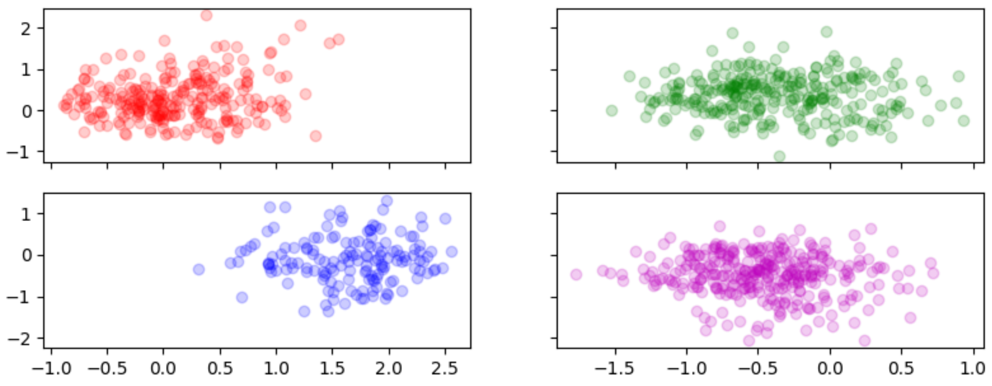
$$p = \sum_{c \in C} \left(\frac{c}{T}\right)^2 = 31.28\%$$

Mathematically, this function is known as the Herfindahl index, and is often used as a measure of the concentration of numbers in a list, for instance when measuring the concentration of market share among companies in some industry.

Using this value of $p$, the expected quality of a random clustering is then 59.36%. Empirical estimates confirm this value.

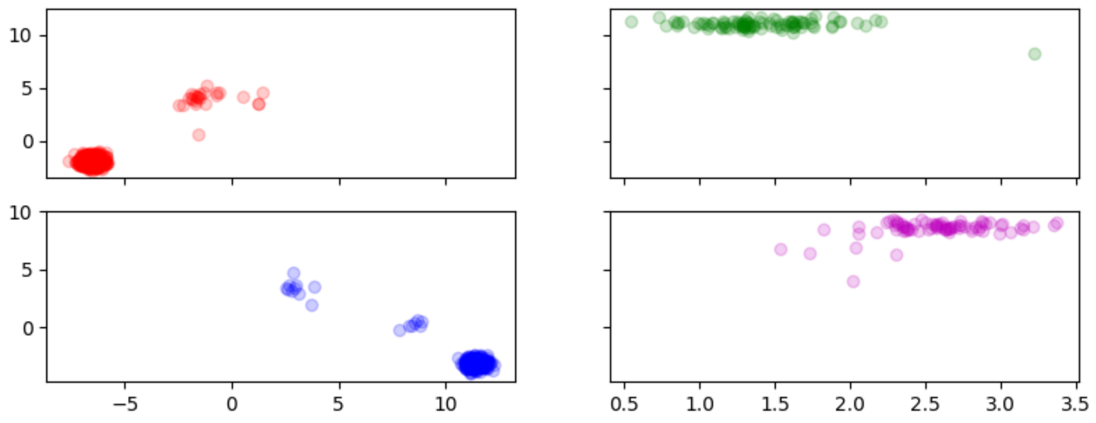
## 4.3. Approach with neural network embedding and clustering

As mentioned above, I will use weights from two different neural network models (GloVe and the lm_1b LSTM network) to convert cards into vectors. Once we have the vectors, there are a variety of clustering algorithms we can apply. Given the fairly small size of our dataset, the fact that true categories must remain hidden from the algorithm, and the fact that we want exactly 4 inferred categories, a good approach is k-means[v]. This algorithm will create 4 points in the embedding space, called centroids. We assign vectors (cards) to the nearest centroid, then shift the centroids to the center of the cards assigned to them, and repeat this until convergence.



Performing k-means on the GloVe embedding of the cards, and evaluating its quality as described above, we achieve an average quality of 58.71% averaged over 100 runs. (K-means does not always produce the same clustering, so running it more than once is necessary.) This is lower than the expected quality of a random clustering. Below is a plot of the first two principal components (on the x- and y-axes respectively) of the GloVe embeddings, where colors indicate the inferred categories (rather than the true categories as in the plots above). We can see that k-means places the centroids in slightly different parts of the embedding space, but there is still a large overlap. One might expect no overlap between inferred categories because of how k-means works, but the PCA transformation removes this structure. There is no overlap in the full 100-dimensional embedding space.

Doing the same with the LSTM embedding, the average quality is 54.11%, which is again worse than the performance of the random clustering. Below is a plot of the first two principal components of the LSTM embeddings, with colors indicating inferred categories. As predicted, k-means successfully identified the clusters in the embedding space, but these clusters do not correspond to the cards' true categories.



Taken together, the neural network clustering approaches tried here have failed completely. Although there may be other standard machine learning approaches that would work better, embedding the cards in some space and then clustering the resulting points is the most straightforward and what I would have tried first. The next section will explore whether network theory approaches can improve on this terrible performance.

## 4.4. Network theory approach

### 4.4.1. Defining the card graph

In the network theory approach, we view the full collection of cards as a network, where the nodes are cards and the edge weights are determined by some similarity measure between cards. We then apply a community detection method to infer categories.

One possible similarity measure is to use the two embeddings above and take the cosine similarity between cards' vectors. Given vectors *v* and *w*, the cosine distance is the cosine of the angle between them, which can be derived using the dot product as follows, where *d* is the metric and $t_1$ and $t_2$ denote the two cards' vectors:

$$d(t_1, t_2) = \frac{t_1 \cdot t_2}{||t_1||\ ||t_2||}$$

However, because of the poor results from clustering the two embedding above, I decided against using them.

Instead, we can view each card's question as a multiset of words by discarding information about word order; this is known as the bag-of-words model. Then we can compute the pairwise similarity by using for instance the Jaccard similarity between these multisets. The Jaccard similarity for two sets *A* and *B* is defined as $|A \cap B|/|A \cup B|$, or in other words the size of the intersection $A \cap B$ as a fraction of $A \cup B$.

However, this approach places too much significance on *stop words* like "the" and "of", which occur often but are not very meaningful. A better way to convert cards to vectors is through the "term-frequency inverse-document-frequency" (tf-idf) algorithm, often used in information retrieval. This is the same as the bag-of-words vector above, except that instead of vector elements being the term's frequency, they are defined as follows, where *t* is the term, *d* is our document, *D* is the corpus of all documents:

$$tfidf(t, d, D) = tf(t, d) \times \log \frac{|D|}{|\{x \in D : t \in x\}|}$$

Here the denominator is called the document frequency, denoting the number of documents that the term *t* appears in. Since stop words appear in many documents, they are given very low weight, so that in the cosine distance their frequency does not have much effect.

Note that tf-idf still technically describes an embedding of cards into some high-dimensional space $R^n$. However, here *n* is the number of distinct words, which will be very high, and the vectors will be very sparse. Thus these vectors may be better thought of as weighted multisets, or as a mapping from cards and words to weights. We still use the same cosine similarity metric to measure similarity between the sparse card representations created by tf-idf. Note that cosine similarities will always be

When creating the graph, to prevent it from being a complete graph which will be computationally expensive to process, we apply a threshold to the similarities. Edges are weighted by their similarity, except where that similarity is lower than the threshold, in which case the edge is omitted.

### 4.4.3. Louvain clustering

For the clustering, I used the Louvain community detection algorithm, which tries to maximize the network's modularity. The modularity implied by a graph and a partition of that graph into communities is defined as the fraction of edges with both endpoints in the same community, weighted by edge weights, minus the expected value of this number if communities were assigned at random. Thus a high modularity means that we've chosen communities that have high-weight edges inside rather than between them. Maximizing modularity is infeasible, so the Louvain algorithm uses a heuristic to approximately maximize it, by first choosing small local communities, then contracting each local community into a node and repeating the local community detection.

Louvain community detection achieved a quality of 34.4%, averaged over 10 runs. This is far below the baseline of random clustering. How is this possible? The answer is that the Louvain algorithm's inferred categories tend to have very lopsided sizes; a typical run produced categories of sizes 985, 14, 29, and 1. Therefore, when evaluating quality, the inferred categories will almost always be equal, so the classification will be deemed correct only when the true categories are also equal, which is fairly uncommon.

To prevent this lopsidedness, we can try to transform the edge weights in various ways. The reason for the failure of Louvain clustering is that some small subsets of nodes are disconnected or almost disconnected from the rest of the graph, and the Louvain algorithm tends to group these small subsets together in a separate group.

### 4.4.3. Growth clustering

To fix this, I implemented a different clustering algorithm which is guaranteed to produce community sizes that are almost equal. This "growth clustering" algorithm proceeds as follows. First we choose 4 random seed nodes to be the first elements of our 4 communities. Then in each step, we grow each community by adding to that community the unassigned node which is most similar to a node already in the community. In this way communities grow at the same rate by adding a single nearby node each step, so that eventually all nodes will be assigned to a community and all communities will have the same size (except for size differences at most 1 introduced at the last step when the number of nodes is not divisible by 4).

This algorithm achieves a quality of 59.61% averaged over 10 runs, with the best run achieving 61.7%. This is barely above the quality of the random clustering, but notably is the first algorithm with a quality above the baseline at all.

We can modify the algorithm so that, instead of adding the new node which is closest to a node already in the community, we add the node whose average similarity to all nodes already in the community is greatest. This modification increases the average quality to 59.69%, with the best run achieving 61.3%.

Overall, the network theory approaches tried here were marginally better than the previous approaches, but still were not capable of producing usably high-quality categories.

## References

i Zuo et al., December 2014. *Word Network Topic Model: A Simple but General Solution for Short and Imbalanced Texts.* https://arxiv.org/abs/1412.5404v1

ii Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. See the paper at GloVe: Global Vectors for Word Representation (PDF) or the website at https://nlp.stanford.edu/projects/glove/.

iii Oriol Vinyals, Xin Pan. See the code and documentation at https://github.com/tensorflow/models/tree/master/research/lm_1b.

iv All 4 decks are downloadable at https://ankiweb.net/shared/decks/US%20history (the 5 ones with year ranges, except for the 1914-1945 deck). They are also available in preprocessed form in my project submission.

v This is for instance the algorithm recommended by a flowchart from SciKit-Learn, at http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.