Document Classification Algorithms

Michael Mulder

This document serves to accompany my project on classifying job posing by the contents of the job descriptions. This document contains the method and theory behind two different types of algorithms to use for document classification. The first algorithm uses non negative matrix factorization (NMF) to determine topics and PCA to visualize these topics. PCA works by maximizing the variance of linear combinations of all features. The second algorithm uses deep learning to determine topics then another neural net to classify the documents into those topics.

1 Data Preprocessing

Before either algorithm can be created we have to convert the text from each document into a text matrix. This can traditionally be done using a count vectorizer or a term frequency-inverse document frequency (tf-idf) vectorizer. In both cases each unique word in the corpus is assigned a number. A vector is created for each document that has a length of the total number of unique words in the corpus. This is done such that each word in the corpus corresponds to an index in the vector. At this point for both vectorizers the word vector is updated such that each time a word occurs, the index of the word in increased by one. For a tf-idf the the log of the inverse document frequency is also calculated as such:

$$idf_i = \log\left(\frac{N}{df_i}\right)$$

Where idf_i is the inverse document frequency for word i. N is the total number of documents and df_i is the number of documents in the corpus that contain i.

At this point a weight $w_{i,j}$ is calculated and updated for each word in the original term frequency matrix. This weight is calculated as,

$$w_{i,j} = tf_{i,j} \times idf_i,$$

Where $w_{i,j}$ is the new tf-idf vectorizer weight for the jth document and the ith word. $tf_{i,j}$ is the count of the ith word in the jth document.

For this project in both algorithms tf-idf was used, however this is just a parameter that can be changed, depending on use case.

2 Algorithm 1: Traditional Matrix Dimension Reduction

Non-Negative Matrix Factorization (NMF) is the gold standard for semi-supervised or unsupervised clustering. NMF is a form of matrix decomposition that inherently lends itself to clustering. The algorithm works by creating two partial matrices W and H. W is a matrix with shape (number of documents, categories), this is your category (a.k.a features, topics) matrix. H is a matrix with shape (categories, number of words). These are the decomposed versions of the total tf-idf matrix, $V_{original}$. In a perfect scenario, the equation

$$V_{original} = WH$$

holds. In reality it will not likely be a perfect match as such $V_{predict}$ is the prediction matrix, such that

$$V_{predict} = WH.$$

Once $V_{predict}$ is found a loss function is calculated against $V_{original}$ as shown and the H and W matrices are updated according to the minimization rule below.

$$\min_{W,H} \|V_{original} - WH\|, WH \in R_+$$

The most commonly used algorithm to solve this minimization rule is Lee and Seung's multiplicative update rule (http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf). The multiplicative update rule uses element wise multiplication instead of full matrix multiplication. Exact update rules for NMF and other matrix decompositions can be found in the linked paper.

This process is done for a certain number of iterations or until a tolerance level is found in the loss function. Once the matrices have been found, we can look at the H matrix to determine which are the most important features or in this case words in each topic. This is just a simple argmax sort, that gives us our topics. Now we can fit each document to the NMF matrix to determine which topic it most likely belongs in.

3 Algorithm 2: Deep Learning and Autoencoding for Dimension Reduction

The first portion of this algorithm is to use an autoencoding net to do some unsupervised learning to determine how many latent topics exist.

Autoencoding neural nets can be thought of as a compression algorithm. The data (x's) gets passed through the net and then reduced into a smaller level of dimensions (z's) and then re-expanded into it's original dimensions (x primes). This is done through a series of non-linear transformations, within the net.

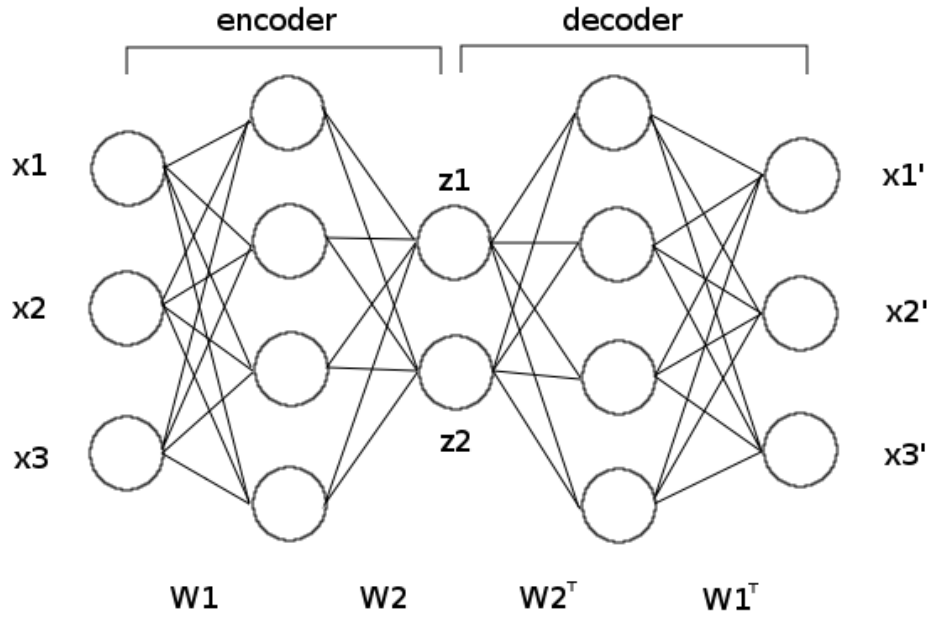

Figure 1: Autoencoding Net Example (http://inspirehep.net/record/1252540/plots)

Before we can implement this method we have to preprocess this data according to the tf-idf method provided above. The algorithm is outlined above however the sklearn tf-idf vectorizer was used as it has some useful methods. These include smoothing, which allows for handling of new words not seen before, as well as max features which limits the number of words used in the vectorizer.

Determining the architecture for a neural net is commonly the most difficult part. With this particular example we can assume 2000 unique words, so our input has to be 2000 nodes, then we can slowly reduce dimension and add regularization until we start to actually get clustering. At each layer we can reduce the number of dimensions until we feel we have a reasonable clustering of the data. Then we can visualize. In the example in the

jupyter notebook each layer is fully connected with tanh activation functions. Essentially this allows dimension reduction in non-linear space.

Pros:

- Can discern very complex underlying traits
- With good training, good data, and a strong architecture very accurate

Cons:

- Computationally expensive
- Some simpler models might perform almost as well
- Complicated
- Lots of hyper parameters to tune

Once the data has been categorized and visualized in 2D or 3D space. We have to create labels for each data point. This could be done by using a k-nearest neighbors clustering algorithm in the embedded space. K-nearest neighbors works by choosing a random initial start and the linking all closest neighbors. Next the algorithm calculates the center of each cluster and iterates through the process again. This is continued until a stopping point or a certain distance threshold is met.

After the data has been labeled we can create a new net the automatically classifies new documents. There are many ways and architectures for this net. Some guidelines would be:

- Input layer would be vocabulary length long
- Output layer would be categories length long
- Categorical crossentropy

Some other things to consider would include:

- LSTM layers to handle sequences of text instead of just vocabulary
- Regularization (L2/Dropout)
- Hidden Layer Length/number of layers

This process should actually work pretty well, I think using a net for the final classification would likely return the best results compared to any other algorithm. Labeling and classifying the data initially will be more difficult, it might be worthwhile to mix and match and use NMF to create document categories and use the net to classify.

4 Conclusion

Either method should work quite well. As mentioned I believe the best method for interpretability and results, would be using NMF to get the major categories on a test dataset and create labels. Then using a neural net to categorize according to those labels. In either method you're categories don't have to be strict, for example if I had chosen just two categories we might have a data category and a web category. If there are a lot of categories (more than 10) it may begin to be more difficult to determine them and create labels but you may still be able to use NMF to get the major components in each topic. Categories may have underlaying features that are not strictly apparent either, for example perhaps all data science jobs require machine learning and no other job does, so instead of classifying data science we are classifying jobs that require machine learning. This can be useful if you are trying to determine the best way to split data such that each has it's own unique properties.