Notes: Neural Network Methods for Natural Language Processing – Part 2 Working with Natural Language Data, Ch6-8

Yingbo Li

01/20/2021

Table of Contents

Feature extraction

- The mapping from textural data to real valued vectors is called feature extraction or feature representation
- A process called tokenization is in charge of splitting text into tokens (what we call here words) based on whitespace and punctuation

Features of words

- We often look at the lemma (the dictionary entry) of the word, mapping forms such as *booking, booked, books* to their common lemma *book*.
- A coarser process than lemmatization, that can work on any sequence of letters, is called stemming. For example, *picture, pictures, pictured* will all be stemmed to *pictur*

Features of text: weighting

- When using the bag-of-words approach, it is common to use TF-IDF weighting: TF $\times$ IDF
- Consider a document d which is part of a larger corpus D. For each word w in d, its normalized count in the document is the Term Frequency:

$$\text{TF} = \frac{\#_d(w)}{\sum_{w' \in d} \#_d(w')}$$

- Inverse Document Frequency

$$\text{IDF} = \log \frac{|D|}{|\{d \in D : w \in d\}|}$$

Ngram features

- Word-bigrams, as well as trigrams of letters or words, are common
- A bag-of-bigrams representation is much more powerful than bag-of-words, and in many cases proves very hard to beat
- Since it is hard to know a-priori which ngrams will be useful for a given task, a common solution is to include all ngrams up to a given length, and let the model regularization discard of the less interesting ones by assigning them very low weights

Ngram and neural networks

- Note that vanilla neural network architectures such as MLP cannot infer ngram features from a document on their own in the general case. Thus, ngram features are also useful in the context of nonlinear classification
- Bidirectional RNNs generalize the ngram concept even further, and can be sensitive to information ngrams of varying lengths, as well as ngrams with gaps in them

NLP features for document topic classification

- A good set of features will be the bag-of words, perhaps plus a bag-of-word-bigrams
- If we do not have many training examples
  - We may benefit from pre-processing document by replacing each word with its lemma
  - We may also replace or supplement words by distributional features such as word clusters or word-embedding vectors
- When using a bag-of-words, it is sometimes useful to weigh each word with proportion to its informativeness, for example, using TF-IDF weighting

Dense encoding (feature embeddings)

- Each core feature (e.g., word) is embedded into a d dimensional space, and represented as a vector in that space
- The dimension d is usually much smaller than the number of features
  - For example, each item in a vocabulary of 40000 items can be represented as 100 or 200 dimensional vector
- In current research, d ranges between 50 to a few hundreds (and in some extreme cases, thousands)
- A good rule of thumb would be to experiment with a few different sizes, and choose a good trade-off between speed and task accuracy

Embeddings and neural networks

- The biggest change in the input when moving from linear to deeper classfier is the move from sparse representations (one-hot encoding) to a dense representation (embedding)
- Another difference is that we mostly need to extract only core features and not feature combinations (we don not need to manually do feature engineering)
- One benefit of using dense and low-dimensional vectors (embeddings) is computational: the majority of neural network toolkits do not play well with very high-dimensional, sparse vectors

One-hot vectors vs dense embeddings

- The main benefit of the dense representations is in generalization power: if we believe some features may provide similar clues, it is worthwhile to provide a representation that is able to capture these similarities
- When one-hot representations might be better than embeddings: when the feature space is relatively small and the training data is plentiful

Combining dense vectors: window based features

- The prominent options are concatenation, summation (or averaging), and combinations of the two
- Consider the case of encoding a window of size $k = 2$ to each side of a focus word. The word vectors of the window items are $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$
  - Summation: if we do not care about the relative positions of the words within the window, we can encode the window as a sum $\mathbf{a} + \mathbf{b} + \mathbf{c} + \mathbf{d}$
  - Concatenation: if we do care about the order, then we should rather use concatenation $[\mathbf{a}; \mathbf{b}; \mathbf{c}; \mathbf{d}]$
  - We may not care about the order, but would want to consider words further away from the context word less important. Then we can use a weighted sum $\frac{1}{2}\mathbf{a} + \mathbf{b} + \mathbf{c} + \frac{1}{2}\mathbf{d}$
  - We can mix-and-match. For example, $[(\mathbf{a} + \mathbf{b}); (\mathbf{c} + \mathbf{d})]$

Variable number of features: continuous bag of words

- Feed-forward networks assume a fixed dimensional inputs
- When we need to represent an unbounded number of features using a fixed size vector, one way is through continuous bag of words (CBOW). For features $f_1, \ldots, f_k$, CBOW is the average of and their corresponding vectors $v(f_1), \ldots, v(f_k)$, i.e.,

$$\text{CBOW}(f_1, \ldots, f_k) = \frac{1}{k} \sum_{i=1}^{k} v(f_i)$$

- A simple variation is weighted CBOW, where each feature f_i has weight a_i

$$\text{WCBOW}(f_1, \ldots, f_k) = \frac{1}{\sum_{i=1}^{k} a_i} \sum_{i=1}^{k} a_i v(f_i)$$

  – For example, in a document classification task, we can use TF-IDF as the weights

Relationship Between One-Hot and Dense Vectors

Relationship between one-hot and dense vectors

- Using one-hot vectors as input when training a neural network is to dedicate the first layer of the network to learning a dense embedding vector for each feature based on the training data
- When using dense vectors, each categorical feature value (e.g., word) f_i is mapped to a dense, d-dimensional vector $v(f_i)$. This mapping is performed through the use of an embedding layer or a lookup layer
- For a vocabulary of $|V|$ words, the collection of vectors can be thought of as a $|V| \times d$ embedding matrix $\mathbf{E}$, in which each row corresponds to an embedded feature
- Let $\mathbf{f}_i$ be the one-hot representation of feature f_i, then its embedding row is

$$v(f_i) = \mathbf{f}_i \mathbf{E}$$

- The word vectors are often concatenated to each other before being passed to the next layer

Padding and unknown words

- In some cases the feature extractor will look for things that do not exist. The suggested solution is to add a special padding symbol to the embedding vocabulary
- For out-of-vocabulary (OOV) items, it is recommended to reserve a special symbol Unk, representing an unknown token
- In any case, it is advised to not share the padding and the unknown vectors, as they reflect two very different conditions

Word dropout

- Reserving a special embedding vector for unknown words is not enough, because if all the features in the training set have their own embedding vectors, the unknown-word condition will not be observed in training
- Since the model needs to be exposed to the unknown-word condition during training, we can use word-dropout, i.e., when extracting features in training, randomly replace words with the unknown symbol
- This can be based on word's frequency: less frequent words will be more likely to be replaced by the unknown symbol than frequent ones

Feature combinations

- One of the promises of the nonlinear neural network models is that one needs to define only the core features. The nonlinearity of the classifier, as defined by the network structure, is expected to take care of finding the indicative feature combinations, alleviating the need for feature combination engineering
- Computational complexity of classification in kernel methods scales linearly with the size of the training data, make them too slow for most practical purpose
- Computational complexity of classification using neural networks scales linearly with the size of the network, regardless of the training data size

References

- Goldberg, Yoav. (2017). Neural Network Methods for Natural Language Processing, Morgan & Claypool