STAT430: Machine Learning for Financial Data

LarryHua.com/teaching

Spring 2019

Deep-learning Models for sequence data

Sequence data as input

- Two approaches
  - 1D convnet
  - Recurrent neural network (RNN)
- Applications
  - Document classification and time series classification, such as identifying the topic of an article or the author of a book
  - Time series comparisons, such as estimating how closely related two documents or two stock tickers
  - Sequence-to-sequence learning, such as language translation
  - Sentiment analysis
  - Time series forecasting

Dealing with text data - vectorize text

- tokenize text into tokens (words/characters/n-grams)
  - segment text into words, and transform each word into a vector
  - segment text into characters, and transform each character into a vector
  - extract n-grams of words or characters, and transform each n-gram into a vector
    - n-grams are overlapping groups of multiple consecutive words or characters
- associate numeric vectors with tokens
  - one-hot encoding
  - token embedding / word embedding

Dealing with text data - one-hot encoding

- One-hot encoding
  - assign a unique integer i to each word / character
    - In Keras, 0, 1, and 2 are reserved for padding, start of sequence, and unknown
  - construct a binary vector of size N for each word / character, with all entries being 0 except for the ith entry, which is 1
- Keras has built-in utilities for doing one-hot encoding text at the word level or character level
- one-hot hashing trick: can be used when the number of unique tokens in the vocabulary is too large to handle explicitly
  - hash words into vectors of fixed size (hash functions: arbitrary size to fixed size)
  - allows online encoding of the data (starting to generate token vectors right away, before having seen all of the available data)
  - limitation: hash collisions: the likelihood of hash collisions decreases when the dimensionality of the hashing space is much larger than the total number of unique tokens being hashed
- Try R

Dealing with text data - word embedding

- Word embedding (i.e., word vectors) are low-dimensional dense floating point vectors
  - unlike one-hot encoding, word embedding are learned from data
  - much lower dimension 256 / 512 / 1024 when dealing with very large vocabularies
    - one-hot encoding generally leads to vectors that are 20,000-dimensional or higher
- two ways to obtain word embedding
  - learn word embedding jointly with the main task, just like learning the weights of a neural network
  - pre-trained word embedding

Word embedding - learned jointly with tasks

- if randomly assign a vector to a word, then resulting embedding space would have no structure
- does not reflect semantic distance
  - eg, the words "accurate" and "exact" may be of very different embedding
- data-and-task driven, so need to learn word embedding for different data/task

Word embedding - learned jointly with tasks

- `layer_embedding()`
  - input: 2D tensor of integers of shape `(samples, sequence_length)`
    - sequences in a batch must have the same length
      - sequences that are shorter should be padded with zeros
      - sequences that are longer should be truncated
  - output: 3D floating-point tensor, of shape `(samples, sequence_length, embedding_dimensionality)`
    - such a 3D tensor can then be processed by an RNN layer or a 1D convolution layer
  - Some commonly used parameters
    - `input_dim`: size of the vocabulary (maximum integer + 1)
    - `output_dim`: dimension of the dense embedding
    - `input_length`: length of the input sequences
- Try R

Word embedding - pre-trained

- for small data
- pre-computed databases of word embedding in a Keras
  - Word2Vec
  - GloVe
- Try R
- Back to Course Scheduler