Relevant app prediction from app details

Arijit Pramanik

Abstract. We want to give the user the list of most relevant apps corresponding to his preferences, extracted from a set of keywords.

1. Libraries in Python

We require gensim and spaCy libraries in python for NLP. Refer to these site for an intuition:
https://elitedatascience.com/python-nlp-libraries

2. Approaches

2.1. Approach 1: Using scikit-learn One-vs-Rest classifier

- Label each of the 400 descriptions by the category of the app. This becomes a classification task now
- This will be a multiclass multilabel classification. Use the keywords of the query to try and predict the label of those keywords post training on the descriptive chunk of words
- This will give poor results, since categories are 36, whereas training examples are 400, with some categories having barely 2-3 examples to train.

2.2. Approach 2: Train on the chunk using gensim

- Use NLTK library in python to download Brown Corpus, which is a general corpus of sentences for training, covering a wide range of words. We may use this to build our vocabulary
- A list of descriptions for each app is used for training, which form a list of documents. Then we use the inbuilt class of docsim Similarity to calculate the similarities.
- We may also use a word2vec model, which converts this list of documents into a vector space with a desired number of features and other parameters. Then we can use inbuilt functions for calculating similarity among words, (for each document, take the cbow mean of each of the word vectors, representative of the document and use this to calculate the similarity between the keywords and each document
• Instead, we can use doc2vec for representing each of the documents, and then use
the inbuilt library functions to carry out a comparison.

2.3. Approach 3: Using spaCy to leverage industrial NLP training

• So, we use the ‘en’ model of spaCy and convert the list of documents and the list
of keywords to relevant vector representations.
• Using the inbuilt similarity model, calculate similarity scores between the specified
keywords and each of the documents

3. Drawing Inference From a Bayesian Network

3.1. Building the Model

• Use a python driver for neo4j for connecting to the graph database hosted on
the server
• The model has been built intuitively and attached herewith in BayesianNetwork.pdf
• Each node in neo4j has an attached name, flag to distinguish between discrete
and boolean variables, parent nodes in-order of appearance, boundaries of
each bucket for discrete variables and boolean values in case of boolean variables,
probabilities of each value of the random variable associated with the node, and
units of the respective quantity
  NOTE: Probabilites are specified herewith as prob_0, prob_1 and so on for each
value of the random variable in [0, 1, ....]

3.2. Verifying the correctness of the Model with careful insertions and modifications

• You should follow the neo4j syntax for creating a node with desired label and
properties with CREATE command
• You can modify or add new properties to a node by using its labels and the SET
command with MATCH
• Use get_nodes(), get_relations() and verify_model() to see the nodes,
relations and verify whether the model is consistent with the CPDs specified

3.3. Drawing the inference

• Inferences are specified as variables which you want to infer, given the evidence,
i.e., all observed variables are specified in the evidence with appropriate values in
their range reflecting the appropriate buckets of values.
• Inference Query returns the probability of the unobserved variables conditioned on
the values of the given variables.