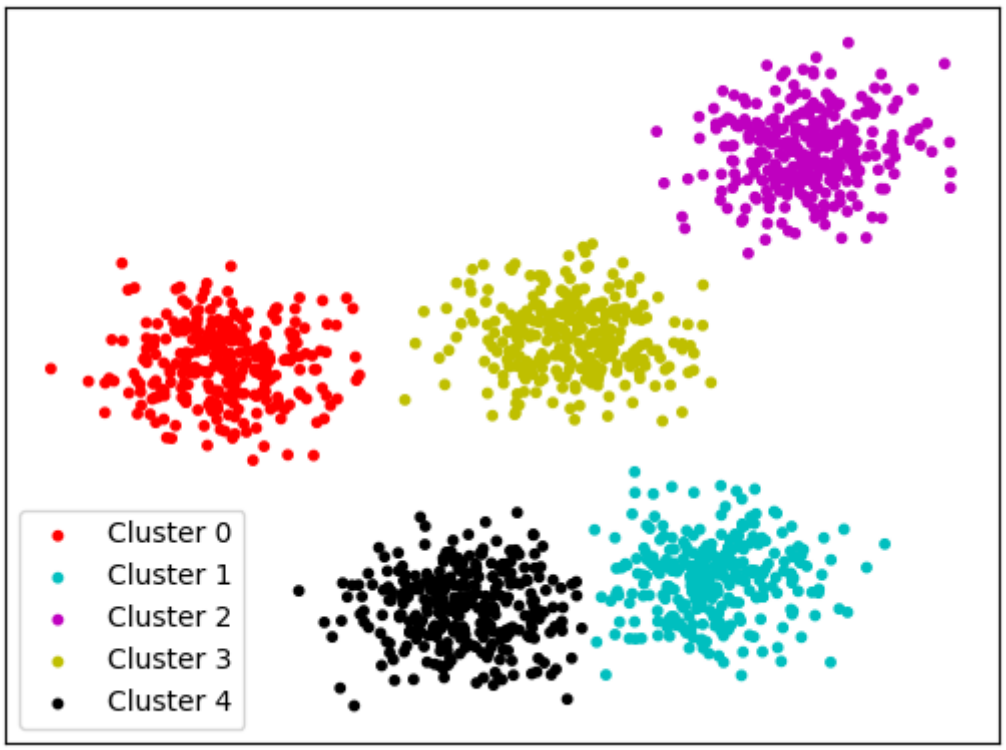
Semantic similarity classifier and clustering sentences based on semantic similarity.

Manish Chablani Jun 29, 2019 · 2 min read ★



Recently we have been doing some experiments to cluster semantically similar messages, by leveraging pre-trained models so we can get something off the ground

using no labelled data. Task here is given a list of sentences we cluster them such that semantically similar sentences are in same cluster and number of clusters is not predetermined.

Task of semantic similarity classifier is: Given two sentences/messages/paragraphs classify of they are semantically equivalent.

Step 1: Represent each sentence/message/paragraph by an embedding. For this task we used infersent and it worked quite well.

> *InferSent* is a *sentence embeddings* method that provides semantic representations for English sentences. It is trained on natural language inference data and generalizes well to many different tasks.

facebookresearch/InferSent

InferSent sentence embeddings. Contribute to facebookresearch/InferSent development by creating an account on...

github.com

Here is the code:

```
# Load infersent model
model_version = 2
MODEL_PATH = "infersent_sentence_encoder/infersent%s.pkl" %
model_version
params_model = {'bsize': 64, 'word_emb_dim': 300, 'enc_lstm_dim':
2048,
                'pool_type': 'max', 'dpout_model': 0.0, 'version':
model_version}
model = InferSent(params_model)
model.load_state_dict(torch.load(MODEL_PATH))

# If infersent1 -> use GloVe embeddings. If infersent2 -> use
InferSent embeddings.
W2V_PATH = 'infersent_sentence_encoder/GloVe/glove.840B.300d.txt' if
model_version == 1 else 'infersent_sentence_encoder/fastText/crawl-
300d-2M.vec'
model.set_w2v_path(W2V_PATH)
```

```
#load data
ds = pd.read_msgpack('./ds.mp')
sentences = ds['text']

# generate infersent sentence embeddings
model.build_vocab(sentences, tokenize=True)
embs = model.encode(sentences, tokenize=True)
```

Step 2: Find candidates of semantically similar sentences/messages/paragraphs

Idea here is to index representation (embedding) of each sentence/message/paragraph and pick k (=10) NN (nearest neighbor) candidates for each sentence based on distance threshold. We found that nmslib is very fast and efficient.

```
import nmslib

NTHREADS = 8
def create_index(a):
    index = nmslib.init(space='angulardist')
    index.addDataPointBatch(a)
    index.createIndex()
    return index

def get_knns(index, vecs, k=3):
    return zip(*index.knnQueryBatch(vecs, k=k,num_threads=NTHREADS))

nn_wvs = create_index(embs)

to_frame = lambda x: pd.DataFrame(np.array(x)[:,1:])

idxs, dists = map(to_frame, get_knns(nn_wvs, embs, k=10))

catted = pd.concat([idxs.stack().to_frame('idx'),
dists.stack().to_frame('dist')],
axis=1).reset_index().drop('level_1',1).rename(columns={'level_0':
'v1', 'idx': 'v2'})
```

Step 3: Get prediction probability of candidate pairs on semantic similarity classifier. (Details on the semantic similarity classifier in a future blog post)

Think of step 2 as candidate generation (focusing on recall) and step 3 as focusing on precision. Of all the candidates that are considered potential duplicates here we assign probability to each pair.

Step 4: Agglomerative clustering to merge clusters

Based on candidates that are considered duplicates in step 3 we merge clusters using agglomerative clustering implementation in scikit. In agglomerative clustering all observations start as thier own clusters and clusters are merge using the merge criteria specified until convergence, at which point no more merges are happening.

sklearn.cluster.AgglomerativeClustering - scikit-learn 0.21.2 documentation

Connectivity matrix. Defines for each sample the neighboring samples following a given structure of the data. This can...

scikit-learn.org

This works fairly well in practice and the clusters formed have good semantic equivalence.

Machine Learning NLP


App Store


Google Play