DeepLearningLab

January 7, 2020

1 Data import

1.1 Question 0 - Get common wikidata occupations

Write a sparql query that retrieves the top 20 occupations on wikidata (wikidata property P106).

You may use the interface https://query.wikidata.org/ to try different queries. Here are some example sparql queries: https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples

[1]:
```
query = """
 SELECT ?o
 WHERE {
     ?item wdt:P106 ?o
 }
 GROUP BY ?o
 ORDER BY DESC(COUNT(?o))
 LIMIT 20
"""
```

The following assertion should pass if your answer is correct.

[2]:
```
import requests

occupations = ['Q82955', 'Q1650915', 'Q937857', 'Q36180', 'Q33999', 'Q1028181',␣
 ↪'Q1930187', 'Q1622272', 'Q177220', 'Q49757', 'Q36834', 'Q47064', 'Q40348',␣
 ↪'Q10800557', 'Q43845', 'Q201788', 'Q639669', 'Q2526255', 'Q28389', 'Q39631']

def evalSparql(query):
    return requests.post('https://query.wikidata.org/sparql', data=query,␣
 ↪headers={
        'content-type': 'application/sparql-query',
        'accept': 'application/json',
        'user-agent': 'User:Tpt'
    }).json()['results']['bindings']

myOccupations = [val['o']['value'].replace('http://www.wikidata.org/entity/',␣
 ↪'') for val in evalSparql(query)]
```

```
assert(frozenset(occupations) == frozenset(myOccupations))
```

1.2 Occupations labels

We load the labels of the occupations from Wikidata

```
[3]: occupations_label = {}

     query = """
     SELECT DISTINCT ?o ?oLabel
     WHERE {
         VALUES ?o { %s }
         SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
     }"""% ' '.join('wd:' + o for o in occupations)

     for result in evalSparql(query):
         occupations_label[result['o']['value'].replace('http://www.wikidata.org/
      ↪entity/', '')] = result['oLabel']['value']

     print(occupations_label)
```

```
{'Q82955': 'politician', 'Q177220': 'singer', 'Q201788': 'historian', 'Q639669':
'musician', 'Q937857': 'association football player', 'Q1028181': 'painter',
'Q1622272': 'university teacher', 'Q1650915': 'researcher', 'Q1930187':
'journalist', 'Q2526255': 'film director', 'Q10800557': 'film actor', 'Q40348':
'lawyer', 'Q33999': 'actor', 'Q36834': 'composer', 'Q36180': 'writer', 'Q43845':
'businessperson', 'Q39631': 'physician', 'Q28389': 'screenwriter', 'Q49757':
'poet', 'Q47064': 'military personnel'}
```

We load *all* the labels of the occupations from Wikipedia

```
[4]: occupations_labels = {k: [v] for k, v in occupations_label.items()}

     query = """
     SELECT ?o ?altLabel
     WHERE {
       VALUES ?o { %s }
       ?o skos:altLabel ?altLabel . FILTER (lang(?altLabel) = "en")
     }""" % ' '.join('wd:' + o for o in occupations)

     for result in evalSparql(query):
         occupations_labels[result['o']['value'].replace('http://www.wikidata.org/
      ↪entity/', '')].append(result['altLabel']['value'])

     print(occupations_labels)
```

```
{'Q82955': ['politician', 'political leader', 'political figure', 'polit.'],
'Q177220': ['singer', 'singer', 'vocalist'], 'Q201788': ['historian',
'historians', 'historiographer'], 'Q639669': ['musician'], 'Q937857':
['association football player', 'footballer', 'football player', 'soccer
player', 'association footballer'], 'Q1028181': ['painter'], 'Q1622272':
['university teacher', 'professor', 'lecturer', 'college lecturer', 'college
professor', 'university teachers'], 'Q1650915': ['researcher', 'researchers',
'research personnel'], 'Q1930187': ['journalist', 'journo'], 'Q2526255': ['film
director', 'director', 'movie director', 'motion picture director'],
'Q10800557': ['film actor', 'film actress', 'film actor', 'movie actor', 'movie
actress'], 'Q40348': ['lawyer', 'attorney', 'Jurisprudente', 'lawyers'],
'Q33999': ['actor', 'actress', 'actors', 'actresses', 'thespian'], 'Q36834':
['composer'], 'Q36180': ['writer', 'author', 'authors', 'writers'], 'Q43845':
['businessperson', 'businessman', 'dealer', 'business person', 'business woman',
'businesswoman', 'business man'], 'Q39631': ['physician', 'doctor', 'medical
doctor', 'medical practitioner', 'physicians'], 'Q28389': ['screenwriter',
'writer', 'screen writer', 'scriptwriter', 'scenarist', 'script writer'],
'Q49757': ['poet', 'bard', 'poetess'], 'Q47064': ['military personnel',
'military member']}
```

1.3 Wikipedia articles

Here we load the training and the testing sets. To save memory space we use a generator that will read the file each time we iterate over the training or the testing examples.

[5]:
```python
import gzip
import json

def loadJson(filename):
    with gzip.open(filename, 'rt') as fp:
        for line in fp:
            yield json.loads(line)

class MakeIter(object):
    def __init__(self, generator_func, **kwargs):
        self.generator_func = generator_func
        self.kwargs = kwargs
    def __iter__(self):
        return self.generator_func(**self.kwargs)

training_set = MakeIter(loadJson, filename='wiki-train.json.gz')
testing_set = MakeIter(loadJson, filename='wiki-test.json.gz')
```

2 Extract occupations from summaries

2.1 Task 1 - Dictionnary extraction

Using `occupations_labels` dictionnary, identify all occupations for each articles. Complete the function predict_dictionary() below and then evaluate the accuracy of such approach. It will serve as a baseline.

[6]:
```python
def predict_dictionnary(example, occupations_labels):
    ## example['summary'] contains the summary of the article
    ## Code here
    result = []
    for k in occupations_labels:
        v = occupations_labels[k]
        for o in v:
            if o in example['summary']:
                result.append(k)
    return result

def evaluate_dictionnary(training_set, occupations_labels):
    nexample = 0
    accuracy = 0.
    prediction = None
    for example in training_set:
        prediction = predict_dictionnary(example, occupations_labels)

        p = frozenset(prediction)
        g = frozenset(example['occupations'])
        accuracy += 1.*len(p & g) / len(p | g)
        nexample += 1
    return accuracy / nexample

evaluate_dictionnary(training_set, occupations_labels)
```

[6]:
```
0.5678982587144555
```

2.2 Task 2 - Simple neural network

We load the articles "summary" and we encode the data using the average of the word vectors. This is done with spacy loaded with the fast text vectors. To do the installation/loading [takes 8-10 minutes, dl 1.2Go] :

```
pip3 install spacy

wget https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.en.300.vec.gz
```

Use the following command to create the new model from the fast text vectors (please adjust the path_to_file)

```
python3 -m spacy init-model en path_to_file/en_vectors_wiki_lg --vectors-loc cc.en.300.vec.gz
rm cc.en.300.vec.gz
```

[7]:
```python
import spacy
nlp = spacy.load('en_vectors_wiki_lg')

def vectorize(dataset, nlp):
    result = {}
    for example in dataset:
        doc = nlp(example['summary'], disable=['parser', 'tagger'])
        result[example['title']] = {}
        result[example['title']]['vector'] = doc.vector
        if 'occupations' in example:
            result[example['title']]['occupations'] = example['occupations']
    return result

vectorized_training = vectorize(training_set, nlp)
vectorized_testing = vectorize(testing_set, nlp)
nlp = None
```

Next, we encode the output as multi-hot vectors.

[8]:
```
# We encode the data
import numpy as np

inputs = np.array([vectorized_training[article]['vector'] for article in␣
 ↪vectorized_training])
outputs = np.array([[(1 if occupation in␣
 ↪vectorized_training[article]['occupations'] else 0)
                    for occupation in occupations ] for article in␣
 ↪vectorized_training])
```

[9]:
```
print(len(outputs[0]))
```

```
20
```

> Using keras, define a sequential neural network with two layers. Use **binary_crossentropy** as a loss function and **sigmoid** as the activation function of the output layer

You can look into the documentation here: https://keras.io/getting-started/sequential-model-guide/

[10]:
```
from tensorflow import keras

model = keras.models.Sequential([
keras.layers.Dense(300, input_shape=inputs[0].shape),
keras.layers.Activation('relu'),
keras.layers.Dense(20),
keras.layers.Activation('sigmoid'),
])

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[11]:
```
## Then train the model on ```inputs``` and ```outputs```
model.fit(inputs,outputs)
```

```
Train on 271049 samples
271049/271049 [==============================] - 17s 63us/sample - loss: 0.0870
- accuracy: 0.9691
```

[11]:
```
<tensorflow.python.keras.callbacks.History at 0x221b7cdd0>
```

Complete the function predict: output the list of occupations where the corresponding neuron on the output layer of our model has a value > 0.1 For predicting with your model use **model.predict_on_batch()**

[12]:
```
def predict(model, article_name, vectorized_dataset):
    prediction = None
    ## Code here
    prediction = model.
 ↪predict_on_batch(vectorized_dataset[article_name]['vector'].reshape(1,300))
    result = []
    index = np.where(prediction > 0.1)
    for i in index[1]:
        result.append(occupations[i])
    return result

print(predict(model, 'Elvis_Presley', vectorized_training))
# should be subset of {'Q33999', 'Q177220', 'Q10800557', 'Q28389'}
```

```
['Q33999', 'Q177220', 'Q36834', 'Q10800557', 'Q639669']
```

[13]:
```
def evaluate_nn(vectorized_training, model):
    nexample = 0
    accuracy = 0.
    prediction = None
    for article_name in vectorized_training:
        prediction = predict(model, article_name, vectorized_training)
        p = frozenset(prediction)
        g = frozenset(vectorized_training[article_name]['occupations'])
        accuracy += 1.*len(p & g) / len(p | g)
        nexample += 1
    return accuracy / nexample
evaluate_nn(vectorized_training, model)
```

[13]:
```
0.6541374477249777
```

2.3 Task 3 - Your approach

Propose your own approach (extend previous examples or use original approaches) to improve the accuracy for this task. Apply it to the testing set and put the result as a json file with your submission.

```
[14]: model = keras.models.Sequential([
      keras.layers.Dense(300, input_shape=inputs[0].shape),
      keras.layers.Activation('relu'),
      keras.layers.Dense(20),
      keras.layers.Activation('sigmoid'),
      ])

      model.compile(optimizer='rmsprop',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
      model.fit(inputs, outputs, epochs=20, batch_size=256)
```

```
Train on 271049 samples
Epoch 1/20
271049/271049 [==============================] - 4s 14us/sample - loss: 0.1210 -
accuracy: 0.9599
Epoch 2/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0787 -
accuracy: 0.9712
Epoch 3/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0735 -
accuracy: 0.9732
Epoch 4/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0712 -
accuracy: 0.9742
Epoch 5/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0698 -
accuracy: 0.9747
Epoch 6/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0688 -
accuracy: 0.9751
Epoch 7/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0681 -
accuracy: 0.9754
Epoch 8/20
271049/271049 [==============================] - 4s 13us/sample - loss: 0.0675 -
accuracy: 0.9756
Epoch 9/20
271049/271049 [==============================] - 4s 13us/sample - loss: 0.0670 -
accuracy: 0.9758
Epoch 10/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0666 -
accuracy: 0.9759
Epoch 11/20
271049/271049 [==============================] - 4s 13us/sample - loss: 0.0662 -
accuracy: 0.9761
```

```
Epoch 12/20
271049/271049 [==============================] - 4s 14us/sample - loss: 0.0659 -
accuracy: 0.9762
Epoch 13/20
271049/271049 [==============================] - 3s 13us/sample - loss: 0.0655 -
accuracy: 0.9763
Epoch 14/20
271049/271049 [==============================] - 4s 14us/sample - loss: 0.0652 -
accuracy: 0.9765
Epoch 15/20
271049/271049 [==============================] - 3s 13us/sample - loss: 0.0649 -
accuracy: 0.9765
Epoch 16/20
271049/271049 [==============================] - 3s 13us/sample - loss: 0.0647 -
accuracy: 0.9767
Epoch 17/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0644 -
accuracy: 0.9768
Epoch 18/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0642 -
accuracy: 0.9768
Epoch 19/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0640 -
accuracy: 0.9770
Epoch 20/20
271049/271049 [==============================] - 3s 12us/sample - loss: 0.0637 -
accuracy: 0.9770
```

```
[14]: <tensorflow.python.keras.callbacks.History at 0x215e3fcd0>
```

```
[15]: evaluate_nn(vectorized_training, model)
```

```
[15]: 0.6837642196526171
```

IMPORTANT Output format of requested file 'results.json.gz': each line must be a json string representing a dictionnary: > `{ 'title': THE_ARTICLE_NAME, 'prediction': [THE_LIST_OF_OCCUPATIONS]}`

```
[16]: # For example if testset_solutions is a dictionnary: article_name (key) ->␣
       ↪prediction_list (value) use this function:
      def export(testset_solutions):
          with gzip.open('results.json.gz', 'wt') as output:
              for article in testset_solutions:
                  output.write(json.dumps({'title':article, 'prediction':
       ↪testset_solutions[article]}) + "\n")
```

[18]:
```
testset_solutions = {}
for article in vectorized_testing:
    testset_solutions[article] = predict(model, article, vectorized_testing)
```

[19]:
```
export(testset_solutions)
```