Lab4_students

January 14, 2019

1 Data import

1.1 Question 0 - Get common wikidata occupations

Write a sparql query that retrieves the top 100 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

```
In [1]: query = """
        SELECT ?o WHERE { ?p wdt:P106 ?o. } GROUP BY ?o ORDER BY DESC(COUNT(?p)) LIMIT 100
        """
```

The following assertion should pass if your answer is correct.

```
In [4]: import requests

        occupations = ['Q82955', 'Q937857', 'Q36180', 'Q33999', 'Q1650915', 'Q1028181', 'Q1930

        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

```
In [5]: 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/',

    print(occupations_label)
```

```
{'Q82955': 'politician', 'Q121594': 'professor', 'Q177220': 'singer', 'Q169470': 'physicist',
```

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

```
In [6]: 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/',

        print(occupations_labels)
```

```
{'Q82955': ['politician', 'political leader', 'polit.', 'political figure'], 'Q121594': ['profe
```

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.

```
In [7]: 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 below to evaluate the accuracy of such approach. It will serve as a baseline.

```
In [8]: def predict_dictionnary(example, occupations_labels):
            ## example['summary'] contains the summary of the article
            ## Code here
            summary = example['summary'].lower()
            hits = set()
            for (k,v) in occupations_labels.items():
                for vv in v:
                    if summary.find(vv)!= -1:
                        hits.add(k)
            return hits

        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)
```

```
Out[8]: 0.4739681674799319
```

2.2 Task 2 - Simple neural network

We load the articles "summary" and we take 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://s3-us-west-1.amazonaws.com/fasttext-vectors/cc.en.300.vec.gz
python3 -m spacy init-model en /tmp/en_vectors_wiki_lg --vectors-loc cc.en.300.vec.gz
rm cc.en.300.vec.gz
```

```
In [16]: import spacy
         nlp = spacy.load('/tmp/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
```

```
In [17]: print(vectorized_training['George_Washington']['vector'])
```

```
[-1.45162819e-02 -2.45802402e-02 -4.59302496e-03 -4.09372151e-02
 -4.47662771e-02 -4.18604538e-03 -3.15232435e-03 -1.44802360e-02
 -1.68499984e-02 -3.69651243e-03 -1.16255814e-02  1.43651171e-02
  2.02674349e-03 -5.88953542e-03 -2.17011590e-02  1.02302311e-02
 -2.49313917e-02 -5.65232616e-03 -2.25581434e-02  8.29069968e-03
 -1.44069805e-03  2.25197673e-02 -6.81395701e-04 -1.37232570e-02
 -1.26674427e-02 -3.35569866e-02  1.10627888e-02 -2.37208814e-03
 -2.30000000e-02  7.58616179e-02 -5.03487710e-04 -2.51116175e-02
  9.26511642e-03 -2.52558179e-02 -1.51058156e-02 -9.51627828e-03
  1.17523270e-02  1.22441910e-03  1.08139520e-03  3.39302444e-03
  2.20116391e-03  1.46860480e-02 -1.43686021e-02  5.76395402e-03
  1.74162779e-02 -4.76220921e-02 -1.72569733e-02 -1.49988411e-02
 -1.77732538e-02  1.58907007e-02 -7.23255938e-03  2.43825577e-02
 -2.73104683e-02 -3.67430188e-02 -1.48802334e-02 -1.34825567e-02
 -3.14348824e-02  1.95930228e-02 -6.68605033e-04 -9.24302172e-03
  1.56976283e-04 -1.65674444e-02 -1.30372085e-02  6.16298130e-05
 -3.63139645e-03  2.74534873e-03 -1.62697677e-02 -4.70697694e-03
  5.48139494e-03  4.39302297e-03  4.65523303e-02  2.29872130e-02
  2.72058025e-02 -5.52790612e-03  2.19720937e-02 -4.41581383e-02
  1.33255811e-03  1.20244222e-02  3.49267460e-02  3.76593024e-02
  8.65232572e-03 -6.52325572e-03 -1.90407019e-02  1.03569757e-02
  1.09301973e-03 -6.28488278e-03  3.98965068e-02 -3.81744131e-02
 -1.35965087e-02  1.74023230e-02 -1.48686031e-02  5.78604685e-03
 -8.59186146e-03  4.74418374e-03  1.54720917e-02 -6.42325589e-03
```

-1.58430226e-02	-2.98779178e-02	-1.54255824e-02	3.28209326e-02
2.43825577e-02	1.32907031e-03	1.80883706e-02	-2.72825565e-02
9.28488653e-03	-7.39418622e-03	-7.98023026e-03	1.84244160e-02
-9.45350039e-04	-1.16825579e-02	1.15813862e-03	-2.10464321e-04
-3.00813979e-03	4.75407019e-02	-8.32790602e-03	4.11511678e-03
-1.25604663e-02	8.92209262e-03	7.64534995e-03	-2.65965052e-02
6.58837147e-03	-1.12011610e-02	-9.68022924e-03	1.60023291e-02
1.61629519e-04	3.20906974e-02	-1.59848798e-02	1.14162825e-02
-2.40430199e-02	5.39906919e-02	-4.80814092e-03	3.02209193e-03
5.89418598e-03	-3.94418649e-03	-2.68058274e-02	-8.98256153e-03
-2.94616278e-02	3.90697829e-03	4.68255766e-03	3.96162830e-03
-2.68069748e-02	-2.68395394e-02	-9.76740339e-05	5.67557989e-03
4.43197712e-02	-1.38953477e-02	-3.69888335e-01	1.04639539e-02
1.55372089e-02	-1.35093015e-02	-8.09988379e-02	2.67802346e-02
2.21941881e-02	-7.86627829e-03	-1.00313956e-02	1.52511625e-02
1.45744160e-01	4.61395411e-03	7.26162829e-03	3.14453505e-02
-7.95465056e-03	-1.25395320e-02	6.95348764e-03	-2.48023286e-03
6.17325725e-03	1.26546472e-02	1.03558144e-02	-1.21616265e-02
-1.27907039e-03	-1.99348871e-02	-9.01860371e-03	4.25581448e-03
7.45790750e-02	1.02186035e-02	-9.93953645e-03	1.72848776e-02
-1.03779081e-02	1.46616297e-02	-3.75465187e-03	-2.26953458e-02
5.36046689e-04	6.64511696e-02	-2.53790785e-02	5.80627881e-02
-1.42732579e-02	9.22453254e-02	-1.12825576e-02	-2.51837187e-02
3.90697736e-03	5.96395321e-03	-3.02476659e-02	2.63883732e-02
-1.69488378e-02	7.39418576e-03	1.60662793e-02	-1.68313961e-02
-8.25814065e-03	-1.36965141e-02	7.30697624e-03	1.63453538e-02
-4.15407047e-02	1.05633713e-01	1.53325591e-02	6.63023209e-03
3.93279046e-02	-1.27697680e-02	-5.95697621e-03	-8.67441762e-03
1.58593040e-02	9.42093134e-03	-4.15697647e-03	1.34639572e-02
-4.10383604e-02	-2.82325619e-03	-2.43790708e-02	-4.02325485e-03
1.65058132e-02	4.21395432e-03	1.25813941e-02	1.64744183e-02
-2.81162816e-03	1.34813897e-02	-8.19302350e-03	-7.04767322e-03
1.67139638e-02	1.43581396e-02	1.20023256e-02	4.96162800e-03
1.76325571e-02	-7.07674446e-03	-4.24197726e-02	-2.34697610e-02
-1.86058115e-02	-2.32790736e-03	2.98906974e-02	1.53604464e-03
1.95941851e-02	-2.67104693e-02	-1.12453466e-02	-2.54534930e-03
-4.29302268e-03	3.56558077e-02	-4.36046888e-04	-8.16406980e-02
5.04779041e-01	-2.18813960e-02	1.15883695e-02	2.14848872e-02
7.80581404e-03	1.55116236e-02	-1.11523261e-02	4.61628864e-04
1.72918607e-02	1.43034859e-02	2.05546506e-02	-8.23488459e-03
-3.16290706e-02	-4.83953534e-03	-1.82697661e-02	2.02907110e-03
-3.51163093e-04	1.10220918e-02	-8.54755938e-02	-2.68255756e-03
1.83174424e-02	1.91116314e-02	-4.73488262e-03	-8.08255840e-03
1.37906978e-02	-7.76046468e-03	-2.82767452e-02	-2.99069774e-03
1.06569799e-02	-5.99999772e-03	1.11883730e-02	4.28720983e-03
-3.12255807e-02	-8.07186142e-02	8.59302282e-03	-8.11744668e-03
-5.36279054e-03	1.87046509e-02	-1.10972092e-01	-3.07988375e-02
9.47441999e-03	-1.03662787e-02	1.16337193e-02	3.22093032e-02

```
-2.69790720e-02  2.25430205e-02 -1.49802361e-02 -1.05290683e-02
-4.36534919e-02  6.34883530e-04 -2.83197612e-02 -1.37674408e-02
-1.50220934e-02  1.30851150e-01 -1.22430259e-02  2.38767453e-02]
```

```
In [18]: # We encode the data

         import numpy as np

         inputs = np.array([vectorized_training[article]['vector'] for article in vectorized_t
         outputs = np.array([[(1 if occupation in vectorized_training[article]['occupations'] e
                             for occupation in occupations ] for article in vectorized_training
```

```
In [19]: print(len(outputs[0]))
```

```
100
```

Using keras, define a sequential neural network with two layers. Use categorical_crossentropy as a loss function and softmax as the activation function of the output layer

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

```
In [20]: from tensorflow import keras
         ## Compile the model here
         model = keras.models.Sequential([
             keras.layers.Dense(200, input_shape=(300,)),
             keras.layers.Activation('relu'),
             keras.layers.Dense(100),
             keras.layers.Activation('softmax'),
         ])
         model.compile(optimizer='rmsprop',
                       loss='categorical_crossentropy',
                       metrics=['accuracy'])
```

```
/Users/hchen/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Convers
  from ._conv import register_converters as _register_converters
```

```
In [21]: ## Then train the model on ```inputs``` and ```outputs```
         model.fit(inputs,outputs)
```

```
Epoch 1/1
427798/427798 [==============================] - 16s 37us/step - loss: 2.4468 - acc: 0.6529
```

```
Out[21]: <tensorflow.python.keras.callbacks.History at 0x152113710>
```

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

```
In [22]: def predict(model, article_name, vectorized_dataset):
             prediction = []
             ## Code here
             result = model.predict(vectorized_dataset[article_name]['vector'].reshape(1,300))
             for i in range(0,len(result)):
                 if result[i] > 0.1:
                     prediction.append(occupations[i])
             return prediction

         print(predict(model, 'Elvis_Presley', vectorized_training))
         # should be {'Q177220'}
```

```
['Q33999', 'Q177220']
```

```
In [24]: 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)
```

```
Out[24]: 0.6048549771493903
```

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.

```
In [33]: 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)
```

Out[33]: 0.657609295561242

In [45]:
```
import spacy
training_set_copy = training_set
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
```

In [46]:
```
import spacy
import nltk
training_set_copy = training_set
def vectorize1(dataset, nlp):
    result = {}
    for example in training_set_copy:
        summary = example['summary']
        new_sentence = ""
        tokens = nltk.word_tokenize(summary)
        t = nltk.pos_tag(tokens)
        for i in t:
            if i[1] in ["NN","NNS","NNP","NNPS"]:
                new_sentence += i[0]
                new_sentence += " "
        example['summary'] = new_sentence
        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_training1 = vectorize1(training_set_copy, nlp)
```

In [47]:
```
import numpy as np

inputs1 = np.array([vectorized_training1[article]['vector'] for article in vectorized_
outputs1 = np.array([[(1 if occupation in vectorized_training1[article]['occupations']
                     for occupation in occupations ] for article in vectorized_trainin
```

In [48]:
```
from tensorflow import keras
model1 = keras.models.Sequential([
    keras.layers.Dense(601, input_shape=(300,)),
```

```
        keras.layers.Activation('relu'),
        keras.layers.Dense(100),
        keras.layers.Activation('softmax'),
    ])
    model1.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

In [49]:
```
model1.fit(inputs1,outputs1)
```

```
Epoch 1/1
427798/427798 [==============================] - 25s 58us/step - loss: 2.1117 - acc: 0.7069
```

Out[49]: <tensorflow.python.keras.callbacks.History at 0x129478940>

In [56]:
```
def predict(model, article_name, vectorized_dataset):
    prediction = []
    ## Code here
    result = model.predict(vectorized_dataset[article_name]['vector'].reshape(1,300))
    max_r = 0
    max_name = ""
    for i in range(0,len(result)):
        if result[i] > max_r:
            max_r = result[i]
            max_name = occupations[i]
        if result[i] > 0.2:
            prediction.append(occupations[i])
    if len(prediction) == 0:
        prediction.append(max_name)
    return prediction
```

In [57]:
```
testset_solutions = {}
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)
        testset_solutions[article_name] = prediction
        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_training1, model1)
```

Out[57]: 0.6979087873507037

In [58]:
```
testset_solutions = {}
for article_name in vectorized_testing:
        prediction = predict(model, article_name, vectorized_testing)
        testset_solutions[article_name] = prediction
```

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]}

In [59]:
```
# For example if testset_solutions is a dictionnary: article_name (key) -> prediction
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[a
```

In [60]:
```
export(testset_solutions)
```