CS 6220 Big Data Homework 2 Report – Problem 2

By: Nipun Chhajed

Problem 1 – Representational Learning with Word2Vec

Introduction:

- For this problem I utilized the gensim word2vec model (https://radimrehurek.com/gensim/models/word2vec.html) and trained it on a JSON dataset of recipes which can be found in this example notebook on Kaggle called ***Word2Vec with Ingredients***: https://www.kaggle.com/code/ccorbi/word2vec-with-ingredients/notebook. The dataset is originally from the Kaggle competition ***What's Cooking?***: https://www.kaggle.com/competitions/whats-cooking.
- The dataset is named "train.json" on in the Kaggle notebook, however I have renamed it to "recipes.json" and the dataset has been uploaded in the zip file alongside my Jupyter notebook.
- I also utilized code from the ***Word2Vec with Ingredients*** notebook for data preprocessing and model training. I also utilized code from a Kaggle notebook called ***Gensim Word2Vec Tutorial***: https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.kaggle.com%2Fcode%2Fpierremegret%2Fgensim-word2vec-tutorial and the code I used was specifically the code to build the vocabulary and train the model with epochs.
- All the code that I used from those two notebooks has been labelled in the corresponding cells in my Jupyter notebook that had all my code.
- I also wrote my own function called **topWordsCalculations()** that handled all the top-n word calculations that needed to be done for each model, the code for that can be found in the notebook.
- All the model outputs and top-n word outputs can be found in the notebook.
- Instructions to run the notebook can be found in the README file. I ran the code in Google Colab, but I also included instructions for running the notebook locally.

Training:

All my program outputs can be seen in the jupyter notebook; however, I have included relevant screenshots and such in this report.

Due to this bug with gensim loss reporting not working properly: https://github.com/RaRe-Technologies/gensim/issues/2617, I was unable to report loss when training the gensim word2vec model. However, I did measure the runtime of each of my models.

I essentially trained 4 different word2vec models:

- **Model 1** – Base model that was given in the ***Word2Vec with Ingredients*** notebook:

  ▾ Word2Vec Model 1

```
[237] # code taken from:  https://www.kaggle.com/code/ccorbi/word2vec-with-ingredients/notebook (word2vec with ingredients)

      # Set values for NN parameters
      num_features = 300    # Word vector dimensionality
      min_word_count = 3    # 50% of the corpus
      num_workers = 4       # Number of CPUs
      context = 10          # window size
      downsampling = 1e-3   # threshold for configuring which
                            # higher-frequency words are randomly downsampled

      # Initialize and train the model
      model = word2vec.Word2Vec(workers=num_workers, \
                  vector_size=num_features, min_count = min_word_count, \
                  window = context, sample = downsampling)
```

- **Model 2** – Decreased the window size by ½ (from window size of 10 to window size of 5):

  ▾ Word2Vec Model 2 - Decreased Window Size by 1/2

```
[242] # code taken from:  https://www.kaggle.com/code/ccorbi/word2vec-with-ingredients/notebook (word2vec with ingredients)

      # Set values for NN parameters
      num_features = 300    # Word vector dimensionality
      min_word_count = 3    # 50% of the corpus
      num_workers = 4       # Number of CPUs
      context = 5          # window size
      downsampling = 1e-3   # threshold for configuring which
                            # higher-frequency words are randomly downsampled

      # Initialize and train the model
      model2 = word2vec.Word2Vec(workers=num_workers, \
                  vector_size=num_features, min_count = min_word_count, \
                  window = context, sample = downsampling)
```

- **Model 3** – Added learning rate parameter (learning rate of 0.005), went back to window size of 10:

  ▾ Word2Vec Model 3 - Added learning rate of 0.005

```
[247] # code taken from:  https://www.kaggle.com/code/ccorbi/word2vec-with-ingredients/notebook (word2vec with ingredients)

      # Set values for NN parameters
      num_features = 300    # Word vector dimensionality
      min_word_count = 3    # 50% of the corpus
      num_workers = 4       # Number of CPUs
      context = 10          # window size
      downsampling = 1e-3   # threshold for configuring which
                            # higher-frequency words are randomly downsampled

      # Initialize and train the model
      model3 = word2vec.Word2Vec(workers=num_workers, \
                  vector_size=num_features, min_count = min_word_count, \
                  window = context, sample = downsampling, alpha=0.005)
```

- **Model 4** – Decreased the window size (window size = 5) and added learning rate (learning rate = 0.005), basically a combination of model 2 & 3:

Word2Vec Model 4 - Decreased Window Size (window = 5), Learning Rate Added (lr = 0.005)

```
[252] # code taken from:  https://www.kaggle.com/code/ccorbi/word2vec-with-ingredients/notebook (word2vec with ingredients)

      # Set values for NN parameters
      num_features = 300    # Word vector dimensionality
      min_word_count = 3    # 50% of the corpus
      num_workers = 4       # Number of CPUs
      context = 5          # window size
      downsampling = 1e-3   # threshold for configuring which
                            # higher-frequency words are randomly downsampled

      # Initialize and train the model
      model4 = word2vec.Word2Vec(workers=num_workers, \
                  vector_size=num_features, min_count = min_word_count, \
                  window = context, sample = downsampling, alpha=0.005)
```

Each of the models was trained for 10 epochs and in the table below you can see the training time for each model. For each model I also built the vocabulary based on the recipe ingredients, and then trained it.

Training Time:

Model	Vocab Building Time	Training Time
Model 1 – Base model that was given in the ***Word2Vec with Ingredients*** notebook	0.1715 seconds	9.646 seconds
Model 2 – Decreased the window size by ½ (from window size of 10 to window size of 5)	0.2367 seconds	7.734 seconds
Model 3 – Added learning rate parameter (learning rate of 0.005), went back to window size of 10	0.2329 seconds	10.1708 seconds
Model 4 – Decreased the window size (window size = 5) and added learning rate (learning rate = 0.005), basically a combination of model 2 & 3	0.225 seconds	9.091 seconds

As can be seen from the table, the models all had around the same training time of 7 – 10 seconds, and each took roughly the same amount of time to build the vocab, between 0.15 – 0.24 seconds.

Testing:

- For the testing, I basically tested each model on these 5 words individually (which are ingredients): **fresh basil, cilantro, kalamata olives, black beans, and garam masala** (an indian spice). For each of the words, I found the top-1, top-5, and top-10 most similar words, and found the average cosine similarity score for each top-n result, by passing them into my method that I wrote - **topWordsCalculations()**. I did this for all 4 models. I also gave the models a list of ingredients: **[eggs, milk, vanilla extract]** to see what kind of top 10 most similar words it would output for those 3 ingredients together.

(RESULTS OF EACH MODEL ON NEXT PAGES)

Model 1 Results:

```
In [240]: topWordsCalculations(model, test_words)
```

```
word= fresh basil
output: [('fresh basil leaves', 0.7428125143051147), ('basil leaves', 0.7322972416877747), ('chees fresh mozzarella', 0.6709
271669387817), ('penne', 0.6659830808639526), ('Italian turkey sausage', 0.6659452319145203), ('olive oil flavored cooking s
pray', 0.647891104221344), ('bow-tie pasta', 0.6407570838928223), ('pesto', 0.6385067701339722), ('yellow squash', 0.6304804
086685181), ('spinach leaves', 0.6240914463996887)]
top-1 average: 0.7428125143051147
top-5 average: 0.6955930471420289
top-10 average: 0.6659692049026489


word= cilantro
output: [('fresh cilantro', 0.8856534361839294), ('cilantro leaves', 0.858186662197113), ('cilantro fresh', 0.71678727865219
12), ('fresh coriander', 0.6562798023223877), ('cilantro stems', 0.6503143310546875), ('sweet corn', 0.5807778239250183),
('serrano peppers', 0.5773773789405823), ('chile', 0.5482127666473389), ('hot chili', 0.5370160937309265), ('Mexican oregan
o', 0.5146968960762024)]
top-1 average: 0.8856534361839294
top-5 average: 0.7534443020820618
top-10 average: 0.6525302469730377


word= kalamata olives
output: [('extra fine granulated sugar', 0.9092204570770264), ('grass-fed butter', 0.9065915942192078), ('ouzo', 0.885994553
565979), ('globe eggplant', 0.8858441114425659), ('assorted fresh vegetables', 0.8842945098876953), ('pita pockets', 0.87846
3625907898), ('branzino', 0.8783990144729614), ('red russian kale', 0.8746089339256287), ('enriched white rice', 0.872351109
9815369), ('loaves', 0.8691681623458862)]
top-1 average: 0.9092204570770264
top-5 average: 0.8943890452384948
top-10 average: 0.8844936072826386


word= black beans
output: [('canned black beans', 0.8852002024650574), ('pinto beans', 0.838304877281189), ('frozen corn', 0.826698482036590
6), ('poblano peppers', 0.8193145394325256), ('whole wheat tortillas', 0.808560848236084), ('chipotle peppers', 0.8071742057
800293), ('guacamole', 0.8059782385826111), ('chipotle chile pepper', 0.7915951013565063), ('tortilla chips', 0.786643564701
0803), ('tortillas', 0.7674216032028198)]
top-1 average: 0.8852002024650574
top-5 average: 0.8356157898902893
top-10 average: 0.8136891663074494


word= garam masala
output: [('tumeric', 0.8292970657348633), ('coriander', 0.8267992734909058), ('masala', 0.807604968547821), ('turmeric', 0.8
073639273643494), ('basmati rice', 0.8043537139892578), ('yoghurt', 0.7899531722068787), ('whole milk yoghurt', 0.7791794538
497925), ('fenugreek leaves', 0.7783532738685608), ('fenugreek', 0.7642073035240173), ('red lentils', 0.7571282386779785)]
top-1 average: 0.8292970657348633
top-5 average: 0.8150837898254395
top-10 average: 0.7944240391254425
```

```
model.wv.most_similar(['eggs', 'milk', 'vanilla extract'])
```

```
]: [('evaporated milk', 0.8159870505332947),
    ('pastry', 0.7985823750495911),
    ('softened butter', 0.7826281189918518),
    ('beaten eggs', 0.780570924282074),
    ('melted butter', 0.772914707660675),
    ('semi-sweet chocolate morsels', 0.7638320922851562),
    ('shortening', 0.7602747082710266),
    ('single crust pie', 0.7598553895950317),
    ('pie crust', 0.7565968632698059),
    ('whole wheat pastry flour', 0.7527552247047424)]
```

Model 2 Results:

```
topWordsCalculations(model2, test_words)
```

```
word= fresh basil
output: [('fresh basil leaves', 0.7621052861213684), ('basil leaves', 0.7479035258293152), ('linguine', 0.7066890001296997),
('bow-tie pasta', 0.7018566131591797), ('chees fresh mozzarella', 0.6827179789543152), ('penne', 0.6805809140205383), ('pest
o', 0.662975549697876), ('yellow squash', 0.6602261066436768), ('spinach leaves', 0.6521127820014954), ('fusilli', 0.6481409
668922424)]
top-1 average: 0.7621052861213684
top-5 average: 0.7202544808387756
top-10 average: 0.6905308723449707


word= cilantro
output: [('fresh cilantro', 0.8399322032928467), ('cilantro leaves', 0.8198192715644836), ('cilantro fresh', 0.6991016268730
164), ('chile', 0.6756662130355835), ('serrano peppers', 0.6597104668617249), ('cooked brown rice', 0.6166772842407227), ('c
ilantro stems', 0.606759786605835), ('fresh coriander', 0.6028186082839966), ('hot chili', 0.6027405261993408), ('sweet cor
n', 0.5893930792808533)]
top-1 average: 0.8399322032928467
top-5 average: 0.738845956325531
top-10 average: 0.6712619066238403


word= kalamata olives
output: [('tuna packed in water', 0.9445316195487976), ('flat anchovy', 0.9378742575645447), ('hot cherry pepper', 0.9252677
5598526), ('pita chips', 0.9172925353050232), ('country loaf', 0.9148755073547363), ('pita rounds', 0.9143105149269104), ('p
eperoncini', 0.9113659858703613), ('balsamic vinaigrette', 0.9021040201187134), ('tapenade', 0.8968738913536072), ('bocconci
ni', 0.8945508003234863)]
top-1 average: 0.9445316195487976
top-5 average: 0.9279683351516723
top-10 average: 0.9159046888351441


word= black beans
output: [('canned black beans', 0.9246068000793457), ('poblano peppers', 0.8699742555618286), ('salsa verde', 0.850710809230
8044), ('whole wheat tortillas', 0.8484592437744141), ('chipotle chile pepper', 0.8398043513298035), ('guacamole', 0.8397717
475891113), ('Mexican cheese blend', 0.8382490277290344), ('pinto beans', 0.8378378748893738), ('pepper jack', 0.83246225118
63708), ('taco shells', 0.8302130699157715)]
top-1 average: 0.9246068000793457
top-5 average: 0.8667110919952392
top-10 average: 0.8512089431285859


word= garam masala
output: [('masala', 0.8287578821182251), ('coriander', 0.8212893009185791), ('fenugreek leaves', 0.8117449879646301), ('basm
ati rice', 0.8093999624252319), ('fenugreek', 0.8081806898117065), ('tumeric', 0.8081257939338684), ('red lentils', 0.804702
1627426147), ('turmeric', 0.8042529225349426), ('yoghurt', 0.8033658266067505), ('fenugreek seeds', 0.7943703532218933)]
top-1 average: 0.8287578821182251
top-5 average: 0.8158745646476746
top-10 average: 0.8094189882278442
```

```
model2.wv.most_similar(['eggs', 'milk', 'vanilla extract'])
```

```
]: [('low-fat milk', 0.8713571429252625),
    ('evaporated milk', 0.8594532012939453),
    ('melted butter', 0.8495365381240845),
    ('softened butter', 0.8274590373039246),
    ('pastry', 0.8174067735671997),
    ('shortening', 0.8144609332084656),
    ('single crust pie', 0.7883224487304688),
    ('medium eggs', 0.7871817350387573),
    ('lemon extract', 0.7867909669876099),
    ('golden syrup', 0.7859845757484436)]
```

Model 3 Results:

```
topWordsCalculations(model3, test_words)
```

```
word= fresh basil
output: [('balsamic vinegar', 0.9886016249656677), ('fresh basil leaves', 0.9833713173866272), ('freshly  pepper', 0.9761232
733726501), ('linguine', 0.9750982522964478), ('capers', 0.9738101959228516), ('pitted kalamata olives', 0.973563373088836
7), ('fresh oregano', 0.9724856615066528), ('spaghetti', 0.9699956774711609), ('extra-virgin olive oil', 0.966441512107849
1), ('fat free less sodium chicken broth', 0.966299295425415)]
top-1 average: 0.9886016249656677
top-5 average: 0.9794009327888489
top-10 average: 0.9745790183544158


word= cilantro
output: [('fresh cilantro', 0.9871524572372437), ('lime', 0.9787974953651428), ('lime juice', 0.9727945327758789), ('lime we
dges', 0.9701160788536072), ('cilantro fresh', 0.9577324986457825), ('cilantro leaves', 0.954762876033783), ('chicken breast
s', 0.9540164470672607), ('jalapeno chilies', 0.9519700407981873), ('white onion', 0.9512802362442017), ('chipotle paste',
0.9457230567932129)]
top-1 average: 0.9871524572372437
top-5 average: 0.973318612575531
top-10 average: 0.96243457198143


word= kalamata olives
output: [('dry red wine', 0.9194468259811401), ('dried rosemary', 0.9148616194725037), ('parsley sprigs', 0.914289355278015
1), ('low salt chicken broth', 0.9141350388526917), ('italian plum tomatoes', 0.9134431481361389), ('red wine', 0.9131529927
253723), ('fennel bulb', 0.9128929972648621), ('orzo pasta', 0.9115782380104065), ('white wine', 0.9105476140975952), ('dita
lini', 0.9105328321456909)]
top-1 average: 0.9194468259811401
top-5 average: 0.9152351975440979
top-10 average: 0.9134880661964416


word= black beans
output: [('corn tortillas', 0.9929153323173523), ('tortilla chips', 0.9928678274154663), ('guacamole', 0.9899529218673706),
('cotija', 0.9861535429954529), ('corn', 0.9858227372169495), ('flour tortillas', 0.9850752949714661), ('salsa', 0.984722912
311554), ('enchilada sauce', 0.9845128655433655), ('green chile', 0.984503984451294), ('tortillas', 0.9817508459091187)]
top-1 average: 0.9929153323173523
top-5 average: 0.9895424723625184
top-10 average: 0.9868278264999389


word= garam masala
output: [('tumeric', 0.9976426362991333), ('turmeric', 0.9975494742393494), ('coriander', 0.996055543422699), ('cumin seed',
0.9916380643844604), ('basmati rice', 0.976142942905426), ('clove', 0.9719006419181824), ('mustard seeds', 0.967122852802276
6), ('coriander seeds', 0.9644714593887329), ('garlic paste', 0.964306652545929), ('green chilies', 0.9626431465148926)]
top-1 average: 0.9976426362991333
top-5 average: 0.9918057322502136
top-10 average: 0.9789473414421082
```

```
model3.wv.most_similar(['eggs', 'milk', 'vanilla extract'])
```

```
[('buttermilk', 0.9875501990318298),
 ('unbaked pie crusts', 0.982410192489624),
 ('baking', 0.9808518886566162),
 ('shortening', 0.980090320110321),
 ('melted butter', 0.9729699492454529),
 ('anise extract', 0.9702609181404114),
 ('cornmeal', 0.9696202278137207),
 ('evaporated milk', 0.9695850610733032),
 ('farmer cheese', 0.9665543437004089),
 ('self rising flour', 0.9648932218551636)]
```

Model 4 Results:

```
topWordsCalculations(model4, test_words)
```

```
word= fresh basil
output: [('balsamic vinegar', 0.9876619577407837), ('extra-virgin olive oil', 0.9849947094917297), ('capers', 0.9828872680664062), ('fresh basil leaves', 0.979954719543457), ('fresh oregano', 0.9786477088928223), ('linguine', 0.9735621213912964), ('arborio rice', 0.9734840393066406), ('spaghetti', 0.9728142023086548), ('flat leaf parsley', 0.9714289307594299), ('freshly  pepper', 0.9708184003829956)]
top-1 average: 0.9876619577407837
top-5 average: 0.9828292727470398
top-10 average: 0.9776254057884216


word= cilantro
output: [('fresh cilantro', 0.9888460040092468), ('lime', 0.9884642362594604), ('lime juice', 0.9845849275588989), ('lime wedges', 0.9755030870437622), ('cilantro fresh', 0.9707152247428894), ('chicken breasts', 0.9647049903869629), ('jalapeno chilies', 0.9620205163955688), ('cilantro leaves', 0.9571397304534912), ('white onion', 0.9512550234794617), ('boneless skinless chicken breasts', 0.9496047496795654)]
top-1 average: 0.9888460040092468
top-5 average: 0.9816226959228516
top-10 average: 0.9692838490009308


word= kalamata olives
output: [('pasta', 0.9271238446235657), ('penne pasta', 0.9267390370368958), ('cannellini beans', 0.9247123003005981), ('italian sausage', 0.9239885210990906), ('sweet italian sausage', 0.9229277968406677), ('kalamata', 0.9219010472297668), ('spaghetti', 0.9211446642875671), ('italian seasoning', 0.9210634827613831), ('part-skim mozzarella cheese', 0.9210060238838196), ('finely  onion', 0.9209407567977905)]
top-1 average: 0.9271238446235657
top-5 average: 0.9250982999801636
top-10 average: 0.9231547474861145


word= black beans
output: [('corn tortillas', 0.9968032240867615), ('flour tortillas', 0.9908587336540222), ('tortilla chips', 0.9901202321052551), ('salsa', 0.9884030818939209), ('jalapeno chilies', 0.9873879551887512), ('avocado', 0.9855659008026123), ('enchilada sauce', 0.9847285151481628), ('taco seasoning', 0.984626293182373), ('refried beans', 0.984143078327179), ('shredded Montere y Jack cheese', 0.9828854203224182)]
top-1 average: 0.9968032240867615
top-5 average: 0.9907146453857422
top-10 average: 0.9875522434711457


word= garam masala
output: [('coriander', 0.9978469014167786), ('turmeric', 0.9975991249084473), ('tumeric', 0.9956970810890198), ('cumin seed', 0.9944854378700256), ('green chilies', 0.973973274230957), ('garlic paste', 0.9645842909812927), ('clove', 0.9640976786613464), ('mustard seeds', 0.9541863799095154), ('curry leaves', 0.9514783620834351), ('coriander seeds', 0.946915864944458)]
top-1 average: 0.9978469014167786
top-5 average: 0.9919203639030456
top-10 average: 0.9740864396095276
```

```
model4.wv.most_similar(['eggs', 'milk', 'vanilla extract'])
```

```
[('buttermilk', 0.9928349256515503),
 ('shortening', 0.9897841811180115),
 ('baking', 0.9897537231445312),
 ('cornmeal', 0.9829892516136169),
 ('pecans', 0.9821597933769226),
 ('baking soda', 0.9802330136299133),
 ('vanilla', 0.9801535606384277),
 ('melted butter', 0.9798804521560669),
 ('white cornmeal', 0.9791784882545471),
 ('confectioners sugar', 0.9790623188018799)]
```

Analysis:

- As can be seen from the screenshots of the results, we see a general trend that as we tuned the hyperparameters from model 1 to model 4, the cosine similarity of the test words and the list of words together increased.
- Also, when you look at the actual predictions for each of the ingredients (**fresh basil, cilantro, kalamata olives, black beans, and garam masala**), one can see that the similar words that the models output gets better from models 1 – model 4. For example, in model 1, for **kalamata olives** the predictions of the similar words were not really that accurate, because kalamata olives is used in generally Greek cuisine, however the outputs seemed to be more random that were not related to Greek cuisine. In model 4 however, the outputted similar words for kalamata olives were much closer to stuff that is used in Greek cuisine, and thus showing the improvement made from model to model.
- In addition, for the 3 ingredients together: **[eggs, milk, vanilla extract]**, the predictions for each model seemed reasonable. I specifically chose those ingredients because I wanted to see if the similar words outputted by the models would be related to baking, and for the most part they were.
- I believe that reducing the window size and adding the high learning rate helped the model do better because a higher learning rate tends to help models to converge faster, and the smaller window size helped the model learn more about each word individually, which helps it find more functionally similar words. The larger window size caused the model to be broader, and this can be seen in the **kalamata olives** example, as model 1 was very widespread in the similar word predictions, while model 4 was outputting words that were more similar to recipes that would contain **kalamata olives**.
- I think that further experimentation by reducing the window size even further could help the model more, although, if the window size gets too small (1-2) there could be overfitting that occurs.

Experience w/ Gensim:

- Overall, this project was interesting, and Gensim was not too hard to figure out how to use. I also didn't have to install much because I just ran the code on Google Colab, and all my imported libraries were already preinstalled on Google Colab.
- Gensim made it easy to initialize and train word2vec models.
- Also, the dataset itself was pretty clean, all that needed to be done was parse the JSON and do a little bit of cleaning.