Documentation:

Corpus Codes:

- ‘0_1’: LibEST
- ‘1_1’: EBT
- ‘2_0’: eTour
- ‘3_0’: iTrust
- ‘4_0’: Albergate
- ‘5_0’: SMOS

Create corpus by calling Corpus.get_present_corpus(‘corpus code of desired corpus’)
Generate the VSM by calling VSM(corpus)
Generate the model by calling vsm_generator.generate_model()

doc2vec Tests:

1. simple_test
2. use_negative_test
3. preprocessing_test
4. vector_size_test
5. epochs_test
6. shared_vocab_test

Tests in Detail:

- simple_test
  - For the simple_test, no parameters such as preprocessing, epochs, or similar vocab as these are addressed in separate tests
  - First create a doc2vec_generator by calling Doc2Vec_IR(corpus)
  - (corpus is generated above)
  - Then create doc2vec_model by calling doc2vec_generator.generate_model()
  - Create evaluator by calling Evaluate on the vsm_model and doc2vec_model
  - Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances
- use_negative_test
  - For the used_negative_test, tests between using_negative and not using negatives
  - First create a doc2vec_generator by calling Doc2Vec_IR(corpus)
  - (corpus is generated above)
  - Create doc2vec_model_0 by calling doc2vec_generator.generate_model(parameter use_negatives set to False)
  - Create doc2vec_model_1 by calling doc2vec_generator.generate_model(parameter use_negatives set to True)

- Create evaluator by calling Evaluate on the vsm_model, doc2vec_model_0, and doc2vec_model_1
- Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances

▪ preprocessing_test

- For the prepresocessing_test, preprocessing parameters are used. One test where no preprocessing techniques: only_alphnum, only_alph, split_camel_case, split_snake_case, remove_stop_words, and stem are all turned off (set to False). One test where all preprocessing techniques: only_alphnum, split_camel_case, split_snake_case, remove_stop_words, and stem are all active (set to True)
- Create doc2vec_generator0 by calling Doc2Vec_IR(corpus, preprocessing techniques set to False)
- (corpus is generated above)
- Create doc2vec_generator1 by calling Doc2Vec_Ir(corpus, preprocessing techniques set to True)
- (corpus is generated above)
- Create doc2vec_model_0 by calling doc2vec_generator0.generate_model()
- Create doc2vec_model_1 by calling doc2vec_generator1.generate_model()
- Create evaluator by calling Evaluate on the vsm_model, doc2vec_model_0, and doc2vec_model_1
- Then call evaluator.precision_recal(show_parameters=False, show_random_model=True) to graph the models and performances

▪ vector_size_test

- For the vector_size_test, different sized vectors are used to test different vector sizes. Sizes of 200, 300, and 400 were used
- Create doc2vec_generator by calling Doc2Vec_IR(corpus)
- (corpus is generated above)
- Create doc2vec_model_0 by calling doc2vec_generator.generate_model(parameter of vector size 200)
- Create doc2vec_model_1 by calling doc2vec_generator.generate_model(parameter of vector size 300)
- Create doc2vec_model_2 by calling doc2vec_generator.generate_model(parameter of vector size 400)
- Create evaluator by calling Evaluate on the vsm_model, doc2vec_model_0, doc2vec_model_1, and doc2vec_model_2
- Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances

▪ epochs_test

- For the epochs_test, different sized epochs are used. Epochs of size 25, 50, 75, and 100 were used
- Create doc2vec_generator by calling Doc2Vec_IR(corpus)
- (corpus is generated above)

- Create doc2vec_model_0 by calling doc2vec_generator.generate_model(parameter of epoch size 25)
- Create doc2vec_model_1 by calling doc2vec_generator.generate_model(parameter of epoch size 50)
- Create doc2vec_model_2 by calling doc2vec_generator.generate_model(parameter of epoch size 75)
- Create doc2vec_model_3 by calling doc2vec_generator.generate_model(parameter of epoch size 100)
- Create evaluator by calling Evaluate on the vsm_model, doc2vec_model_0, doc2vec_model_1, doc2vec_model_2, and doc2vec_model_3
- Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances

▪ shared_vocab_test

- The shared_vocab_test tested against shared vocab. The parameter only_common_vocab by default is True, which is used in 1 case, and only_common_vocab set to False in another case
- Create doc2vec_generator0 by calling Doc2Vec_IR(corpus)
- (corpus is generated above)
- Create doc2vec_generator1 by calling doc2vec_generator.generate_model(parameter of only_common_vocab = False)
- Create doc2vec_model0 by calling doc2vec_generator0.generate_model()
- Create doc2vec_model1 by calling doc2vec_generator1.generate_model()
- Create evaluator by calling Evaluate on the vsm_model, doc2vec_model_0 and doc2vec_model_1
- Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances

word2vec Tests:

1. simple_test_sg
2. simple_test_cbow
3. vector_size_test_sg
4. vector_size_test_cbow
5. iter_test_sg
6. iter_test_cbow
7. preprocessing_test_sg
8. preprocessing_test_cbow

Tests in Detail:

- simple_test_sg
  - For the simple_test_sg no parameters are used, and the skip gram method is used. Skip gram(sg) is the default method
  - Create the word2vec_generator by calling Word2Vec_IR(corpus)
  - Create the word2vec_model by calling word2vec_generator.generate_model()
  - Create the evaluator by calling Evaluate on the vsm_model and word2vec_model
  - Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances
- simple_test_cbow
  - For the simple_test_sg no parameters are used, and the cbow method is used
  - Create the word2vec_generator by calling Word2Vec_IR(corpus)
  - Create the word2vec_model by calling word2vec_generator.generate_model(with the parameter sg = 0)
  - sg = 0 makes the cbow method active
  - Create the evaluator by calling Evaluate on the vsm_model and word2vec_model
  - Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances
- vector_size_test_sg
  - For the vector_size_test, different sized vectors are used to test different vector sizes. Sizes of 200, 300, and 400 were used and the sg method was used
  - Create word2vec_generator by calling Word2Vec_IR(corpus)
  - (corpus is generated above)
  - Create word2vec_model_0 by calling word2vec_generator.generate_model(parameter of vector size 200)
  - Create word2vec_model_1 by calling word2vec_generator.generate_model(parameter of vector size 300)
  - Create word2vec_model_2 by calling word2vec_generator.generate_model(parameter of vector size 400)
  - Create evaluator by calling Evaluate on the vsm_model, word2vec_model_0, word2vec_model_1, and word2vec_model_2

- Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances

▪ vector_size_test_cbow

- For the vector_size_test, different sized vectors are used to test different vector sizes. Sizes of 200, 300, and 400 were used and the cbow method was used
- Create word2vec_generator by calling Word2Vec_IR(corpus)
- (corpus is generated above)
- Create word2vec_model_0 by calling word2vec_generator.generate_model(parameter of vector size 200 and sg = 0)
- Create word2vec_model_1 by calling word2vec_generator.generate_model(parameter of vector size 300 and sg = 0)
- Create word2vec_model_2 by calling word2vec_generator.generate_model(parameter of vector size 400 and sg = 0)
- Create evaluator by calling Evaluate on the vsm_model, word2vec_model_0, word2vec_model_1, and word2vec_model_2
- Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances

▪ iter_test_sg

- For the iter_test, different sized iterations are used. Iterations of size 25, 50, 75, and 100 were used and the sg method was used
- Create word2vec_generator by calling Word2Vec_IR(corpus)
- (corpus is generated above)
- Create word2vec_model_0 by calling word2vec_generator.generate_model(parameter of iter size 25)
- Create word2vec_model_1 by calling word2vec_generator.generate_model(parameter of iter size 50)
- Create word2vec_model_2 by calling word2vec_generator.generate_model(parameter of iter size 75)
- Create word2vec_model_3 by calling word2vec_generator.generate_model(parameter of iter size 100)
- Create evaluator by calling Evaluate on the vsm_model, word2vec_model_0, word2vec_model_1, word2vec_model_2, and word2vec_model_3
- Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances

▪ iter_test_cbow

- For the iter_test, different sized iterations are used. Iterations of size 25, 50, 75, and 100 were used and the cbow method was used
- Create word2vec_generator by calling Word2Vec_IR(corpus)
- (corpus is generated above)

- Create word2vec_model_0 by calling word2vec_generator.generate_model(parameter of iter size 25 and sg = 0)
- Create word2vec_model_1 by calling word2vec_generator.generate_model(parameter of iter size 50 and sg = 0)
- Create word2vec_model_2 by calling word2vec_generator.generate_model(parameter of iter size 75 and sg = 0)
- Create word2vec_model_3 by calling word2vec_generator.generate_model(parameter of iter size 100 and sg = 0)
- Create evaluator by calling Evaluate on the vsm_model, word2vec_model_0, word2vec_model_1, word2vec_model_2, and word2vec_model_3
- Then call evaluator.precision_recal(show_parameters=True, show_random_model=True) to graph the models and performances

▪ preprocessing_test_sg

- For the prepresocessing_test, preprocessing parameters are used. One test where no preprocessing techniques: only_alphnum, only_alph, split_camel_case, split_snake_case, remove_stop_words, and stem are all turned off (set to False). One test where all preprocessing techniques: only_alphnum, split_camel_case, split_snake_case, remove_stop_words, and stem are all active (set to True)
- Create word2vec_generator0 by calling Word2Vec_IR(corpus, preprocessing techniques set to False)
- (corpus is generated above)
- Create word2vec_generator1 by calling Word2Vec_Ir(corpus, preprocessing techniques set to True)
- (corpus is generated above)
- Create word2vec_model_0 by calling word2vec_generator0.generate_model()
- Create word2vec_model_1 by calling word2vec_generator1.generate_model()
- Create evaluator by calling Evaluate on the vsm_model, word2vec_model_0, and word2vec_model_1
- Then call evaluator.precision_recal(show_parameters=False, show_random_model=True) to graph the models and performances

▪ preprocessing_test_cbow

- For the prepresocessing_test, preprocessing parameters are used. One test where no preprocessing techniques: only_alphnum, only_alph, split_camel_case, split_snake_case, remove_stop_words, and stem are all turned off (set to False). One test where all preprocessing techniques: only_alphnum, split_camel_case, split_snake_case, remove_stop_words, and stem are all active (set to True)
- Create word2vec_generator0 by calling Word2Vec_IR(corpus, preprocessing techniques set to False)
- (corpus is generated above)
- Create word2vec_generator1 by calling Word2Vec_Ir(corpus, preprocessing techniques set to True)

- (corpus is generated above)
- Create word2vec_model_0 by calling
  word2vec_generator0.generate_model(parameter sg = 0)
- Create word2vec_model_1 by calling
  word2vec_generator1.generate_model(parameter sg = 0)
- Create evaluator by calling Evaluate on the vsm_model, word2vec_model_0,
  and word2vec_model_1
- Then call evaluator.precision_recal(show_parameters=False,
  show_random_model=True) to graph the models and performances