Doc2Vec + Tensorflow

CSCI 6907

Sam Odle

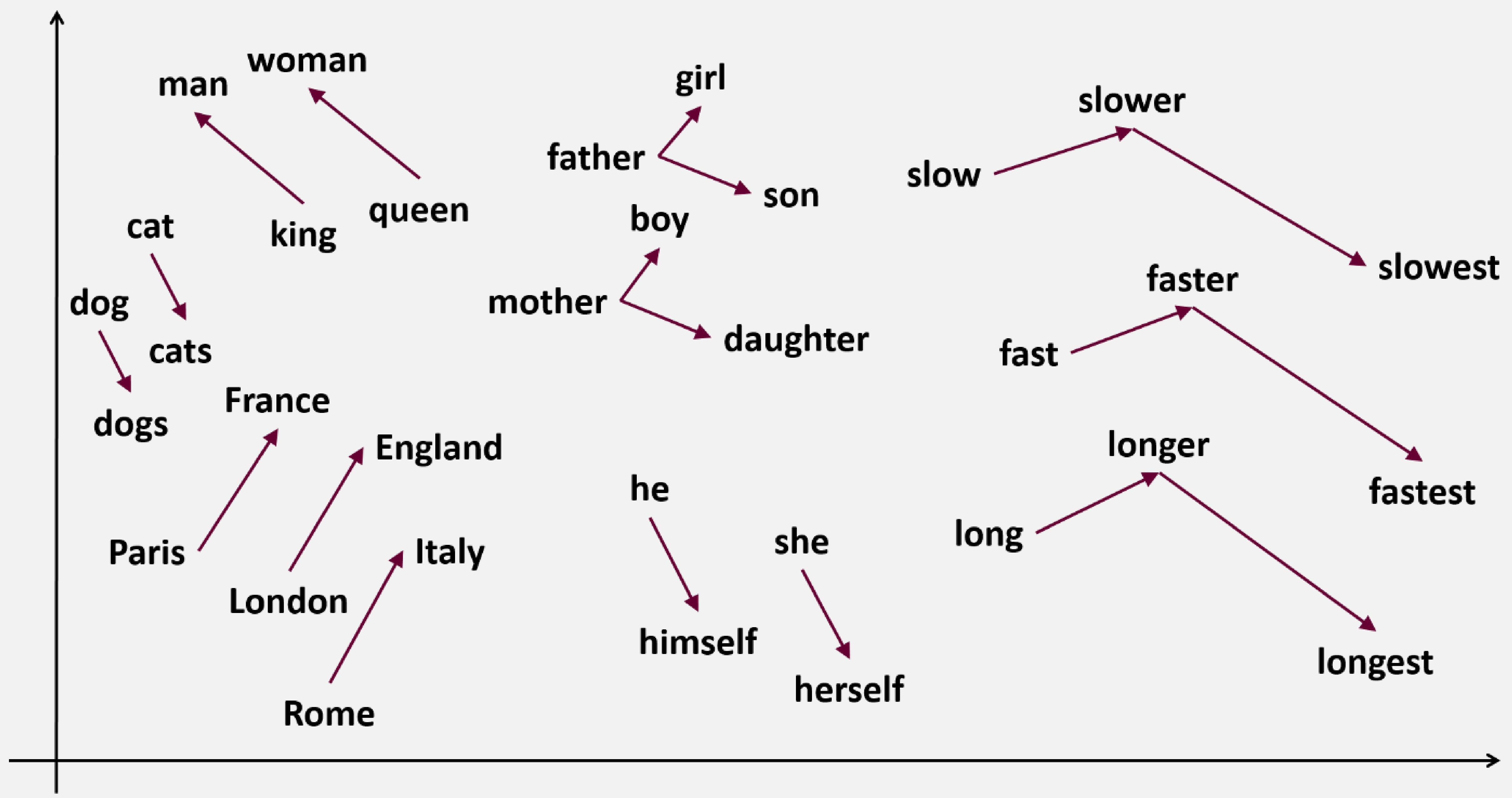
Outline

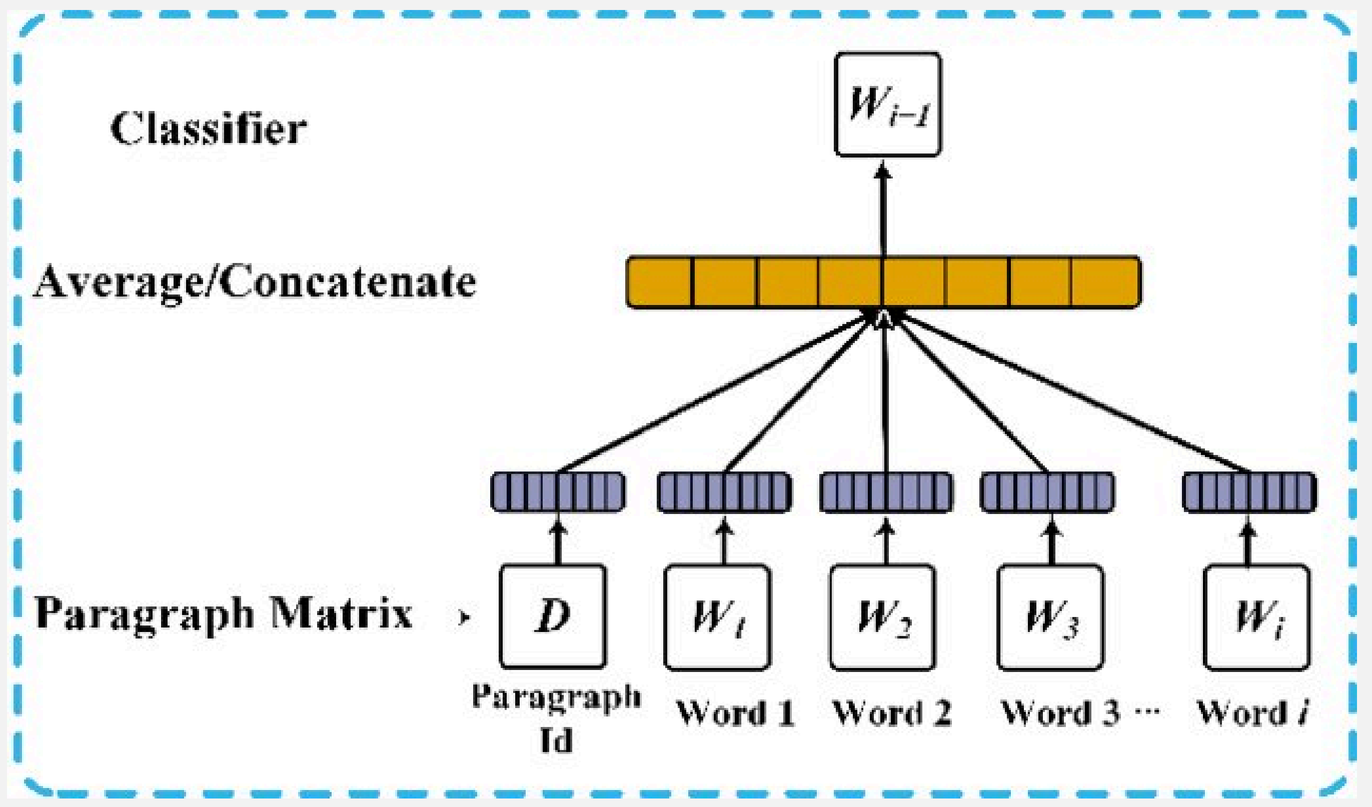
WHY DOC2VEC?

Doc2Vec represents documents as a vector. A document can be a paragraph, phrase, or a full paper.

These vector representations are useful for classification and sentiment analysis.

Doc2Vec is generally considered to be an improvement on LDA and LSA and is a generalization of Word2Vec.





Sources:
https://medium.com/analytics-vidhya/implementing-word2vec-in-tensorflow-44f93cf2665f
https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05
https://radimrehurek.com/gensim/models/word2vec.html

From Words To Documents

Word2Vec

Embeddings To The Rescue

Word2vec generates embeddings (numerical representations) for words that lets us understand contextual relationships.

Semantic & Syntactic Relationships

These word embeddings can capture multiple degrees of similarity between words, including tense differences.

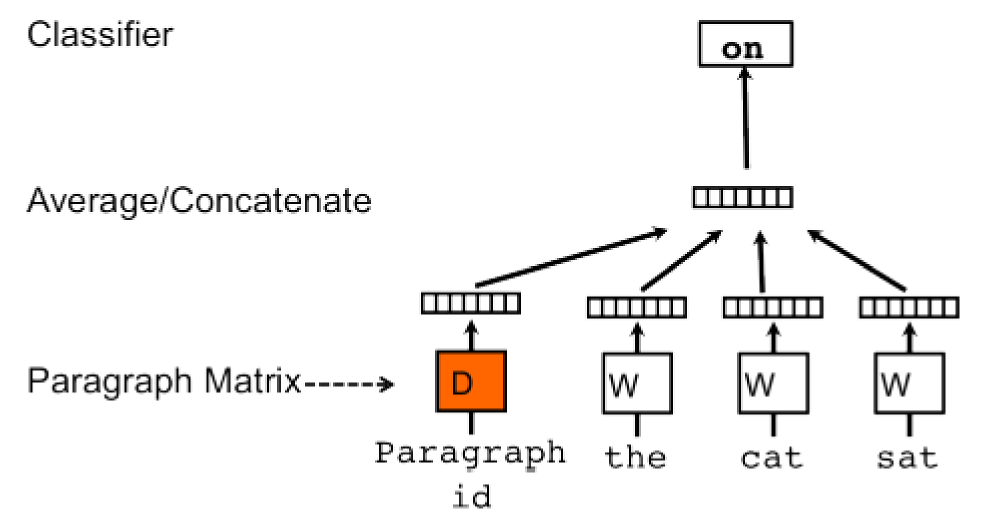
Unknown Words Are Tricky

If the model encounters an unknown word, it will use a random vector which is (unsurprisingly) not how you would ideally like the word to be represented.

Doc2Vec

Numeric Document Representation

Doc2vec adds a document unique feature vector to the word2vec model that will be trained to represent the entire document.



Faster & Less Memory?

If you do not need to save word vectors, doc2vec can provide substantial memory and execution speed savings.

2 Flavors

Like word2vec, there are two primary doc2vec models (see next slide for details).

DM vs DBOW vs Combined

Distributed Bag of Words (PV-DBOW)

- Generally faster to train.
- Single vector for the document which is used to predict each word.
- The document of paragraph vector will be used to classify all the words belonging to the document.
- Analogous to the word2vec skip-gram model, where a central word is used to predict surrounding words.

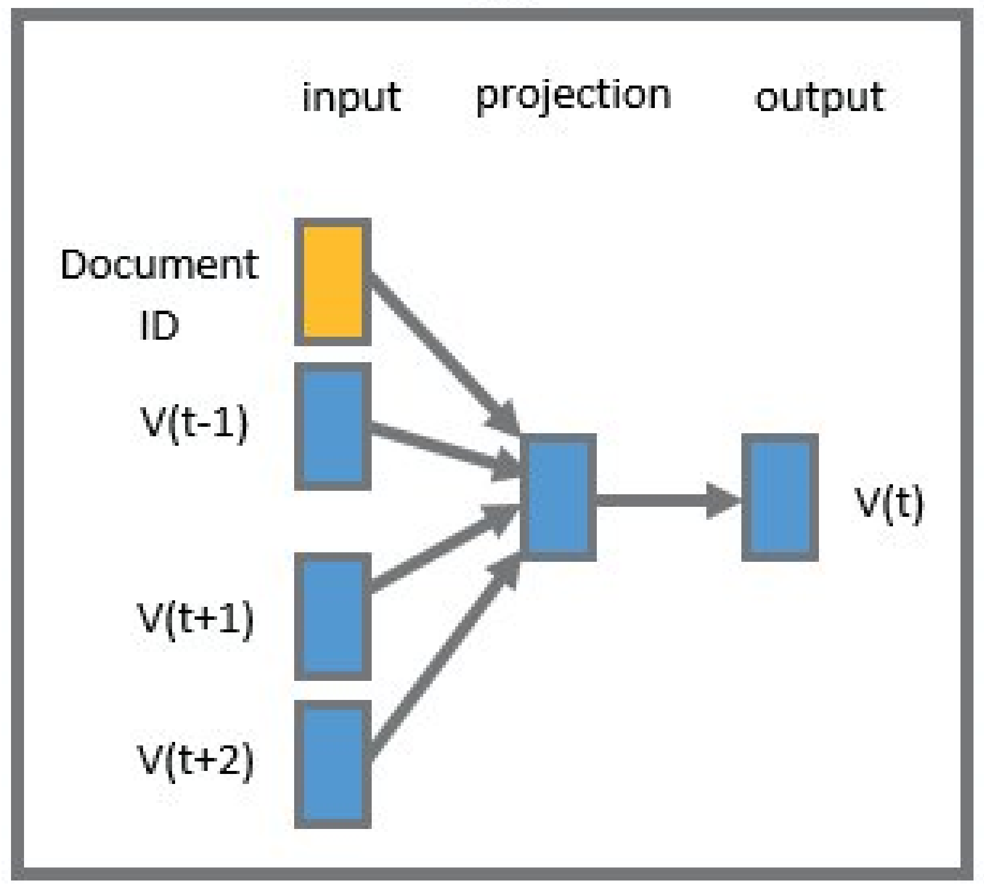
Distributed Memory (PV-DM)

- Randomly sample consecutive words from a document/paragraph then predict a center word from the set by taking the context words and document ID as inputs.
- Analogous to the word2vec CBOW model.
- In some cases can capture context more effectively than DBOW.

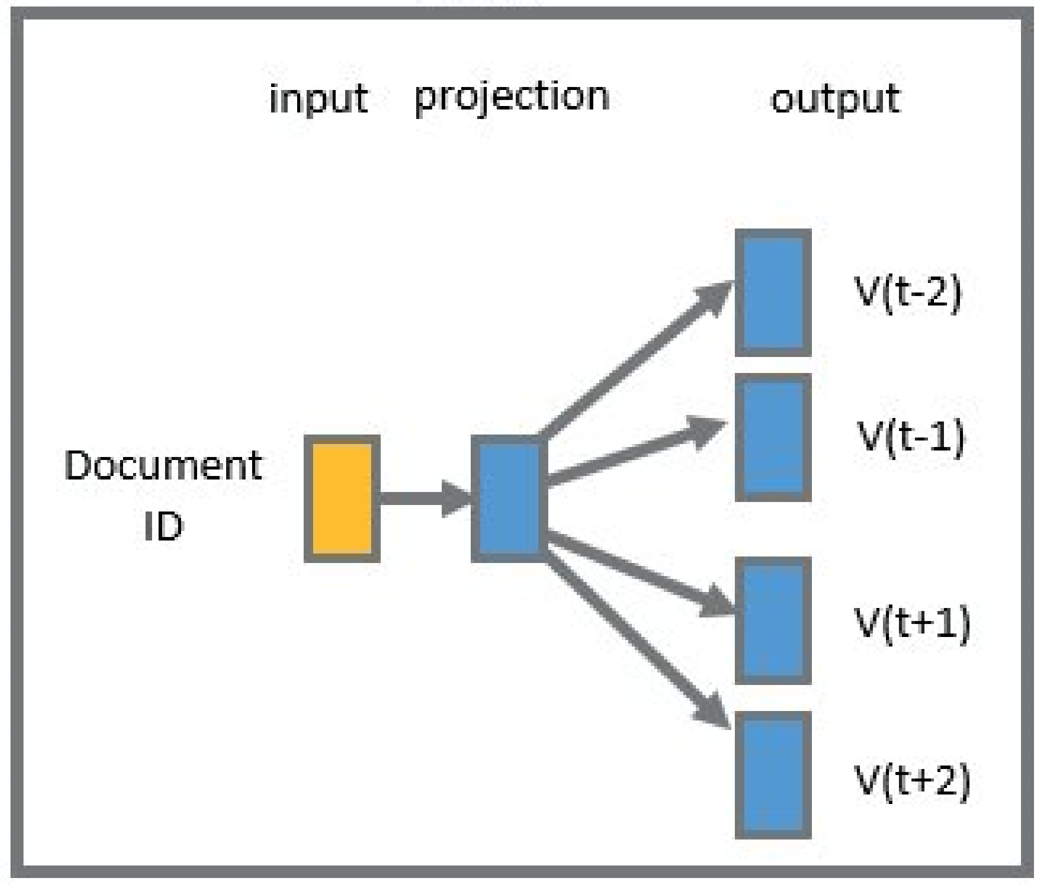
Combined: DBOW + DM

- Recommended by Mikolov et al in initial paper *and* by the Gensim docs as a way to improve performance in larger datasets.
- "Vectors can have much of their training when the model is settling, so it is sometimes the case that new vectors re-inferred from the final state of the model serve better as the input/test data for downstream tasks."

DM



DBoW



Sources:
https://www.researchgate.net/publication/320829283_Sentiment_analysis_on_Twitter_data_with_semi-supervised_Doc2Vec/figures?lo=1
https://medium.com/@amarbudhiraja/understanding-document-embeddings-of-doc2vec-bfe7237a26da

Doc2Vec: A Brief History

Early 2000s

LDA and LSA popular for estimating continuous representations of words.

2013

Word2Vec patented by Google team led by Tomas Mikolov

2014

Doc2Vec published by Mikolov et al

2016

Gensim releases python Word2Vec, porting from C package

c2016 - Present

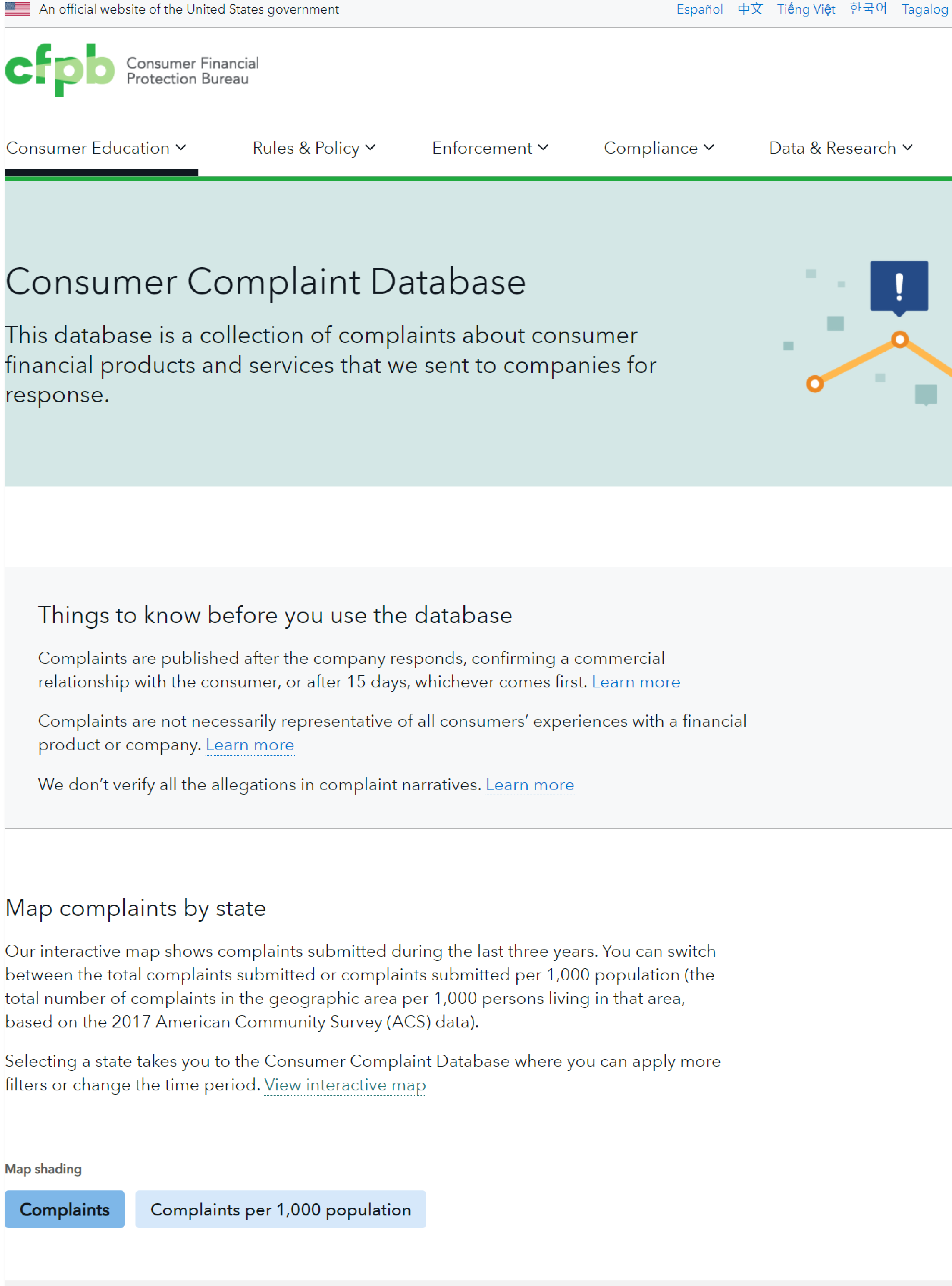
Research into hybrid LDA/LSA/*2vec approaches

Sources:
https://par.nsf.gov/servlets/purl/10055536
https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05
https://radimrehurek.com/gensim/models/word2vec.html

Project Overview

DATASET:
CFPB COMPLAINTS

- **Publicly available collection of complaints** about consumer financial products provided by Consumer Financial Protection Bureau
- Records only added to the set after the company responds to the complaint **to ensure the company has had a relationship with the consumer**.
- **>1 million records** (1.44GB) each related to 1 of 18 products.


An official website of the United States government
Español 中文 Tiếng Việt 한국어 Tagalog
cfpb Consumer Financial Protection Bureau
Consumer Education
Rules & Policy
Enforcement
Compliance
Data & Research

Consumer Complaint Database

This database is a collection of complaints about consumer financial products and services that we sent to companies for response.

Things to know before you use the database

Complaints are published after the company responds, confirming a commercial relationship with the consumer, or after 15 days, whichever comes first. Learn more

Complaints are not necessarily representative of all consumers' experiences with a financial product or company. Learn more

We don't verify all the allegations in complaint narratives. Learn more

Map complaints by state

Our interactive map shows complaints submitted during the last three years. You can switch between the total complaints submitted or complaints submitted per 1,000 population (the total number of complaints in the geographic area per 1,000 persons living in that area, based on the 2017 American Community Survey (ACS) data).

Selecting a state takes you to the Consumer Complaint Database where you can apply more filters or change the time period. View interactive map

Map shading

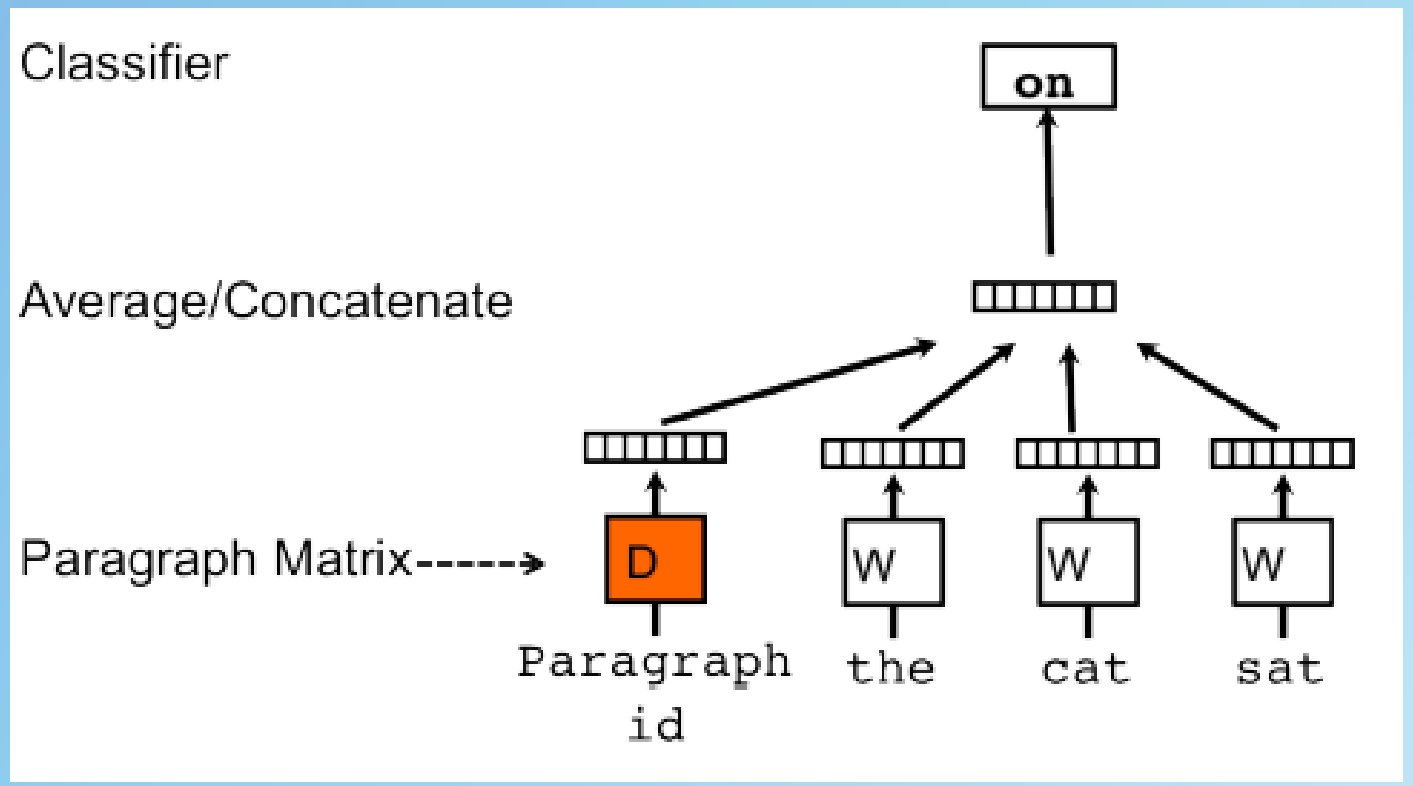
Sources:
https://catalog.data.gov/dataset/consumer-complaint-database

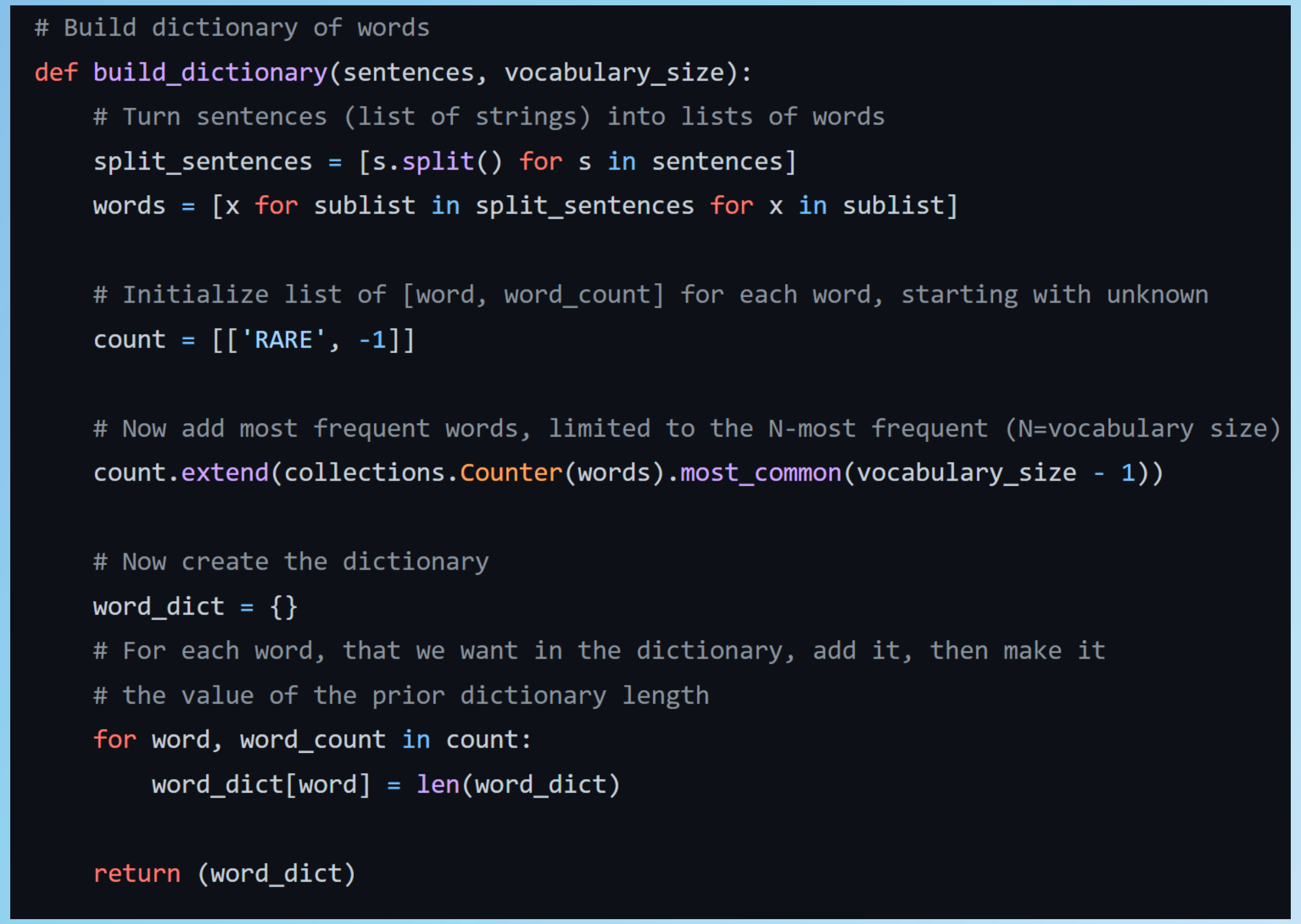
TENSORFLOW PV-DM DOC2VEC IMPLEMENTATION

1. **Clean/Preprocess Data w/ BeautifulSoup & Regex**
   - Remove stopwords, convert to lowercase, remove common special characters from the document.
   - A more robust preprocessing may help with better results, however the same steps were used for both TF and Gensim.

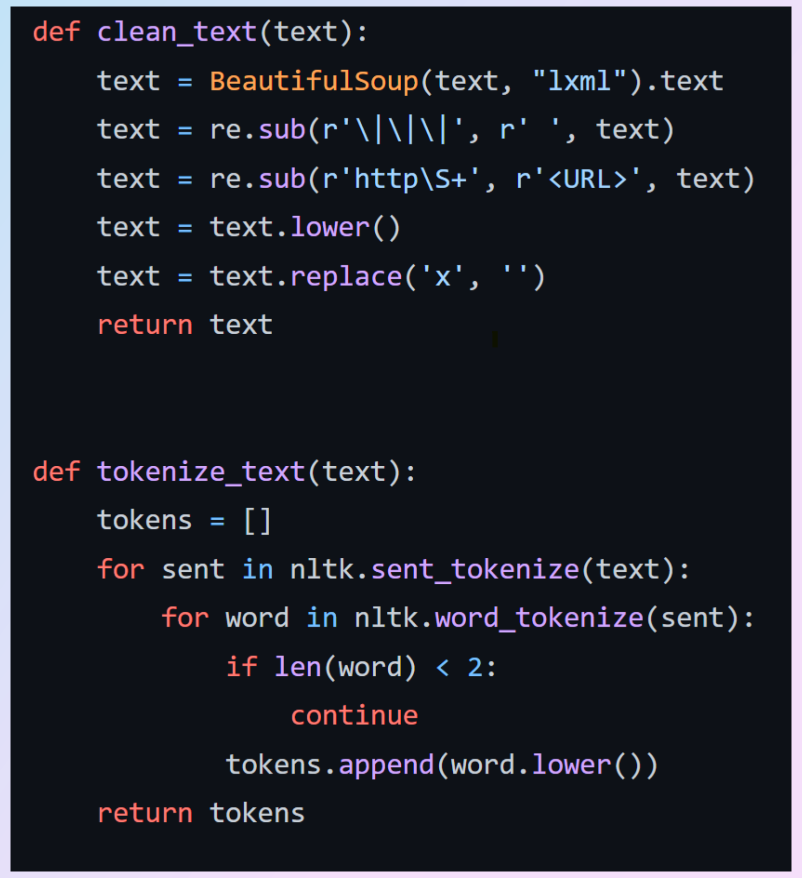
2. **Build a dictionary of the n most common words.**
   - Everything else becomes unknown.

3. **Word vectors are initialized randomly.**
   - Word vectors are shared across paragraphs and contexts. Paragraph vectors can be thought of as word vectors, although they are not shared across paragraphs (but are across contexts, acting as a memory of sorts).





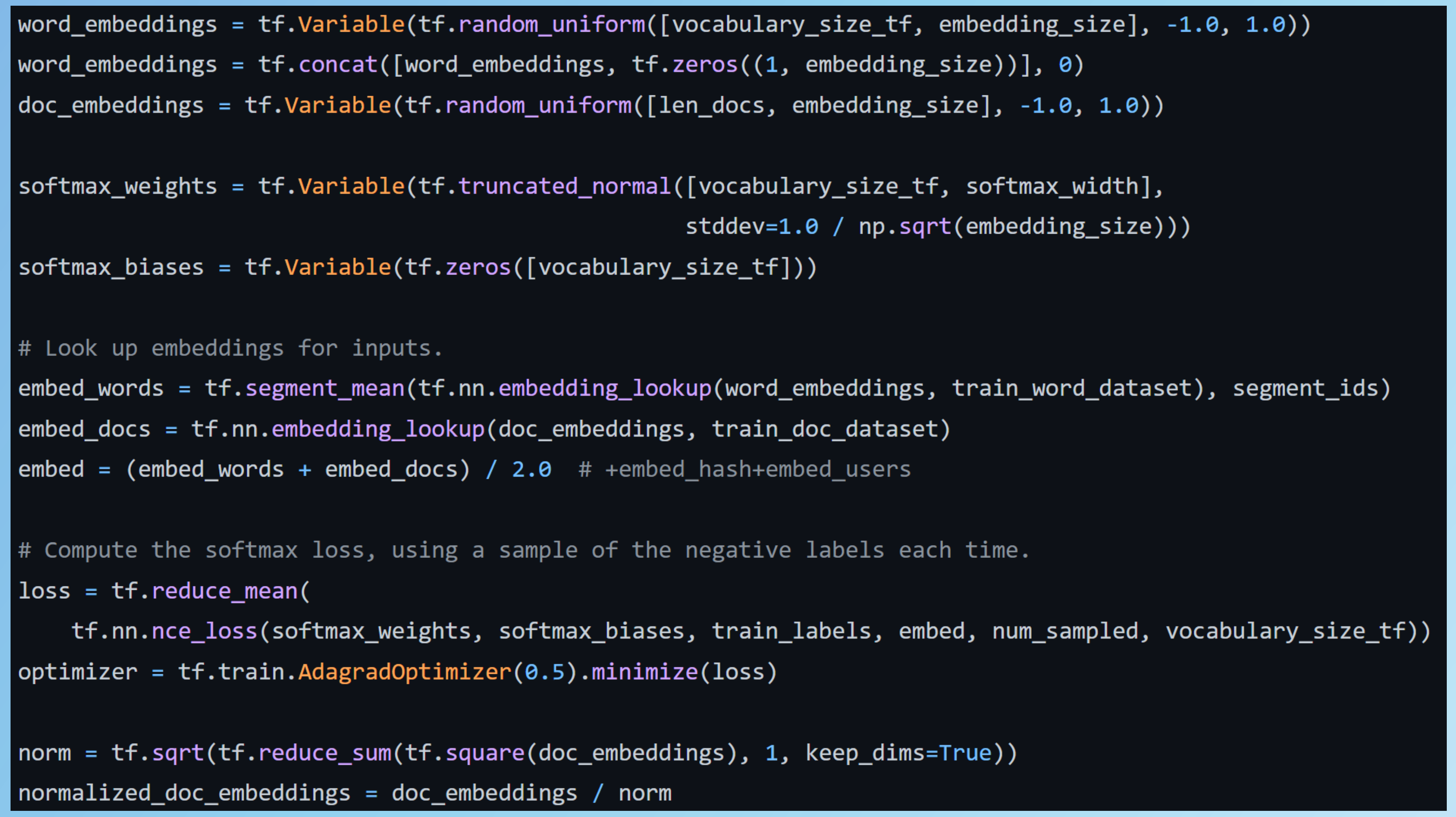
```python
# Build dictionary of words
def build_dictionary(sentences, vocabulary_size):
    # Turn sentences (list of strings) into lists of words
    split_sentences = [s.split() for s in sentences]
    words = [x for sublist in split_sentences for x in sublist]

    # Initialize list of [word, word_count] for each word, starting with unknown
    count = [['RARE', -1]]

    # Now add most frequent words, limited to the N-most frequent (N=vocabulary size)
    count.extend(collections.Counter(words).most_common(vocabulary_size - 1))

    # Now create the dictionary
    word_dict = {}
    # For each word, that we want in the dictionary, add it, then make it
    # the value of the prior dictionary length
    for word, word_count in count:
        word_dict[word] = len(word_dict)

    return (word_dict)
```



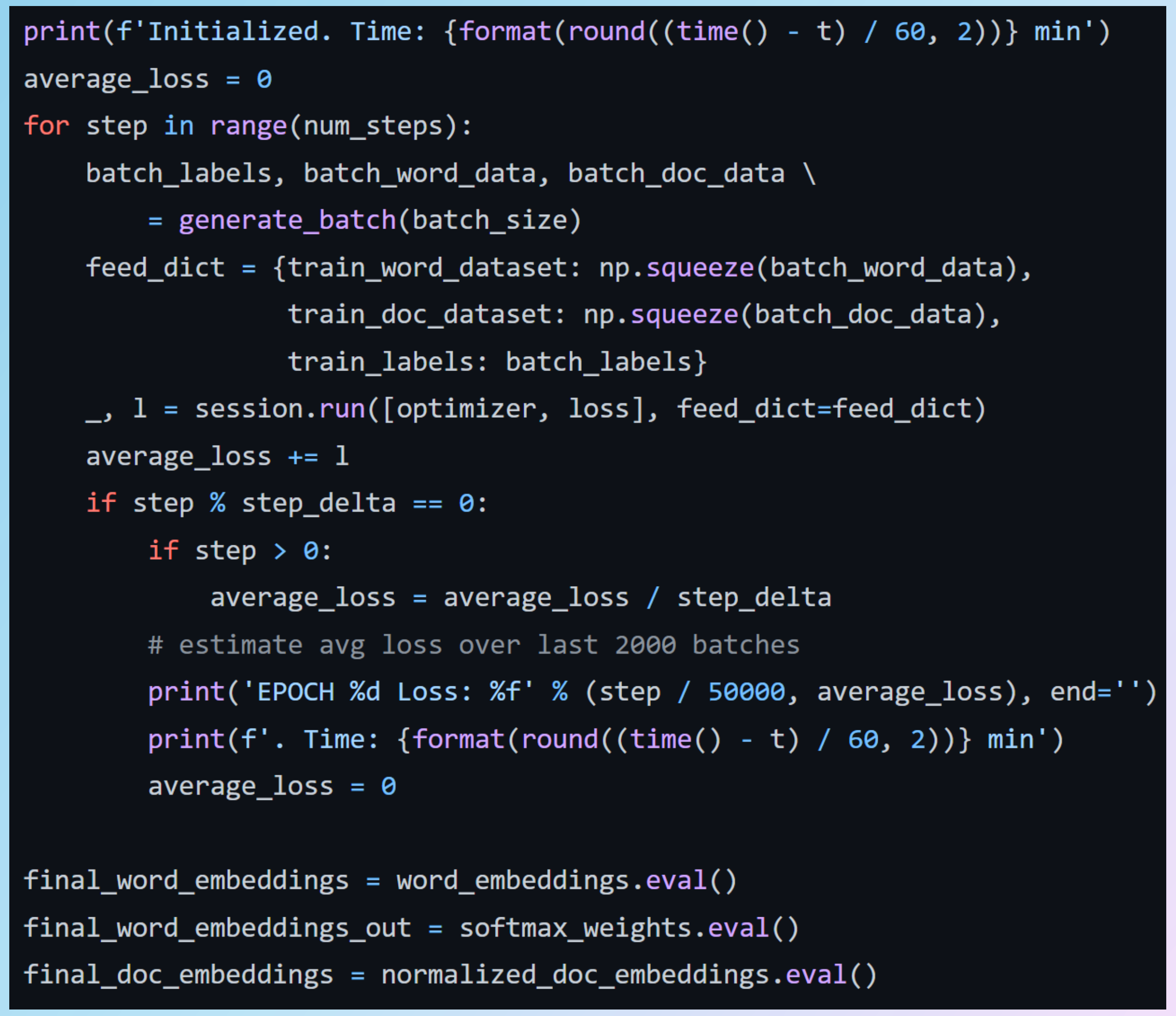
```python
def clean_text(text):
    text = BeautifulSoup(text, "lxml").text
    text = re.sub(r'\|\|\|', r' ', text)
    text = re.sub(r'http\S+', r'<URL>', text)
    text = text.lower()
    text = text.replace('x', '')
    return text


def tokenize_text(text):
    tokens = []
    for sent in nltk.sent_tokenize(text):
        for word in nltk.word_tokenize(sent):
            if len(word) < 2:
                continue
            tokens.append(word.lower())
    return tokens
```

TENSORFLOW PV-DM DOC2VEC IMPLEMENTATION

4. At each step, sample a context from a fixed length sliding window. Lookup word and doc embeddings and average them. Use embeddings to predict next work or missing word.
5. Calculate the loss.
   - Tf.nn.nce_loss.
   - Negative sampling
6. The optimizer updates the weights of the model.
   - tf.train.AdagradOptimizer
   - Under the hood, weights are updated via backpropagation using the gradient to update the parameters in our model.
7. Inference stage holds word embeddings constant and focuses on new paragraph embeddings/updates.



```python
word_embeddings = tf.Variable(tf.random_uniform([vocabulary_size_tf, embedding_size], -1.0, 1.0))
word_embeddings = tf.concat([word_embeddings, tf.zeros((1, embedding_size))], 0)
doc_embeddings = tf.Variable(tf.random_uniform([len_docs, embedding_size], -1.0, 1.0))

softmax_weights = tf.Variable(tf.truncated_normal([vocabulary_size_tf, softmax_width],
                                                  stddev=1.0 / np.sqrt(embedding_size)))
softmax_biases = tf.Variable(tf.zeros([vocabulary_size_tf]))

# Look up embeddings for inputs.
embed_words = tf.segment_mean(tf.nn.embedding_lookup(word_embeddings, train_word_dataset), segment_ids)
embed_docs = tf.nn.embedding_lookup(doc_embeddings, train_doc_dataset)
embed = (embed_words + embed_docs) / 2.0  # +embed_hash+embed_users

# Compute the softmax loss, using a sample of the negative labels each time.
loss = tf.reduce_mean(
    tf.nn.nce_loss(softmax_weights, softmax_biases, train_labels, embed, num_sampled, vocabulary_size_tf))
optimizer = tf.train.AdagradOptimizer(0.5).minimize(loss)

norm = tf.sqrt(tf.reduce_sum(tf.square(doc_embeddings), 1, keep_dims=True))
normalized_doc_embeddings = doc_embeddings / norm
```



```python
print(f'Initialized. Time: {format(round((time() - t) / 60, 2))} min')
average_loss = 0
for step in range(num_steps):
    batch_labels, batch_word_data, batch_doc_data \
        = generate_batch(batch_size)
    feed_dict = {train_word_dataset: np.squeeze(batch_word_data),
                 train_doc_dataset: np.squeeze(batch_doc_data),
                 train_labels: batch_labels}
    _, l = session.run([optimizer, loss], feed_dict=feed_dict)
    average_loss += l
    if step % step_delta == 0:
        if step > 0:
            average_loss = average_loss / step_delta
        # estimate avg loss over last 2000 batches
        print('EPOCH %d Loss: %f' % (step / 50000, average_loss), end='')
        print(f'. Time: {format(round((time() - t) / 60, 2))} min')
        average_loss = 0

final_word_embeddings = word_embeddings.eval()
final_word_embeddings_out = softmax_weights.eval()
final_doc_embeddings = normalized_doc_embeddings.eval()
```

Sources:
https://github.com/sachinruk/doc2vec_tf

CLASSIFICATION

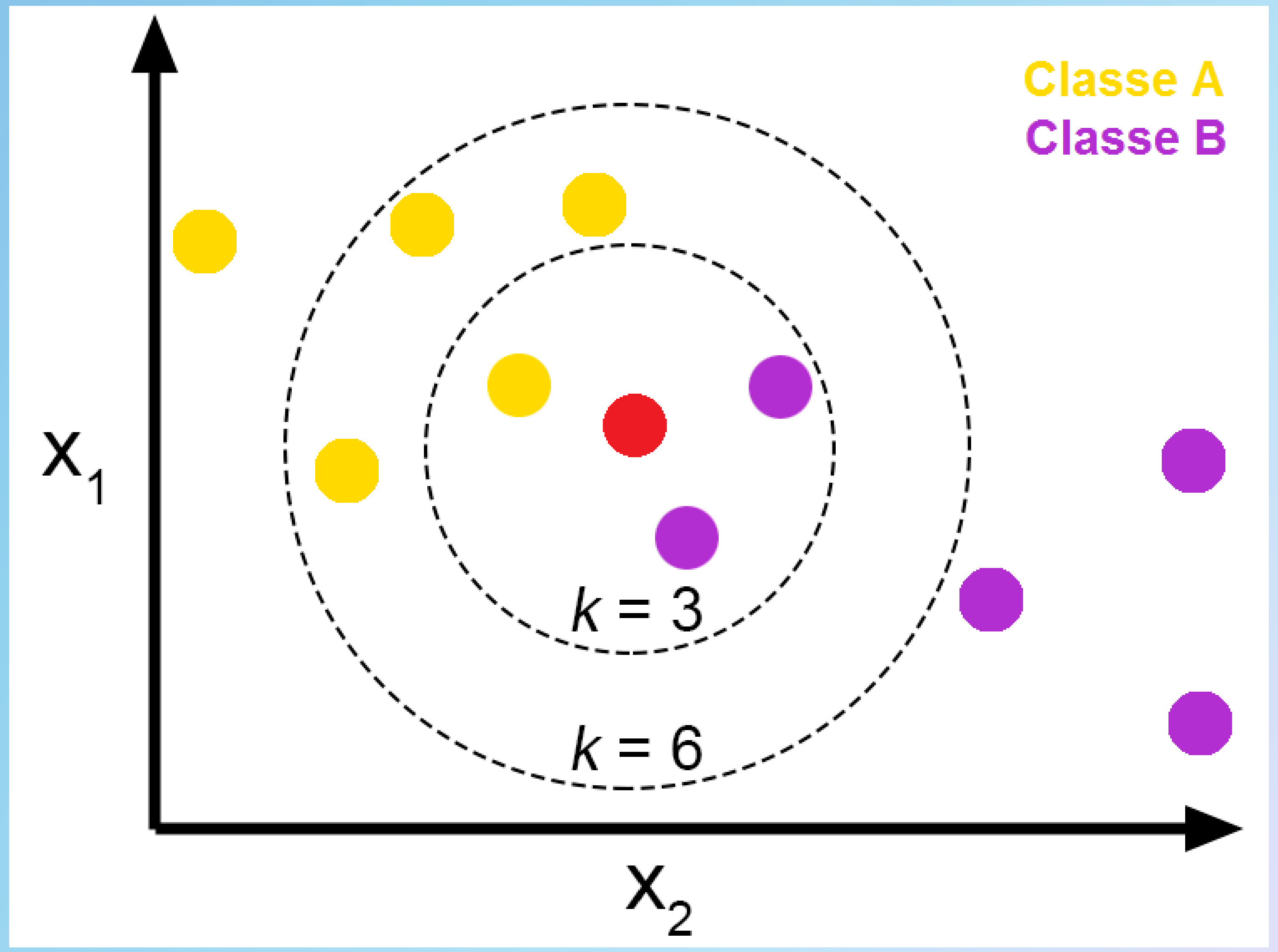
Hypothesis 1: Most of the time, the 'closest' document will have the same label as a given test document.

Result 1: Most of the time, this is not the case. This may be due to the similarities in certain financial categories (e.g., car & house loan vs credit card).

Hypothesis 2: For a given test document, a batch of its nearest neighbors will predominantly be from the correct group.

Result 2: This appears to be true, even with minimal training.

Hypothesis 3: The average distance between a test document and every document from a given label group will be the smallest for the correct label.



Sources:
https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d
https://github.com/sachinruk/doc2vec_tf

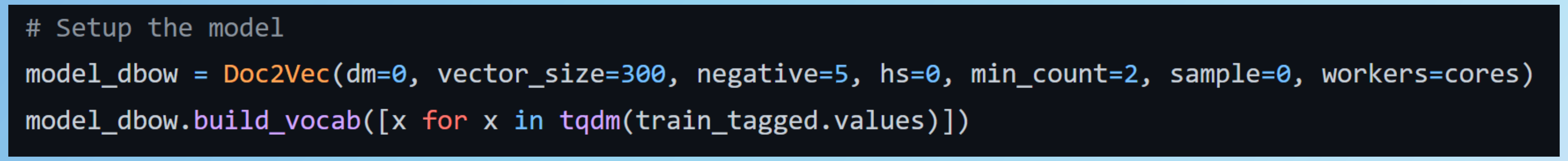
GENSIM IMPLEMENTATION

The most popular python doc2vec library is from gensim.

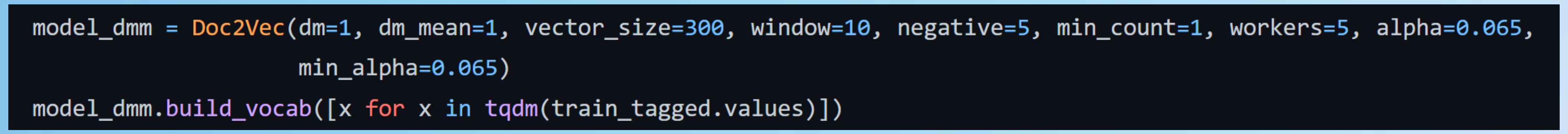
As you can see from the screenshots to the right, setting up models can be done in one or two lines of code – everything else is handled behind the scenes.

Switching between DM and DBOW is as simple as toggling a parameter.

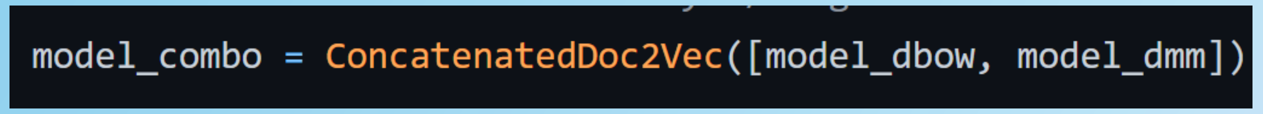
Gensim is heavily optimized and will generally be faster than a TensorFlow implementation, at the cost of being much less customizable.



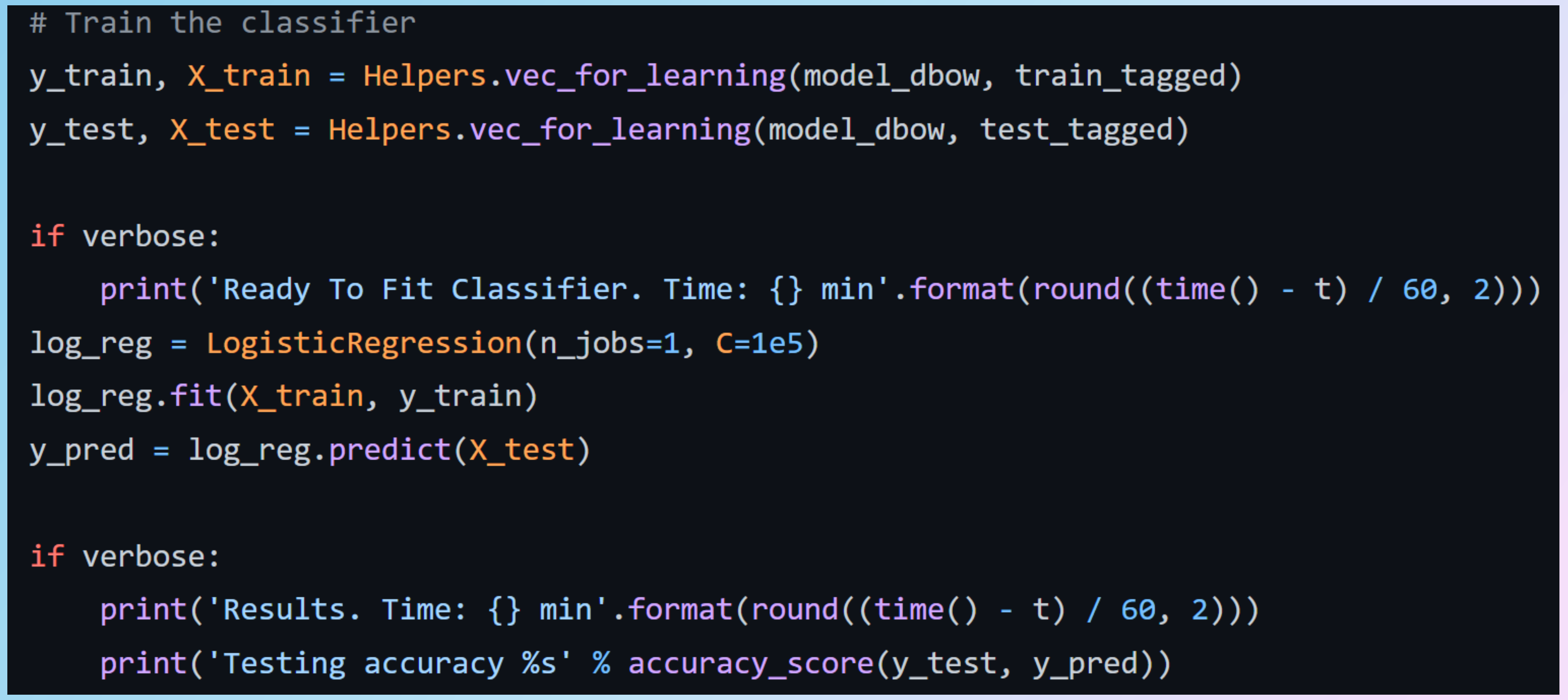
```
# Setup the model
model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, hs=0, min_count=2, sample=0, workers=cores)
model_dbow.build_vocab([x for x in tqdm(train_tagged.values)])
```



```
model_dmm = Doc2Vec(dm=1, dm_mean=1, vector_size=300, window=10, negative=5, min_count=1, workers=5, alpha=0.065,
                    min_alpha=0.065)
model_dmm.build_vocab([x for x in tqdm(train_tagged.values)])
```

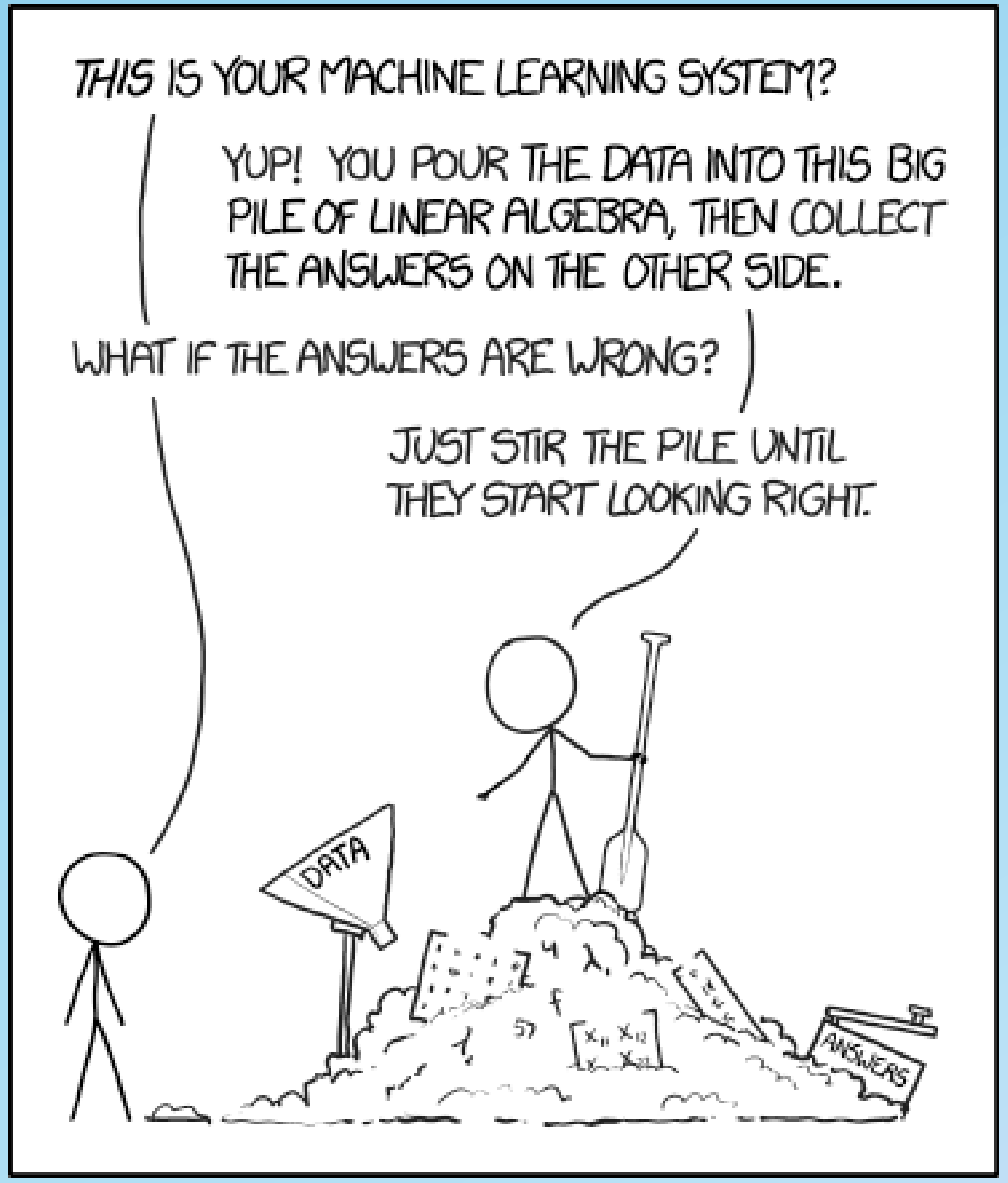


```
model_combo = ConcatenatedDoc2Vec([model_dbow, model_dmm])
```

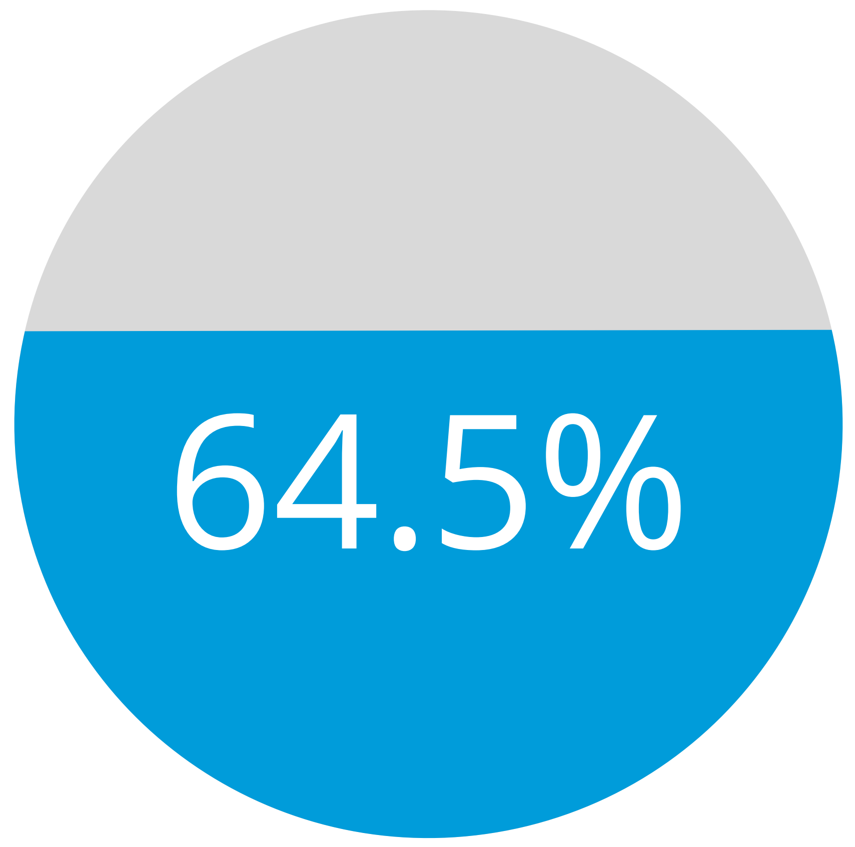


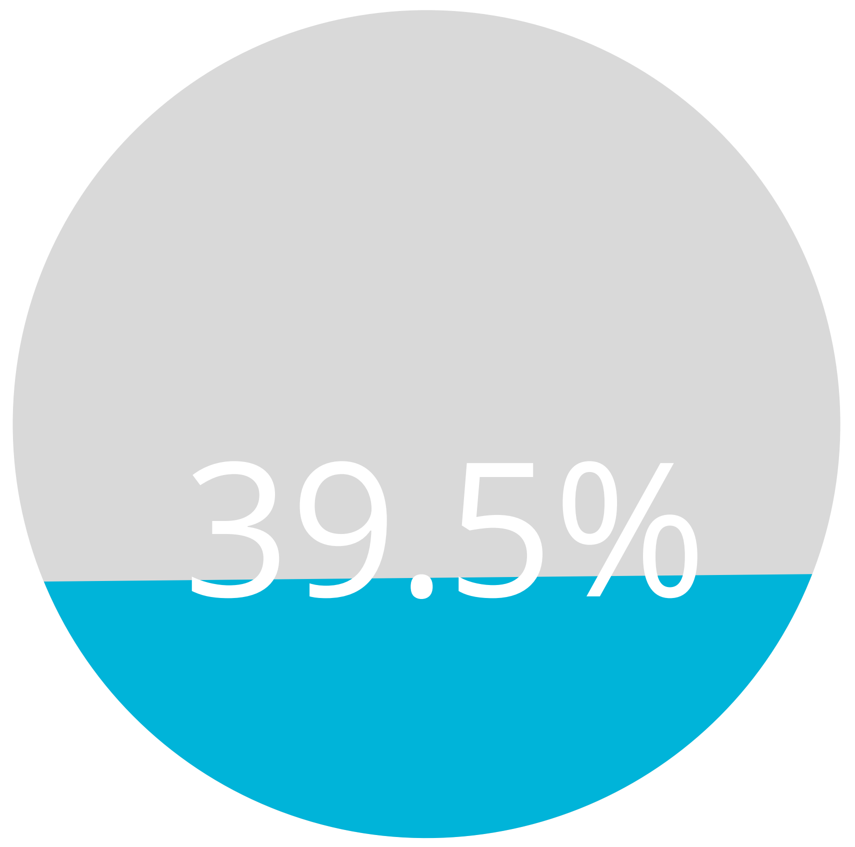
```
# Train the classifier
y_train, X_train = Helpers.vec_for_learning(model_dbow, train_tagged)
y_test, X_test = Helpers.vec_for_learning(model_dbow, test_tagged)

if verbose:
    print('Ready To Fit Classifier. Time: {} min'.format(round((time() - t) / 60, 2)))
log_reg = LogisticRegression(n_jobs=1, C=1e5)
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)

if verbose:
    print('Results. Time: {} min'.format(round((time() - t) / 60, 2)))
    print('Testing accuracy %s' % accuracy_score(y_test, y_pred))
```

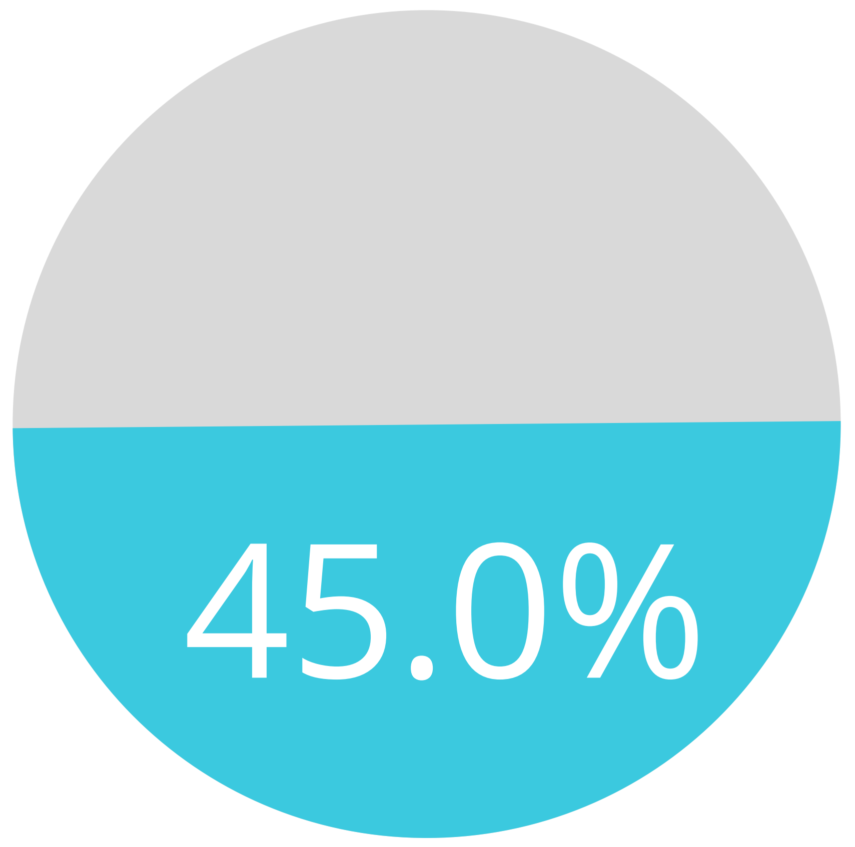
Results

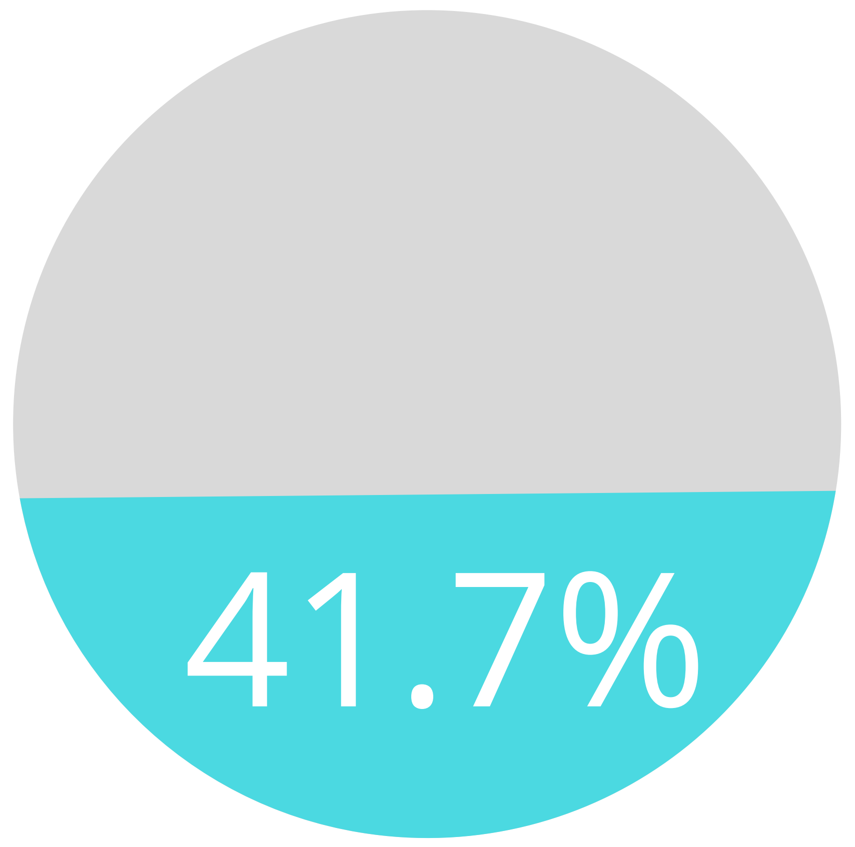

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Model Classification Accuracy









Gensim DM

Gensim DM model using entire dataset split into 70/30 Train/Test.

Gensim DBOW

Gensim DBOW model using entire dataset split into 70/30 Train/Test.

TF DM – Deep Dive

TF Model trained on 200000 documents with 2x vocab (40k) of 'Fast'

TF DM - Fast

TF Model trained on 500 documents w/ abbreviated vobaculary.

Full Results, All Models & All Tests

MODEL	DBOW	DM	COMBO
GENSIM	39.5%	64.5% ✓	64.2%

MODEL	FAST 1N	FAST 5N	ROBUST 1N	ROBUST 5N
TF	24.0%	41.7%	29.3%	45.0% ✓

Conclusions

CONCLUSIONS

Ideas & Opportunities

Recommended actions & next steps (i.e. With more time, I'd would've liked to look into these).

- In TF, develop a more robust classification method.
- Bring in more fields from the source data, possibly document relationships between categories.
- Train TF model on the full dataset.

Open Questions

Areas for further research and exploration based on the results of this analysis.

- Why did the TF model do so well after being trained on so few records?
- The initial Mikolov paper recommends the combo model, however in Gensim it did not perform well.
- Is the dataset biased or unbalanced in a way that would shape these results?

QUESTIONS?

THANK YOU!

References

Research/Supporting Documentation

- Paper, Distributed Representations of Sentences and Documents: https://arxiv.org/pdf/1405.4053.pdf
- Paper, Efficient Estimation of Word Representations in Vector Space: https://arxiv.org/pdf/1301.3781.pdf
- Word2Vec TensorFlow docs: https://www.tensorflow.org/tutorials/text/word2vec
- Doc2Vec TensorFlow Sample Implementation (Basic): https://github.com/wangz10/tensorflow-playground/blob/master/doc2vec.py
- Doc2Vec TensorFlow Sample Implementation (Advanced): https://github.com/PacktPublishing/TensorFlow-Machine- Learning-Cookbook/blob/master/Chapter%2007/doc2vec.py
- Gensim Doc2Vec Examples:
  - https://github.com/RaRe-Technologies/gensim/blob/develop/gensim/models/doc2vec.py
  - https://www.tutorialspoint.com/gensim/gensim_doc2vec_model.htm