CHAPTER .1

Retrofitting

Let's say we have a corpus of texts from various political parties, and we learn word embeddings. In addition to the semantic similarities, these embeddings reflect some of the biases and attitudes reflected in the texts. However, they do not necessarily reflect the party divisions. We can include this information by using document embeddings for the various parties. However, this requires us to have the information at the time of training, and adds a layer of complexity to the training process (i.e., we might need more training data than for pure word embeddings). Instead, we might want to manipulate the existing word embeddings, keeping the original semantic distinctions, but adding further, non-linguistic information we have. In the same vein, if we have learned some document embeddings of firms based on their online self-descriptions, we might want to incorporate external knowledge we have about the markets they are in, or the types of companies they are (privately or publicly owned, etc.)

So far, we have looked at embeddings as a way to represent the *linguistic* similarity between words and documents in terms of spatial distance. However, especially when our documents represent entities in the real world, we might have additional external, non-linguistic knowledge about the entities that we might want to incorporate. In order to do this, we use an algorithm called **retrofitting** (Faruqui et al., 2015; Hovy and Fornaciari, 2018). **retrofitting**

Retrofitting was originally introduced as a way to refine existing word embedding vectors to let them reflect semantic similarities that were not apparent in the training data, but instead came from external lexicons Faruqui et al. (2015). These lexicons included external ontologies, such as WordNet Miller (1995) or the Paraphrase Data Base PPDB Ganitkevitch et al. (2013) to bring synonyms (e.g., "flat" and "apartment") even closer together in embedding space.

Formally, we create a set Ω that contains tuples of words that belong to the same equivalence group, for example $\Omega = \{(apartment, flat)\}$. During retrofitting, we iteratively update the word embeddings of words within the same class (as defined in Ω) to increase the cosine similarity between them. This creates a retrofitted matrix $\hat{D}_{train}$ of the original word embedding matrix D_{train}. The update for a word embedding d_i is

simply a weighted combination of two things: the original word embedding, and the average vector over all the embeddings of its neighbors:

$$\hat{d}_i = \alpha d_i + \beta \frac{\sum_{j:(i,j)\in\Omega} \hat{d}_j}{N}$$

where d_i is the original embeddings vector, $N = |\{\forall j : (i,j) \in \Omega\}|$ is the set of all embeddings in the same equivalence group, and α and β are hyper-parameters that control the trade-off between the original embeddings space and the updates from the neighboring embeddings during retrofitting. In Faruqui et al. (2015), they set $\alpha = \beta$. In contrast, we can also define

$$\beta = 1 - \alpha$$

By varying α from 0 to 1, we can control the strength of the retrofitting process, effectively trading off the influence of the original embeddings versus the vectors of the equivalence classes. So $\alpha = 1$ simply reproduces the original embeddings matrix, i.e., $\hat{D} = D$, whereas $\alpha = 0$ only relies on the retrofitting updates from the neighborhood after the initialization.

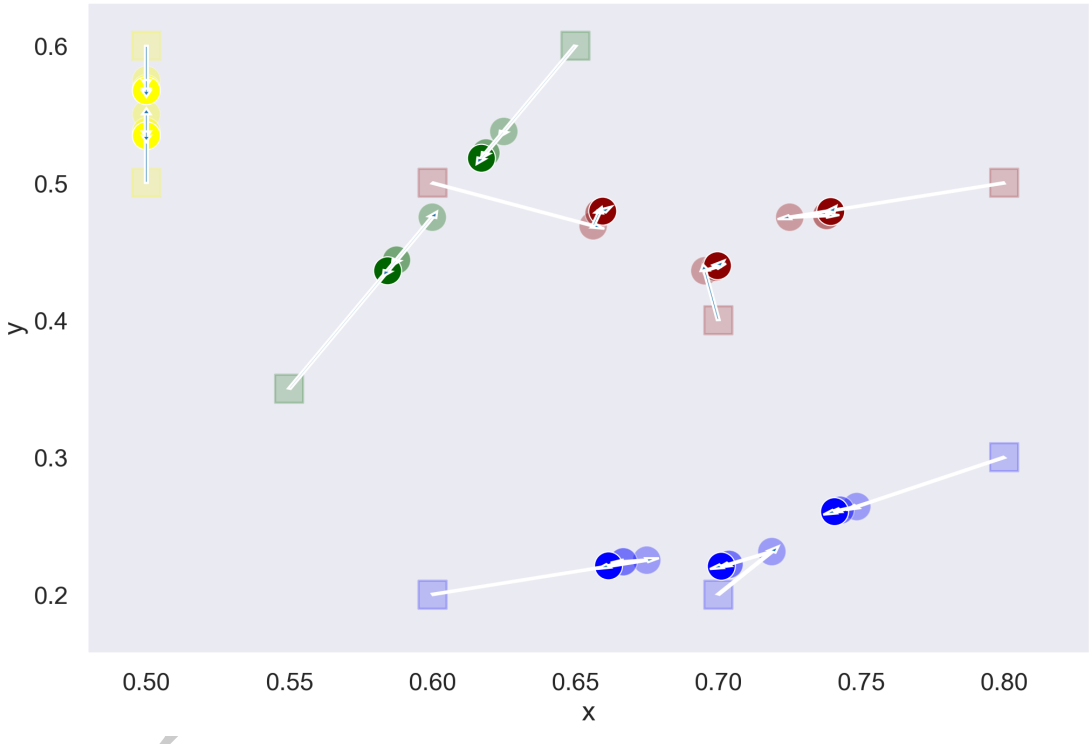

FIGURE 1. Example of retrofitting for 10 words in embeddings space, equivalence class represented by color.

With each iteration of the retrofitting, we move the target word according to the current position of its neighbors. Since those neighbors get moved as well, the average in each round will be slightly different, but converge over time. Because we always use

the original vector as part of the update, the retrofitting does not end up collapsoing all embeddings into the same point. Figure 1 shows the effect of retrofitting on 10 word embeddings, with equivalence classes colored by class. The squares represent the starting point, and the circles show the position at each iteration of the retrofitting. Words belonging to the same equivalence group get drawn closer to each other in embeddings space.

If we are adjusting document embeddings, and our equivalence lexicon Ω is based upon the membership of each document in an outcome class (say, all positive vs. all negative reviews) such that all members of one class are connected, we essentially make the two classes more distinct, which makes them easier to distinguish. We will come back to this in the chapter on classification (see chapter **??**).

The code for retrofitting is extremely simple: it only requires a matrix of N embeddings, and an N-by-N adjacency matrix of neighbors (which is programmatically faster and easier to handle than a dictionary):

```
from copy import deepcopy
import numpy as np
import sys

def retrofit(vectors, neighbors, normalize=False, num_iters=10, alpha
    =0.5):
    beta = 1-alpha
    N, D = vectors.shape

    new_vectors = deepcopy(vectors)

    if normalize:
        for c in range(N):
            vectors[c] /= np.sqrt((vectors[c]**2).sum() + 1e-6)

    # run retrofitting
    print("retrofitting", file=sys.stderr)

    for it in range(num_iters):
        print("\t{}:".format(it + 1), end=' ', file=sys.stderr)
        # loop through every instance
        for c in range(N):
            print(".", end='', file=sys.stderr)
            instance_neighbours = neighbors.get(c, [])
            num_neighbours = len(instance_neighbours)
            #no neighbours, pass - use data estimate
            if num_neighbours == 0:
                continue

```

```
29              new_vectors[c] = alpha * vectors[c] + beta * new_vectors[
    instance_neighbours].sum(axis=0)
30
31          print('', file=sys.stderr)
32
33      return new_vectors
```

CODE 1. A simple trigram LM.

Applications of retrofitting can also include geographic information, such as the adjacency of cells, see Hovy et al. (In Preparation); Purschke and Hovy (In Preparation).

Bibliography

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. 2015. Retrofitting word vectors to semantic lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1606–1615.

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. PPDB: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764.

Dirk Hovy and Tommaso Fornaciari. 2018. Improving author attribute prediction by retrofitting linguistic representations with homophily. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 671–677.

Dirk Hovy, Afhsin Rahimi, Tim Baldwin, and Julian Brooke. In Preparation. Visualizing Regional Language Variation Across Europe on Twitter. In Stanley D. Brunn and Roland Kehrein, editors, *Handbook of the Changing World Language Map*. Dordrecht: Springer.

George A Miller. 1995. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

Christoph Purschke and Dirk Hovy. In Preparation. Lörres, Möppes, and the Swiss. (Re)Discovering Regional Patterns in Anonymous Social Media Data. *Journal of Linguistic Geography*.