Introducing Fuzzy Logic, K-Means, Brown Clustering and Their Use Cases in Understanding Source Code

There are primarily two different types of clustering, mainly hierarchical clustering and non-hierarchical clustering. Word embeddings utilize the distributional hypothesis. It's mapping each word into a vector space with n-dimensions. K-means is a type of non-hiearchical clustering method. If we talk about K-means, it is not the only vector-based clustering. For example, EM (expectation maximization) is also a vector-based clustering.

Brown clustering (Brown et al., 1992) relies on the distributional hypothesis and is a form of hierarchical clustering. Owoputi et al. use the brown clustering method to find clusters in conversational language (Koch, 2022). How Brown clustering works is that it takes in a large corpus of words as inputs. The good thing about Brown clustering is that it doesn't require annotated data. The output can be word clusters or hierarchical words clustering. We know that words that are similar appear in similar context.

K-means is one of the vector-based clustering algorithms out there. What the algorithm does is that it assigns one of the k possible clusters for each object. We can specify how many clusters we want in our data via the k-value. If we do K = 5, that means we would have 5 clusters. How the k-means clustering works is that we first select 5 initial data points, then we measure the distance of the first point with the 5 initial clusters, and assign the point to the nearest clusters. We then proceed with calculating the mean of each cluster. The re-clustering will then happen based on the new means. We evaluate the clusters by calculating the variation in each cluster. The repetition goes on until the clusters are stagnant.

Clustering is further divided into hard clustering and soft clustering. In hard clustering, a data point can only belong to one specific cluster. K-means clustering is an example of a hard-clustering algorithm. In soft clustering, the data point can belong to more than one clusters. Fuzzy C-means clustering is an example of soft clustering. For datasets with known overlapping points, we might consider using the soft clustering method like the Fuzzy C-means clustering as it's known to perform much better than K-means clustering. Fuzzy K-means is similar to Fuzzy C-means in that both are soft clustering algorithms.

There is a Python open-resource library that we can leverage on for the fuzzy c-means algorithm (Ghosh & Kumar, 2013). Although there are a few cons related to K-means such that we will need to define the k-value beforehand and there is the problem of local-optima, but in terms of time complexity, K-means is a lot better than C Fuzzy C-Means (Ghosh & Kumar, 2013). FuzzyWuzzy (Gitau, 2020) is another type of open-source tool for text analysis and NLP tasks. It mainly leverages Levenshtein distance to compute the differences between the strings. The biggest cons for Fuzzy Wuzzy, however, is that it's incredibly slow. An alternative to Fuzzy Wuzzy might be using TF-IDF as vectorizer and then compute the cosine similarity. Fuzzy wuzzy is commonly used when we want to find alternative names for a specific entity, let's say 'USA' and "US", "Vietnam" and "Viet", "Switzerland" and "Switz". Out of curiosity, I also use the Fuzzy Wuzzy to see if I could

classify codes that belong to the same library, but somehow named differently, here is the results.
For example, I want to know what are the functions/methods relates to maps in JavaScript, after I extracted all the functions, I run a Fuzzy Wuzzy and the algorithm would cluster these methods together.

```
google.maps.Marker' ,  'map.setCenter' ,  'google.maps.InfoWindow' ,
'google.maps.event.addListenerOnce'
```

This is just one of the many use cases of Fuzzy Logic for text clustering, and depending further on your end goal, you might want to apply LDA to further cluster these topics.

Sometimes, having these clusters of documents are not enough, and we want to know how the chance of a specific topic appears in a particular document, and that's when Latent Dirichlet Allocation (LDA) comes in. Given than programming languages are quite different from the English language (Abramovsky, 2022), an example of this is the code2vec (Alon et al., 2019) code embeddings which is specifically allocated to understanding source code. CodeT5 (Wang et al., 2021, p. 5) and CodeBert (Feng et al., 2020) are examples of attempt made to train a variety of programming languages. GitHub recently developed CodeQL (*CodeQL*, n.d.) which is an attempt to analyze program data flow. Allamanis et al. states that software is also a form of communication, and its characteristics should be exploited to make software engineering easier (Allamanis et al., 2022). These techniques for understanding source code however, uses deep learning approaches rather than the more simplistic LDA method.

There are primarily 2 main methods of code understanding, majorly from the commonly used NLP methods where the codes are treated as bag of words, and the other way is the syntactic way since the grammatical rules of codes and natural languages can be quite different. It's imperative for us to understand the use cases before going to a specific approach and code understanding is still a budding field which can leverage existing traditional techniques such as K-Means and LDA besides the more commonly used deep learning approaches.

References

Abramovsky, O. (2022, March 28). *SourceCodeAI — Deep Learning for Source Code—Why and How*. Medium. https://towardsdatascience.com/sourcecodeai-deep-learning-for-source-code-why-and-how-50eba7ff0329

Allamanis, M., Brockschmidt, M., & Khademi, M. (2022, March 4). *Learning to Represent Programs with Graphs*. International Conference on Learning Representations. https://openreview.net/forum?id=BJOFETxR-

Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, *3*(POPL), 40:1-40:29. https://doi.org/10.1145/3290353

Brown, P. F., Della Pietra, V. J., deSouza, P. V., Lai, J. C., & Mercer, R. L. (1992). Class-Based *n*-gram Models of Natural Language. *Computational Linguistics*, *18*(4), 467–480.

CodeQL. (n.d.). Retrieved November 6, 2022, from https://codeql.github.com/

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). *CodeBERT: A Pre-Trained Model for Programming and Natural Languages* (arXiv:2002.08155). arXiv. https://doi.org/10.48550/arXiv.2002.08155

Ghosh, S., & Kumar, S. (2013). Comparative Analysis of K-Means and Fuzzy C-Means Algorithms. *International Journal of Advanced Computer Science and Applications*, *4*. https://doi.org/10.14569/IJACSA.2013.040406

Gitau, C. (2020, January 11). *Fuzzy String Matching in Python*. Medium. https://towardsdatascience.com/fuzzy-string-matching-in-python-68f240d910fe

Koch, K. (2022, October 27). *A Friendly Introduction to Text Clustering*. Medium. https://towardsdatascience.com/a-friendly-introduction-to-text-clustering-fa996bcefd04

Wang, Y., Wang, W., Joty, S., & Hoi, S. C. H. (2021). *CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation* (arXiv:2109.00859). arXiv. https://doi.org/10.48550/arXiv.2109.00859