Analyzing Linguistic Shifts in SpongeBob Squarepants using UMAP

Project GitHub Repository: `https://github.com/yuonny/MAT_128B_Proj`

David Kim
University of California, Davis
Davis, California
dvkim@ucdavis.edu

Dunh Adam Lee
University of California, Davis
Davis, California
dualee@ucdavis.edu

Anthony Chang
University of California, Davis
Davis, California
ankchang@ucdavis.edu

***Abstract*—Language usage in kids television shows has changed tremendously in the past 25 years. This project aims to measure this linguistic shift by analyzing data approximately ten years apart. Naturally, we decided on a show everyone has encountered once in their lives: *SpongeBob Squarepants*. We will use the *SpongeBob Squarepants* transcripts from seasons 1-4 and seasons 8-13 as a case study. Our methodology involves processing the dialogue and transforming them into vector embeddings through Word2Vec, then we apply Uniform Manifold Approximation and Projection (UMAP) to reduce the dimensions of the Word2Vec vectors. We then enhance our UMAP visualizations by implementing a Gaussian Kernel Density Estimation (KDE) which helps us identify clusters of dialogue that represent common linguistic patterns. Lastly, we utilize the sliced Wasserstein distance to quantitatively measure the degree of change between the linguistic patterns of the early seasons and the later seasons. Through the results of our visualizations, we aim to capture how linguistic patterns have shifted within the show over time.**

***Keywords*—Semantics, UMAP, SpongeBob Squarepants, Sliced-Wasserstein, Word2Vec**

I. Introduction

If you have been online, a common opinion you may have seen is that "[Show Name] has declined in quality", usually referring to how the show's scripts or themes have changed over time. More often than not, such views are shaped by nostalgic childhood memories. But that got us thinking: Is there an actual, measurable shift in semantic language usage in our favorite childhood TV shows?

Understanding the relationship between words and phrases within a television show's script can provide insight into society's current values, cultural norms, and generational shifts. Therefore, by measuring changes in language, we can better understand how these shows reflect an evolving culture.

To explore this, we chose *SpongeBob Squarepants*, a show that most, if not all, of us grew up watching. It has been on the air for 25 years, and over this quarter-century run, it has produced more than 15 seasons. Due to its popularity and longevity, *SpongeBob Squarepants* serves as an ideal case study to investigate potential semantic shifts in children's television. To address this problem, we developed a method that allows us to visualize, interpret, and quantify differences in the complex linguistic structures found in the scripts of this long-running animated series.

We are comparing the visualizations of the different eras of *SpongeBob Squarepants* to find linguistic patterns and similarities between the older seasons (1-4) and the newer seasons (8-13). To find the underlying structure of these scripts, we compile the

scripts into text files and generate word embeddings using Google's pre-trained Word2Vec model. We then reduce the dimensionality of the produced embeddings utilizing UMAP. After plotting the reduced embeddings, we calculate the Sliced Wasserstein Distance - otherwise known as Earth Mover's Distance-between distributions to calculate the difference between the produced plots. Therefore, using standard numerical analysis concepts such as dimensionality reduction, optimization, and distribution approximation.

By applying these language processing techniques to a show that many of us grew up with, we hope to uncover meaningful trends in how its language has changed over time. This also shows how tools such as word embeddings and distance measures can help us better understand the stories and messages behind the scripts. Ultimately, our hope is to connect childhood memories with measurable language changes, giving us a new way to look at how our favorite children's shows have evolved.

II. Background

The goal of our project is to compare the semantics of the language used in dialogue between the early and later seasons of *SpongeBob Squarepants*. In order to analyze our data, we employed several techniques:

Word2Vec

Our first challenge was finding out how to format the transcript data so that we can effectively analyze it. Through some research, we found that an effective technique to aid in transcript analysis is using Google's natural language processing model, Word2Vec.

Word2Vec allows us to transform raw text data, such as our transcripts, into high-dimensional vectors that are grouped based on their semantic meaning within their sentence context. This model aligns perfectly with the goal of this project and allows us to perform further data processing as needed.

UMAP

After we convert the transcript data into high-dimensional vectors, we needed a way to visualize and compare the data. To visualize our data, we used a dimension reduction algorithm called Uniform Manifold Approximation and Projection (UMAP).

We chose UMAP because it is a technique that reduces and projects the high-dimensional vectors produced by Word2Vec into 2-D vectors, allowing for ease of visualization. In addition to dimension reduction, UMAP also preserves both the local and global structure of our data set. This preservation allows us to understand and interpret the transcript by placing semantically similar scenes closely in 2-D space, and it also helps us compare the shifts in language semantics between earlier and later season transcripts.

Sliced Wasserstein Distance

After using UMAP to obtain 2-D points from the transcript, we are able to visualize the structure for the earlier seasons and the later seasons. However, visualization alone is not enough for us to interpret how much the semantics have changed throughout the years. To answer the question of how much the language has shifted, we needed a way to numerically quantify the difference between both dialogues. That is where the Sliced Wasserstein Distance comes in.

The Sliced Wasserstein distance measures how much effort it takes to transform one probability distribution into another with a very holistic view. It considers the differences in shape, spread, and clustering for different distributions. In our case, each point in the 2-D UMAP represents the meaning of a sentence, and the full set of points represents a probability distribution. The Wasserstein distance computes a value that allows us to to see how far apart both distributions are from each other. This value, along with our plots, helps give us a more complete understanding of the linguistic shift between the seasons.

Gaussian Kernel Density Estimation (KDE)

While UMAP gives us a raw 2-D representation of the sentence embeddings, it is still difficult to interpret the points due to the amount of overlap between them. To help us understand our plots better, we use Gaussian Kernel Density Estimation (KDE) on our UMAP results. The KDE technique works by placing a smooth, symmetric function at each data point and and sums up these points to produce a smooth surface that represents clearly which points are dense.

With KDE, we are able to create a color-coded plot that aids us with identifying how closely related

sentences are for a season. For example, the more densely colored and clustered points are, the more semantically related they are. However, a dispersed set of points are less semantically related.

III. Mathematical Methods

Word2Vec:

Word2vec is a two-layer neural network model that takes an input of a large dataset of texts. It then maps the set of words to a vector space. The model is trained to predict the context of a word and assigns each word in the dataset a vector. The closer the words are in semantic meaning, the closer the vectors are in the space.

We used the `gensim` Python library [1] to train our Word2Vec models. The key parameters used are:

1. `vector_size = 100`: Each word is transformed into a 100-dimensional vector space
2. `window = 5`: Each word has a context window of 5 on both of its sides
3. `min_count = 1`: The minimum count of words to include. Here, we also include sentences made of a single word.
4. `workers = 1`: Number of cores to train Word2Vec
5. `seed = 42`: Seed for reproducibility

Word2Vec Math Overview:
Cosine similarity: Word2Vec represents words as vectors, and then computes the cosine similarity between words. This gives the semantic similarity between them, or the likeness of their meanings.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

UMAP:

UMAP (Uniform Manifold Approximation and Projection) [2] is built on three key assumptions:

1. The data is uniformly distributed on a Riemannian manifold.
2. The manifold is locally connected.
3. The manifold can be approximated locally as Euclidean.

To construct its embedding, UMAP first builds a graph that captures local relationships in the high-dimensional space. This is done by assigning a fuzzy connection strength between each point x_i and its neighbors:

$$\mu_{ij} = \exp\left(-\frac{d(x_i, x_j) - \rho_i}{\sigma_i}\right)$$

Here:

- $d(x_i, x_j)$ is the distance between points x_i and x_j,
- ρ_i is a distance offset that ensures local connectivity,
- σ_i is a normalization factor that controls the smoothness of the connection.

UMAP builds this fuzzy relationship graph by:

1. Finding the k-nearest neighbors for each point x_i.
2. Computing the distances $d(x_i, x_j)$ to those neighbors.
3. Using the formula above to determine the connection strength μ_{ij}.

To make the graph symmetric, UMAP combines the pairwise connection strengths:

$$\tilde{\mu}_{ij} = \mu_{ij} + \mu_{ji} - \mu_{ij} \cdot \mu_{ji}$$

This results in a fuzzy set which is a probabilistic representation of the manifold structure.

UMAP optimizes a low-dimensional layout (typically 2D or 3D) that best preserves these fuzzy relationships, allowing for meaningful visual interpretation of the original high-dimensional data.

Sliced Wasserstein Distance (discrete)

Given two 1D probability distributions μ and ν, the Wasserstein-1 (a.k.a. Earth Mover's) distance is:

$$W_1(\mu, \nu) = \sum_{i=1}^{n} \left| x_{(i)} - y_{(i)} \right|$$

Where:

$x_{(i)}$ and $y_{(i)}$ are sorted samples.

The Wasserstein distance computes the distance between two probability density functions, or how much you have to "move" one to get to the other.

We then **slice** by projecting high-dimensional distributions onto 1D lines, computing 1D Wasserstein distances for each projection, and then averaging the results.

Slicing:

First, pick a random unit vector $\theta \in \mathbb{R}^2$.

This is the direction we are slicing along. In 2D, a random angle $\phi \in [0, 2\pi]$ gives:

$$\theta = (\cos\phi, \sin\phi)$$

This vector represents a line through the origin at angle ϕ. We are projecting our sample points onto this line.

Project all points onto θ

We project each 2D point x_i or y_j onto the line by computing a dot product between the vectors:

$$x_i^\theta = \langle x_i, \theta \rangle = x_i^{(1)} \cos\phi + x_i^{(2)} \sin\phi$$

Project this for all points in both datasets:

$$\{x_1^\theta, \ldots, x_n^\theta\} \subset \mathbb{R}$$

$$\{y_1^\theta, \ldots, y_m^\theta\} \subset \mathbb{R}$$

This turns our 2D scatterplot into two 1D lists of scalar projections along the randomly chosen direction.

Sort 1D projections

$$x^\theta_{\text{sorted}} = \text{sort}(x^\theta), \quad y^\theta_{\text{sorted}} = \text{sort}(y^\theta)$$

If both sets have n points (or if you resample them to equal sizes), we compute:

$$W_1^\theta(X, Y) = \frac{1}{n} \sum_{i=1}^{n} \left| x^\theta_{(i)} - y^\theta_{(i)} \right|$$

This gives us the Wasserstein distance along slice θ.

Average over many slices

Repeat previous steps for many directions $\theta_1, \theta_2, \ldots, \theta_L$, and average the distances:

$$\text{SlicedW}_1(X, Y) = \frac{1}{L} \sum_{\ell=1}^{L} W_1^{\theta_\ell}(X, Y)$$

Averaging over all the slices gives a complete picture of the scale of the semantic difference between the two datasets.

We used the `sliced_wasserstein_distance` function from the POT (Python Optimal Transport) library [3] to compute the Sliced Wasserstein-1 distance.

Gaussian Kernel Density Estimation (KDE)

For a set of n points $\{x_1, x_2, ..., x_n\}$ in $\mathbb{R}^d$ (in our case, $d = 2$ from UMAP), the KDE at a query point $x \in \mathbb{R}^d$ is:

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x_i)$$

Gaussian Kernel: The kernel function K_h is based on a multivariate normal distribution centered at each data point:

$$K_h(x - x_i) = \frac{1}{(2\pi h^2)^{d/2}} \exp\left(-\frac{\|x - x_i\|^2}{2h^2} \right)$$

Where:

- $K_h(\cdot)$ is the Gaussian kernel with bandwidth parameter h
- x_i are the UMAP data points

Gaussian KDE approximates the probability density function of the UMAP embeddings. This colors the scatterplot, with higher-density regions shown in

warmer colors. It gives a visualization of where the semantic content is most concentrated.

We used the `scipy.stats.gaussian_kde` function from the `scipy` Python library [4] to apply Gaussian kernel density estimates for our UMAP-reduced embeddings.

Challenges

One challenge we had in order to properly use Word2Vec was figuring out how to format our data to be used for training. The raw text included metadata like speaker names and stage directions, which needed to be removed to produce dialogue-only vector embeddings. We solved this problem by writing a custom string parsing script `tokenize_script.py` that used regular expressions to strip out non-dialogue words and tokenize each line into clean lowercase words, while also removing punctuation. These tokenized sentences were stored as lists of lists and stored as files using `pickle` for easy reuse of our scripts. This pre-processing allows Word2vec to be trained effectively.

We had difficulties with computational costs. Running UMAP to create Figures 1 and 2 was extremely slow, and it was hard for our computers to keep up. This made it difficult to visualize our data. We then decided to separate Word2vec and UMAP training in two different files to reduce the runtime. Next, we saved the output of the UMAP to reuse later on instead of computing it every time. Another challenge was figuring out how to quantify the difference between the distributions between older and newer seasons. We eventually found the Wasserstein distance, which allowed us to compute this difference and explain the scale of semantic change across the show's run. However, one of our initial challenges was figuring out which distributions to apply Wasserstein to. We originally wanted to apply it to the Gaussian KDE graphs, but we were unable to. Therefore, we decided to calculate the distance between the original distributions using Sliced Wasserstein distance. We used sliced rather than the general form to reduce the computational complexity required by our project.

IV. Numerical Experiments and Results

As our project aims to calculate the semantic shift between *SpongeBob Squarepants* Eras, we first required a visual represention of the respective scripts. To do this, we gathered *SpongeBob Squarepants* texts from a Kaggle Dataset and parsed them into Google's Pretrained Word2Vec model, which produced 100-dimensional embeddings for each word. We then applied UMAP for dimensionality reduction, reducing these vectors to two dimensions. This allowed us to generate scatterplot representations of each era's language usage—shown in Figure 1 and Figure 2.

From the shape of the scatterplots alone, you are able to notice that there is a difference between the early and later seasons. The older seasons seem to clustered groupings of words, suggesting consistent or repetitive language. In contrast, the newer seasons seem more disparate, suggesting wider language usage throughout the seasons. To make this relationship clearer, we utilized Gaussian Kernel Density Estimation (KDE) to approximate the underlying probability distribution functions, allowing us to visualize the density of points and better understand the concentration of the seasons. However, that then begs the question: how can we numerically demonstrate this difference? To answer this, we calculated the Sliced Wasserstein Distance between the two non-KDE distributions and found an initial distance of approximately 2.70, indicating a meaningful difference between the two distributions. This supports our earlier observation that there has been a noticeable shift in the language used in *SpongeBob SquarePants* over time.

Wasserstein Distance measures the minimum "cost" required to transform one distribution into another. In this context, a larger distance suggests a greater difference between the distributions, while a smaller value indicates more similarity.

Since the Sliced Wasserstein Distance does not have a fixed scale, we rely on later calculations and experimental context to understand its implications. Based on our analyses, a value of 2.70 suggests a clear, but not drastic, semantic shift across *SpongeBob*'s eras.

One thing we noticed, however, while running our model was that the calculated Wasserstein Distance largely varied with each UMAP training— at one iteration getting around 3 and another at around 5. This should not be the case as we had kept our data consistent and declared UMAP's `random_state` parameter to be equal to 42 (this allows for reproducibility of the model). Our intuition lead us to

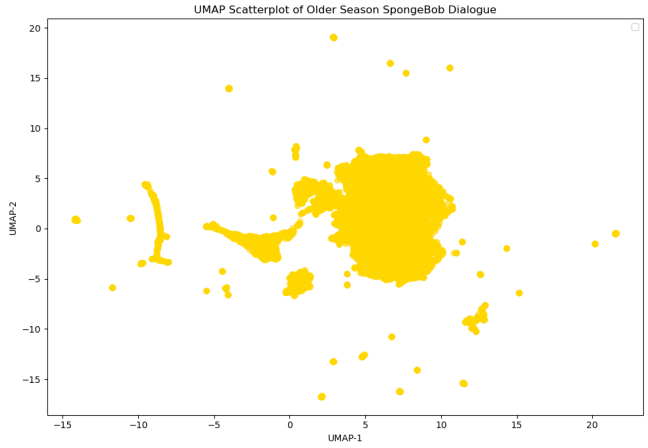

FIGURE I: UMAP OF OLDER SEASONS

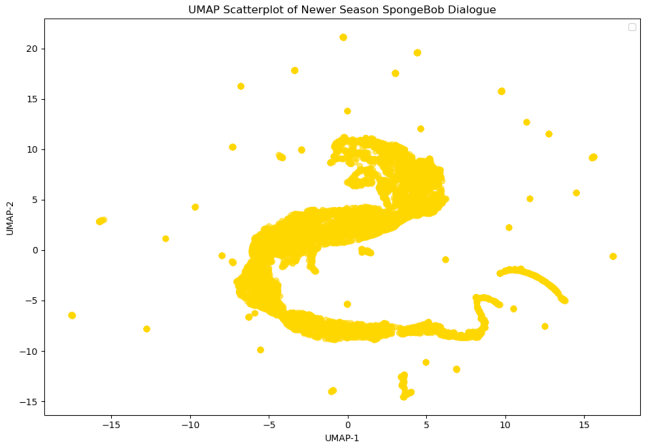

FIGURE II: UMAP OF NEWER SEASONS

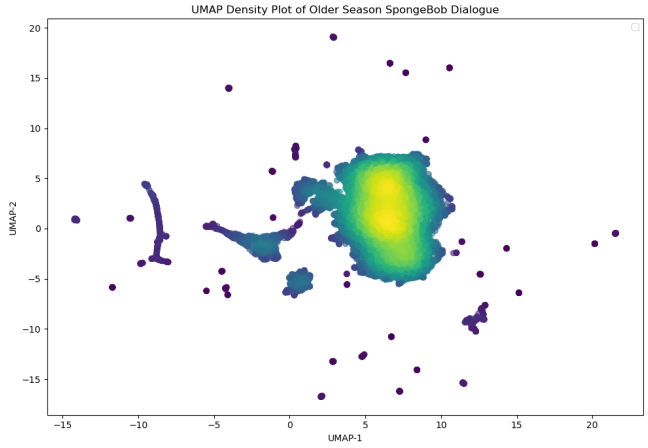

FIGURE III: DENSITY MAP OF OLDER SEASONS

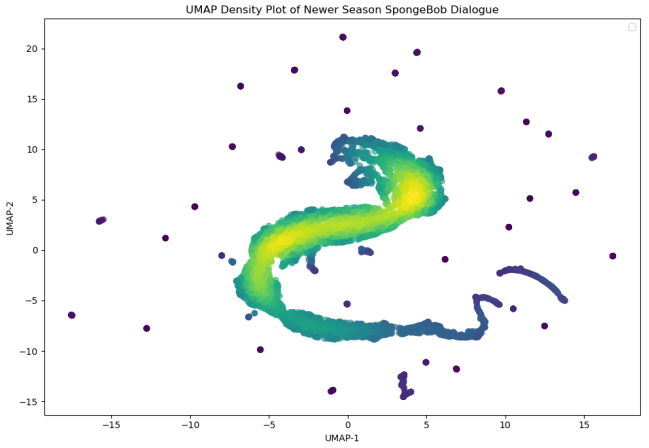

FIGURE IV: DENSITY MAP OF NEWER SEASONS

view Google's Word2Vec model, as it could be outputting different vectors at each iteration of training.

Word2Vec has a parameter `workers` that determines the number of parallel processes used during training. In our case, we had workers equal to 4, resulting in 4 processes running simultaneously to embed the words of our scripts. While this greatly improves training speed and reduces time-complexity, it also means that words were not consistently being converted at the same order— resulting in slightly different vector representations at each run. To attempt to fix this issue, we set workers to be equal to 1— therefore only one process would create embeddings.

However, some variance in the calculated distance still occurred, suggesting that the source of randomness may lie elsewhere, or that the underlying algorithm of Word2Vec inherently introduces variability.

To quantify this variation, we re-ran UMAP thirty times, each time calculating the Sliced Wasserstein Distance, and plotted the resulting values to visualize the fluctuations (see Figure 5). The average distance was calculated to be 2.602, with a standard deviation of approximately 0.0866. This suggests that while some variance remains, our changes contributed to greater stability, and the remaining variation is small enough to be considered negligible.

We then wanted to observe our script's underlying semantic relationship. To do so we removed the random state parameter when calling the UMAP function. This change leads UMAP to become non-deterministic (different outputs with the same initial data). We then re-ran UMAP thirty more times, this time calculating the Wasserstein Distance between an initial UMAP distribution of a script and each newly generated distribution from later runs— then plotting the resulting distances to visualize the degree of variation introduced by UMAP's inherent randomness (see Figure 6). We found that the newer seasons exhibited a wider range of distances—approximately

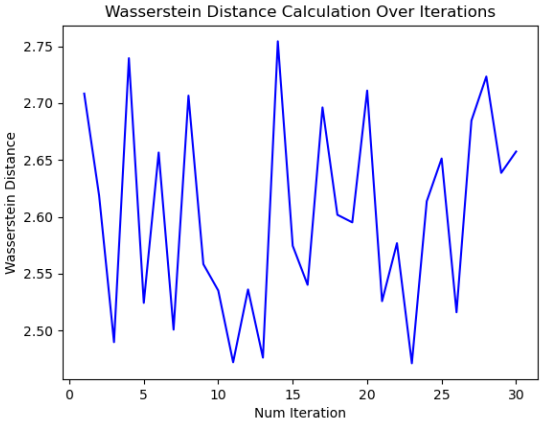

FIGURE V: GRAPH OF WASSERSTEIN CALCULATION STABILITY

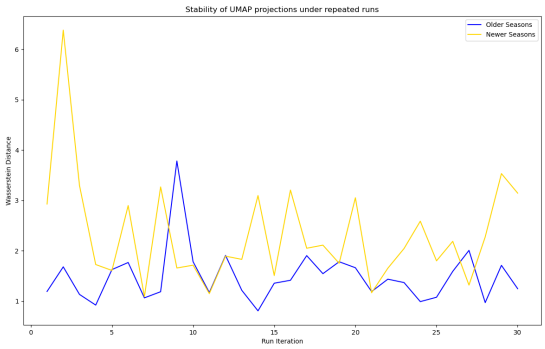

FIGURE VI: UMAP STABILITY

1.0 to 6.3—compared to the older seasons, which ranged from about 0.3 to 3.8. UMAP is designed to preserve local structure when reducing dimensionality; however, if repeated UMAP runs produce significantly different outputs, this may indicate high variability the underlying data. As newer seasons exhibit a wider range of distance, it may suggest that it has a larger variability in the words and phrases compared to that of older seasons. This may be due to the fact that Season 8 of *SpongeBob* aired in 2012, while Season 13 aired in 2020—highlighting that even within our defined "newer seasons" range, a semantic shift likely occurred over time. This variance is also apparent when observing the shapes of Figure 1 and Figure 2. As previously discussed, the data from older seasons appears more condensed compared to that of newer season data. This difference aligns with our experiment, observing UMAP's non-deterministic behavior, where larger distance calculations can suggest high data variability.

V. Conclusions and Future Work

Word2Vec embeddings trained on scripts from different time periods revealed a measurable shift in word usage between early and later seasons. This semantic change suggests that the show's language evolved over time, possibly reflecting changes in tone, character dynamics, or intended audience.

Dimensionality reduction via UMAP allowed us to visualize the high-dimensional word embeddings. By applying Gaussian kernel density estimation (KDE), we observed noticeable differences in the distribution of word clusters between older and newer scripts. These differences point to shifts in semantic density, highlighting how certain themes or types of language became more or less prominent over time.

To quantify the difference between the semantic distributions, we used the Sliced Wasserstein distance. Our results confirmed that the probability distributions from older and newer seasons are distinct, but not large. This method provided a mathematical measure of some semantic change that aligns with visual observations of the graphs.

The combination of UMAP scatterplots and KDE visualization proved effective for visually interpreting complex semantic patterns. The graphs allowed us to see semantic clusters and assess how word meaning and usage changed across the show's lifespan.

We developed a general framework through the use of Word2Vec embeddings, UMAP projection, KDE estimation, and Wasserstein-based distributional comparison. This can be applied to other structured scripts, including film scripts, political speeches, or social media discourse. We can use this framework to study semantic evolution over time.

Future Work Ideas

We plan to compare UMAP with other nonlinear dimensionality reduction techniques such as t-SNE and PCA. This will help assess how different visualizations impact perceived and measured semantic shifts. Future studies could track how specific word vectors evolve season by season, revealing deeper trends in semantic change over time.

By sorting dialogue by speaker, we can analyze how each character's vocabulary differs and evolves throughout the show. This may reveal character-specific linguistic differences or shifts in narrative focus.

Comparing Word2Vec with alternative embedding models such as GloVe or FastText may reveal whether the observed semantic change is dependent on a specific model. Embeddings could be used to train classifiers that predict the season or character of a given quote.

Additionally, generative models might use these embeddings to produce style-specific dialogue, simulating "classic" or "modern" SpongeBob speech. We can study the semantic shift compared with linguistic theory to attempt to explain what meanings are changing and why. In the future, we can do more research to find out why there is variance when running Word2Vec multiple times.

Finally, using our framework, we can analyze how the semantic meaning of other shows or even other forms of media evolve throughout their lifespan.

Acknowledgements

We would like to thank Professor Arsuaga and Teaching Assistant Zhijie Wang for their support.

References

[1] Gensim Developers, *Gensim: Topic Modelling for Humans (GitHub Repository)*, Available at: `https://github.com/RaRe-Technologies/gensim`.

[2] L. McInnes, J. Healy, and J. Melville, *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*, arXiv preprint arXiv:1802.03426, 2018. Available at: `https://arxiv.org/abs/1802.03426`.

[3] POT Developers, *POT: Python Optimal Transport (GitHub Repository)*, Available at: `https://github.com/PythonOT/POT`.

[4] SciPy Developers, *SciPy GitHub Repository*, Available at: `https://github.com/scipy/scipy`.

[5] UMAP Developers, *UMAP GitHub Repository*, Available at: `https://github.com/lmcinnes/umap`.