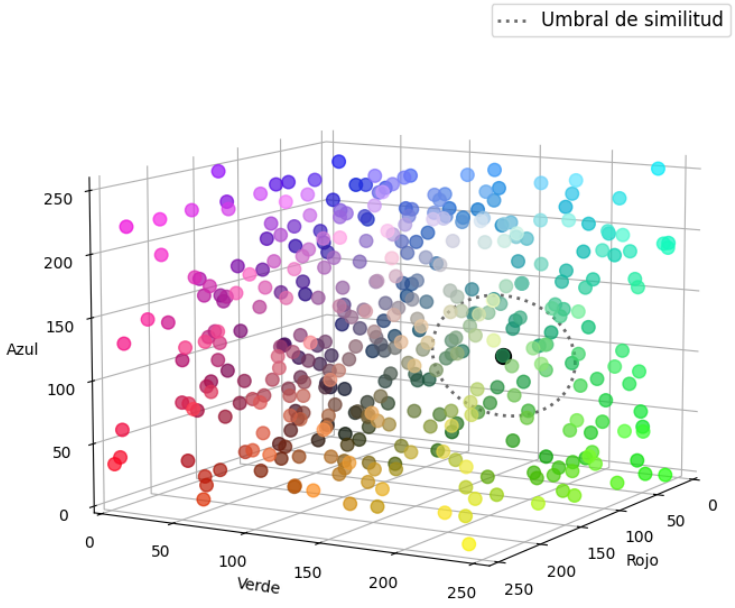
Workshop III: Embeddings

Machine Learning II – University of Antioquia
Anderson Torres Sánchez; Álvaro Gómez Peñuela; Héctor Mauricio Rendon

1. **In your own words, describe what vector embeddings are and what they are useful for.**

Machine learning algorithms work with numbers, and at their core, vector embeddings are how we translate data like text and images into numbers that a computer can understand but keeping its context. Let's understand this intuitively with the following example

We are going to represent the colors in a 3D space. Each axis will represent the amount of red, green, and blue (RGB) that makes up each color. For example, Magenta is represented by (255, 0, 255) and (80, 200, 120) represents Emerald. The resulting three-dimensional image would look like this



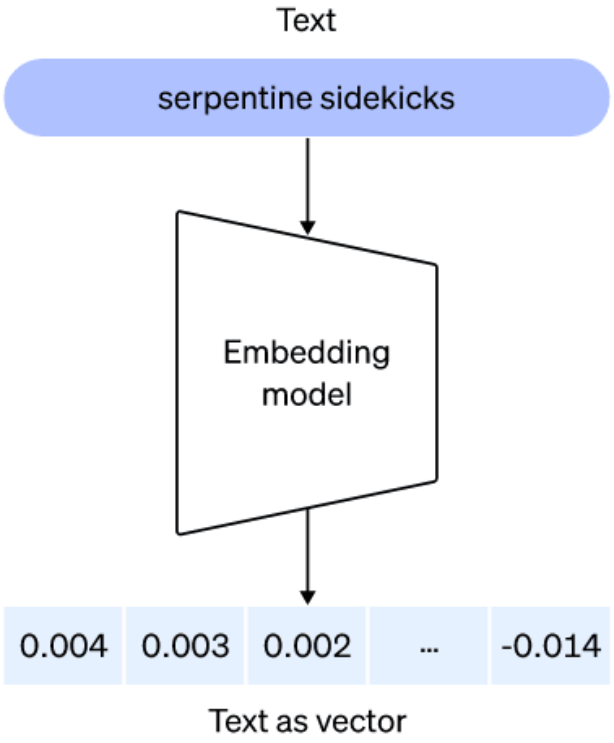
Note that the location of these points in the space based on their color allows an easy and objective evaluation of similarity and this is why a vector representation of data is so powerful: if you need to find colors like a certain color, you just must calculate the distance and accumulate points according to a threshold, since similar colors are physically close to each other.

Colors are particularly easy to represent in a vector space. But words, images, or documents are represented by high dimension vector in a high dimension space that are derived by machine learning to be spatially distributed by similarity.

With the notion of being able to represent information especially where closeness equals similarity, we can now define what embedding is.

In machine learning, an embedding is a way of representing data as points in n-dimensional space so that similar data points cluster together (it is, where the distances between points are meaningful).

The vector embedding is a representation in a vector space of data in numerical forms. By converting text into vector embeddings, for example, NLP models can easily perform tasks such as querying, classification, and applying machine learning algorithms on textual data. So, a vector embedding is nothing more than a mathematical vector generated to be used in machine-learning tasks.



So far embeddings allow us to find similar data points, now let's look at their applications: imagine a movie recommendation system that, given the synopsis of a movie, returns 10 similar movies based on proximity. Vector embedding can be used for:

- **Sentiment analysis**: Vector embeddings are crucial for sentiment analysis. They enable models to understand the connotation and polarity of words in a text, which is essential for determining whether a text has a positive, negative, or neutral emotional tone
- **Text classification**: In tasks such as news categorization, email sorting, or product review sentiment analysis, vector embeddings help models capture

important semantic and syntactic features to classify text into specific categories.

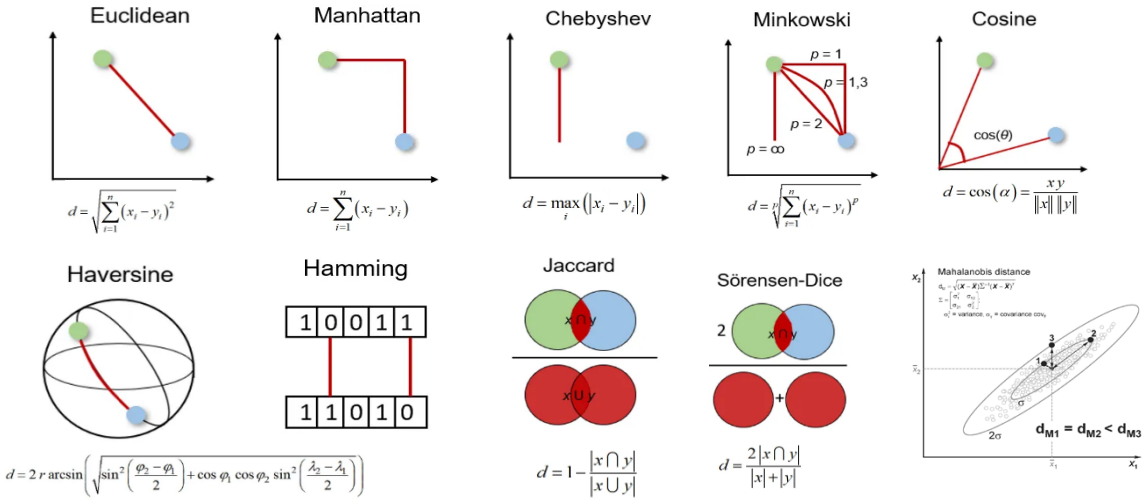
- **Translation**: Vector embeddings have significantly improved machine translation systems. By representing words more meaningfully, they facilitate the alignment and comprehension of phrases in different languages.
- **Text generation**: Vector embeddings are used in language generation models to produce coherent and contextually relevant text. For instance, chatgpt and similar models rely on word embeddings.
- **Q&A applications**: In question-answering (QA) systems, word embeddings enable understanding of the relationship between the question and the context, making it easier to find relevant answers.

Other uses are:

- **Text Clustering**: Vector embeddings allow for the grouping of similar documents or text fragments. This is useful in applications like news clustering, where it's important to group related or similar news articles.
- **Semantic Search**: In search engines or recommendation systems, word embeddings can help understand user context and intent, improving the accuracy of search results or recommendations.
- **Grammar Correction and Auto-completion**: Embeddings can be useful for autocorrection and auto-completion systems, aiding in predicting the next word in a sentence more accurately.
- **Named Entity Recognition (NER)**: They help in identifying and classifying entities such as names of people, places, or organizations in text.
- **Automatic Summarization**: Vector embeddings can be used for automatically summarizing long texts by identifying the most relevant and significant parts of the content.
- **Paraphrase Detection**: In tasks where identifying phrases with similar meanings but different words is important, word embeddings can be highly useful.
- **Virtual Assistance and Chatbots**: Vector embeddings are essential for understanding and generating coherent and contextually relevant responses in conversations with virtual assistants and chatbots.

2. **What do you think is the best distance criterion to estimate how far two embedding (vector) are form each other? Why?**

First, why is it important to calculate the distances between vector embeddings? It's because distance give as numerical similarity scores between embedded data, so the distance metrics used will affect the criteria of what is similar.



There are many types of distances to calculate how far apart two embeddings are, let's see some of them:

- **Euclidean Distance**:
  Formula: $\sqrt{\sum_i (x_i - y_i)^2}$
  Use Case: It's the straight-line distance between two points in space. It can be sensitive to the scale of the features, outliers, and high dimensional spaces. It's commonly used when the features are continuous, and the data distribution is roughly spherical.

- **Cosine Similarity**:
  Formula: $\frac{xy}{\|x\|\|y\|}$
  Use Case: It measures the cosine of the angle between two vectors. It's particularly useful when the magnitude of the vectors doesn't matter, only the direction. This is common in text analysis and information retrieval.

- **Manhattan Distance** (L1 Norm):
  Formula: $\sum_i |x_i - y_i|$

Use Case: it's calculated by summing the absolute distance between the components of two vectors. It's useful when the features are categorical, when you want to penalize differences in individual dimensions or when you want to measure distance along axes in a grid-like structure. It's also faster to calculate than L2 norm.

- **Mahalanobis Distance**:
  Formula: $\sqrt{(x-y)S^T(x-y)}$, where S^T is the covariance matrix of the data.
  Use Case: It's useful when there is correlation between the features or when the data is not isotropically distributed.

- **Jaccard Distance**:
  Formula: $1 - \frac{|A \cap B|}{|A \cup B|}$
  Use Case: Commonly used for comparing sets. It's particularly useful in tasks related to set similarity, such as document or text analysis.

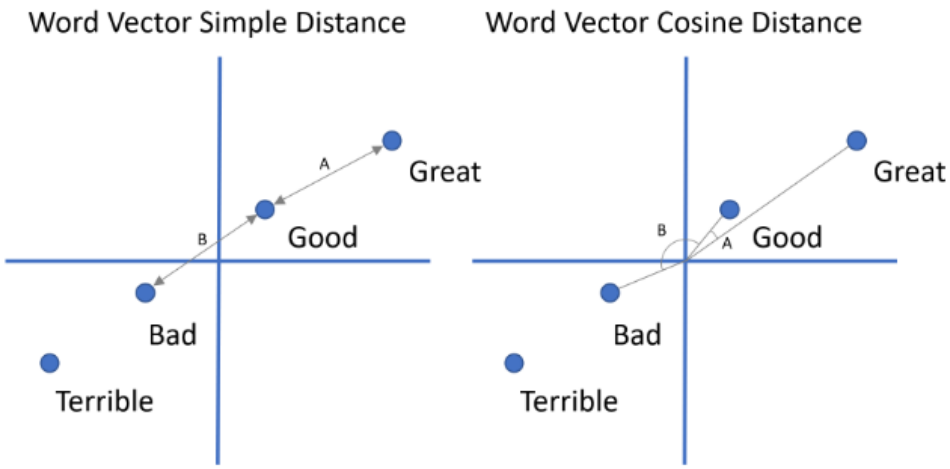
- **Correlation Distance**:
  Formula:$1 - Corr(x, y)$, where $Corr(x, y)$is the Pearson correlation coefficient.
  Use Case: It measures the dissimilarity between two vectors based on their correlation and is particularly useful when you want to compare the shape of the vectors.

Which one to use? There is not universally "best" distance metric, as different metrics are suitable for different scenarios and the general rule is use the metric with which the embedding was trained. A very common distance metric in text embedding is cosine, let's see why:

Cosine distance only cares about the angle between two vectors in a multi-dimensional space. It means that only the direction defines similarity and vector magnitudes are irrelevant. For example, two vectors with similar direction but far apart in magnitude will be interpreted as close regardless of how far apart they are in magnitude (like the word *good* and *great* at below image).

Many embedding spaces place the similar data by orientation rather than magnitude, then cosine similarity is a more suitable distance metric. It's the orientation of the vector is more indicative of the semantic similarity of the entities it represents.

For example, consider the embedded words vectors "good": [0.5, 0.7, 0.3] and "great": [0.6, 0.8, 0.4]. The Euclidean distance between these two vectors is 0.31, but the cosine similarity is 0.98. This is because the two vectors have a very similar orientation, even though they have different magnitudes. In 2D, this looks like in the image below



If we were to use Euclidean distance to measure the similarity, “Bad” and “Great” seems to be just as similar to “Good”. However, if we use cosine similarity, we get a very high score, which is more accurate.

Summing up, embedding spaces often focus on representing data in a way that emphasizes similarity based on orientation rather than magnitude. That’s why cosine metric is more suitable for capturing this kind of similarity in embedding spaces.

3. **Q&A (question answering) system:**

   a. **Pick whatever text you like, in the order of 20+ paragraphs**

      For this item, ChatGPT was used to recommend a text with more than 20 paragraphs and the text that was selected was the classic novel “Pride and Prejudice" by Jane Austen.

   b. **Split that text into meaningful chunks/pieces**

      ChatGPT delivered the text organized by paragraphs, where each line is a paragraph of the original text, therefore, it was decided to divide this text into these paragraphs.

   c. **Implement the embedding generation logic. Which tools and approaches would help you generate them easily and high-level?**

      One popular method is using word embeddings algorithms, such as Word2Vec, GloVe, or FastText, and then aggregating the word embeddings to form a sentence-level vector representation. Another common approach is to use pre-trained language models, like BERT or GPT, which can provide contextualized embeddings for entire sentences.

      Pre-trained language models can provide better contextualized embeddings for sentences. This type of trained models can be found in the page 'Hugging face', in this case we will use the category 'question-answering' and we will use the model 'distilbert-base-cased-distilled-squad' which is one of the best valued of the page.

      We will implement a logic of embedding generation using the library 'transformer' with which we can use the models developed and shared by the page 'Hugging face'.

   d. **For every question asked by the user, return a sorted list of the N chunks/pieces in your text that relate the most to the question. Do results make sense?**

      The results make sense because the chunk/pieces that appear when the question is generated or chunk/pieces in which the words that were used in the question appear, since the separation was made by paragraph and not according to a semantic order, the answers are not adequate at a semantic level, but at a grammatical level.

4. **What do you think that could make these types of systems more robust in terms of semantics and functionality?**

Improving the robustness of vector word embedding systems, such as Word2Vec, GloVe, or more advanced models like BERT and GPT, is a complex and ongoing research challenge.

Some strategies and considerations that could enhance the robustness of such systems in terms of semantics and functionality:

- **Diverse Training Data**: Use a diverse and comprehensive corpus for training word embeddings. Ensure that the data includes a wide range of domains, languages, and writing styles to capture a broader spectrum of semantics and concepts.
- **Multilingual Training**: Train embeddings on multilingual corpora to enable cross-lingual transfer of semantic information. This can help improve the understanding of words and phrases in multiple languages.
- **Contextual Embeddings**: Consider using contextual embeddings from models like BERT, GPT, or RoBERTa, which capture word meanings based on their surrounding context. These embeddings tend to have better semantic understanding than static word embeddings.
- **Subword Embeddings**: Incorporate subword embeddings (e.g., FastText) to handle out-of-vocabulary words and morphological variations. This can improve the coverage and robustness of the embeddings.
- **Fine-Tuning**: Fine-tune pre-trained embeddings on domain-specific data or tasks to adapt them to specific use cases. Fine-tuning allows you to capture domain-specific semantics.
- **Dynamic Word Embeddings**: Explore techniques that allow embeddings to change dynamically based on context or time, reflecting evolving language semantics. This can be crucial for handling semantic drift.
- **Evaluation Metrics**: Develop and use more sophisticated evaluation metrics that assess not only syntactic but also semantic properties of word embeddings. Metrics like word similarity, analogy, and downstream task performance can provide a better assessment of semantic quality.
- **Bias Mitigation**: Address and mitigate biases present in word embeddings, such as gender or racial biases, to ensure that embeddings are more fair and representative of the real world.
- **Conceptual Understanding**: Incorporate external knowledge sources, such as knowledge graphs or ontologies, to enhance the embeddings' understanding of concepts and relationships between words.
- **Adversarial Testing**: Subject word embeddings to adversarial testing to identify vulnerabilities and improve their robustness against attacks or perturbations.
- **Regularization Techniques**: Apply regularization techniques to reduce overfitting during training and improve the generalization of word embeddings.

- **Human-in-the-Loop**: Involve human experts in the development and validation process to ensure that the embeddings capture the intended semantics and are free from biases.
- **User Feedback**: Continuously gather user feedback to identify and rectify any issues or limitations in the embeddings' semantics and functionality.
- **Interdisciplinary Collaboration**: Foster collaboration between linguists, computer scientists, and domain experts to ensure a holistic approach to improving word embeddings.
- **Ethical Considerations**: Be mindful of the ethical implications of word embeddings and ensure that their usage respects privacy, diversity, and fairness principles.

Improving the robustness of word embedding is a ongoing and multifaceted challenge, and it requires a combination of data, model architecture, evaluation methods, and ethical considerations to achieve meaningful progress in capturing and representing semantics accurately.