Embeddings for similarity

Understanding Embeddings and similarity

- **Numerical Representations:** Embeddings are dense numerical vectors that represent words or entire pieces of text. These vectors capture the semantic meaning and context of the text.
- **Semantic Similarity:**
  Embeddings encode meaning in a way that lets you calculate how similar or different pieces of text are. Texts covering similar topics will have embeddings that are closer together in the vector space.

Generation of embeddings

Much of the embedding functionality is handled by the transformers library.

Install the `transformers` library with `pip install transformers`

```
from transformers import AutoTokenizer, AutoModel

model_name: str = "dmis-lab/biobert-v1.1"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)
```

To generate the embeddings for a single medication you can use the following syntax:

```
medication = 'paracetamol'

tokens = tokenizer(
            [medication], padding=True, truncation=True, return_
        )
```

```
  outputs = self.model(**tokens)
  embeddings = outputs.last_hidden_state.mean(dim=1).detach().nu
```

Clustering Embeddings

Clustering algorithms allow you to find groups of related items based on their embeddings.

1. **Clustering Algorithm:** Common choices for clustering embeddings include:
    - **K-Means:** Simple, popular method. You specify the number of clusters (`k`). The algorithm tries to find clusters with minimal intra-cluster distances.
    - **Hierarchical Clustering:** Creates a tree-like structure (dendrogram) showcasing relationships between the embeddings. Useful when you don't know the optimal number of clusters beforehand.
2. **Implementation (Example using scikit-learn):**

```
from sklearn.cluster import KMeans

# Assume you have 'embeddings' from your previous code as a N

kmeans = KMeans(n_clusters=5) # Example: 5 clusters
kmeans.fit(embeddings)

cluster_labels = kmeans.labels_ # Gives you the cluster index
```

Similarity Investigation

To determine how similar two pieces of text are, you typically calculate a similarity metric on their embeddings.

1. **Cosine Similarity:** A standard metric that measures the angle between embedding vectors. A value of 1 means identical; 0 means completely orthogonal (no similarity).
2. **Implementation (Example using scikit-learn):**

```
from sklearn.metrics.pairwise import cosine_similarity

embedding1 = ...  # Embedding of text 1
embedding2 = ...  # Embedding of text 2

similarity = cosine_similarity([embedding1], [embedding2])[0]
print(f"Similarity Score: {similarity}")
```

Important Considerations

- **Choice of Embedding Model:** The BioBERT model used above is tailored for biomedical text. For general text, other models (e.g., SentenceTransformers) might be more suitable. You should experiment with models in hugging face library.
- **Visualisation:** To visually explore clusters and similarities, consider dimensionality reduction techniques like t-SNE or UMAP.
- **Hyperparameter Tuning:** Clustering algorithms like K-Means often require you to experiment with their hyperparameters (like the number of clusters).