**Semantic Search, Classification, and Recommendation Systems Using Free/Open-Source Embeddings: Detailed Guide with Code**

**1. Semantic Search**

**What is Semantic Search?**  
Semantic search finds documents or items that are *similar in meaning* to a user’s query, not just those that share the same keywords. This is done by converting text into embeddings (vectors) and measuring their similarity.

**How does Cosine Similarity work?**  
Cosine similarity measures the angle between two vectors (embeddings). If the vectors point in the same direction (angle close to 0°), the similarity is 1 (very similar). If they are at 90°, the similarity is 0 (not similar). If they are opposite (180°), the similarity is -1 (completely different).  
Mathematically:

Where $ \vec{A} \cdot \vec{B} $ is the dot product, and $ ||\vec{A}|| $ is the vector magnitude

**Python Example: Semantic Search with Sentence Transformers**

from sentence\_transformers import SentenceTransformer  
from sklearn.metrics.pairwise import cosine\_similarity  
import numpy as np  
  
# Load a pre-trained embedding model  
model = SentenceTransformer('all-MiniLM-L6-v2')  
  
# Example corpus and query  
documents = [  
 "Machine learning is fascinating.",  
 "I love playing football.",  
 "Artificial intelligence and machine learning are related fields.",  
 "Cooking is an art."  
]  
query = "What is machine learning?"  
  
# Encode documents and query  
doc\_embeddings = model.encode(documents)  
query\_embedding = model.encode([query])  
  
# Compute cosine similarity  
similarities = cosine\_similarity(query\_embedding, doc\_embeddings)[^0]  
  
# Get top match  
top\_idx = np.argmax(similarities)  
print(f"Best match: {documents[top\_idx]} (Score: {similarities[top\_idx]:.2f})")

This code finds the document most semantically similar to the query using cosine similarity

**2. Classification with Random Forest**

**How does Random Forest work?**  
Random Forest is an ensemble of decision trees:

* Each tree is trained on a random subset of the data (bootstrapping).
* At each split, only a random subset of features is considered.
* For classification, each tree votes for a class; the majority wins.
* This approach reduces overfitting and increases robustness[[4]](#fn4).

**Why use Random Forest?**

* Handles both numerical and categorical data.
* Robust to outliers and missing values.
* Works well out-of-the-box for many tasks.
* Less likely to overfit than a single tree.

**Alternatives:**

* **Logistic Regression:** Simple, interpretable, good for binary classification.
* **Gradient Boosting (e.g., XGBoost, LightGBM):** Often higher accuracy, but more complex.
* **Neural Networks:** For very large or complex datasets (e.g., images, text).

**Python Example: Text Classification with Embeddings + Random Forest**

from sentence\_transformers import SentenceTransformer  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, classification\_report  
  
# Example data  
texts = [  
 "Football match tonight.",  
 "Stock market crashes.",  
 "New AI research published.",  
 "The team won the championship."  
]  
labels = ["sports", "finance", "tech", "sports"]  
  
# Encode texts  
model = SentenceTransformer('all-MiniLM-L6-v2')  
X = model.encode(texts)  
  
# Train-test split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, labels, test\_size=0.25, random\_state=42)  
  
# Random Forest classifier  
clf = RandomForestClassifier(n\_estimators=100, random\_state=42)  
clf.fit(X\_train, y\_train)  
  
# Predict and evaluate  
y\_pred = clf.predict(X\_test)  
print("Accuracy:", accuracy\_score(y\_test, y\_pred))  
print("Classification Report:\n", classification\_report(y\_test, y\_pred))

This example uses embeddings as features for Random Forest, which then classifies the text[[4]](#fn4).

**3. Recommendation Systems**

**How do Recommendation Systems work?**  
There are two main approaches:

* **Content-based filtering:** Recommends items similar to those the user liked, based on item features (e.g., text, genre)[[5]](#fn5).
* **Collaborative filtering:** Recommends items that similar users liked, based on user behavior (e.g., ratings, purchases)[[6]](#fn6).
* **Hybrid systems:** Combine both methods for better results.

**Content-based Example with Embeddings:**  
Embeddings capture the meaning of items (e.g., articles, products). By measuring cosine similarity between user preferences and item embeddings, you can recommend the most similar items.

**Python Example: Content-Based Recommendation with Sentence Transformers**

from sentence\_transformers import SentenceTransformer  
from sklearn.metrics.pairwise import cosine\_similarity  
import numpy as np  
  
# Example items  
items = [  
 "Deep learning for image recognition.",  
 "Healthy recipes for breakfast.",  
 "Understanding neural networks.",  
 "Yoga and mindfulness practices."  
]  
  
# User profile (could be a single string or aggregation of past liked items)  
user\_profile = "I want to learn about artificial intelligence and neural networks."  
  
# Encode items and user profile  
model = SentenceTransformer('all-MiniLM-L6-v2')  
item\_embeddings = model.encode(items)  
user\_embedding = model.encode([user\_profile])  
  
# Compute similarity  
scores = cosine\_similarity(user\_embedding, item\_embeddings)[^0]  
top\_indices = np.argsort(scores)[::-1] # Descending order  
  
# Recommend top 2 items  
for idx in top\_indices[:2]:  
 print(f"Recommended: {items[idx]} (Score: {scores[idx]:.2f})")

This code recommends items most similar to the user’s interests using embeddings and cosine similarity[[7]](#fn7)[[8]](#fn8)[[9]](#fn9).

**4. Cosine Similarity: Simple Python Function**

import numpy as np  
  
def cosine\_similarity(vector1, vector2):  
 dot\_product = np.dot(vector1, vector2)  
 magnitude\_a = np.linalg.norm(vector1)  
 magnitude\_b = np.linalg.norm(vector2)  
 return dot\_product / (magnitude\_a \* magnitude\_b)  
  
vector\_A = [1, 2, 3]  
vector\_B = [4, 5, 6]  
similarity = cosine\_similarity(vector\_A, vector\_B)  
print("Cosine similarity:", similarity)

This function shows how cosine similarity is computed mathematically[[1]](#fn1).

**5. Production Models Commonly Used**

|  |  |
| --- | --- |
| Use Case | Common Production Models |
| Semantic Search | Sentence Transformers (e.g., all-MiniLM-L6-v2), Universal Sentence Encoder, BERT variants |
| Classification | Random Forest, Logistic Regression, Gradient Boosting (XGBoost, LightGBM), Neural Networks |
| Recommendation | Sentence Transformers for content-based, Matrix Factorization, Neural Collaborative Filtering, Hybrid models |

**Summary for Beginners:**

* **Semantic search** uses embeddings and cosine similarity to find meaningfully similar documents.
* **Random Forest** is a robust, easy-to-use classifier that combines many decision trees to improve accuracy and reduce overfitting.
* **Recommendation systems** suggest items by comparing embeddings (content-based) or using user behavior (collaborative filtering).
* **Cosine similarity** measures how close two vectors are in direction, not magnitude; perfect for comparing semantic meaning.

All code examples use free and open-source models and Python libraries, ideal for learning and prototyping

1. <https://builtin.com/machine-learning/cosine-similarity>

1. <https://www.sbert.net/examples/sentence_transformer/applications/semantic-search/README.html>

1. <https://sbert.net/docs/sentence_transformer/usage/semantic_textual_similarity.html>

1. <https://www.sitepoint.com/random-forest-algorithm-in-machine-learning/>

1. <https://thecleverprogrammer.com/2023/04/20/content-based-filtering-and-collaborative-filtering-difference/>

1. <https://www.ibm.com/think/topics/collaborative-filtering>

1. <https://milvus.io/ai-quick-reference/how-can-sentence-transformers-help-in-building-a-recommendation-system-for-content-such-as-articles-or-videos-based-on-text-similarity>

1. <https://www.kaggle.com/code/vineethakkinapalli/sentence-transformer-based-recommender-system>

1. <https://github.com/manavisrani07/Content-Based-Recommendation-System>