TF-IDF: Theory and Mathematics

The `TfidfVectorizer` from `sklearn.feature_extraction.text` is a tool used in natural language processing (NLP) to convert a collection of raw documents into a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features. This matrix is often used as input for machine learning algorithms.

1. Term Frequency (TF)

Term Frequency measures how frequently a term t appears in a document d. It can be computed in several ways, but the most common method is:

$$\text{TF}(t, d) = \frac{f_{t,d}}{N_d}$$

where:

- $f_{t,d}$ is the raw count of term t in document d.
- N_d is the total number of terms in document d.

2. Inverse Document Frequency (IDF)

Inverse Document Frequency measures how important a term is within the entire corpus. It is computed as:

$$\text{IDF}(t, D) = \log\left(\frac{N}{|\{d \in D : t \in d\}|}\right)$$

where:

- N is the total number of documents.
- $\{d \in D : t \in d\}$ is the number of documents containing the term t.

This term is often adjusted to prevent division by zero and to smooth out extreme values. A common adjustment is:

$$\text{IDF}(t, D) = \log\left(\frac{N}{1 + |\{d \in D : t \in d\}|}\right) + 1$$

3. TF-IDF

The TF-IDF score for a term t in a document d is then the product of its TF and IDF scores:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

Using TfidfVectorizer in scikit-learn

`TfidfVectorizer` in scikit-learn automates the process of computing the TF-IDF scores for a collection of documents. Here is how it works step-by-step:

1. **Tokenization**: The text is split into individual terms (words).
2. **Building Vocabulary**: A vocabulary of all unique terms across the documents is created.
3. **Calculating Term Frequencies**: The term frequency for each term in each document is calculated.
4. **Calculating IDF**: The IDF for each term in the vocabulary is calculated based on the entire document collection.
5. **Computing TF-IDF**: The TF-IDF score for each term in each document is computed.

Example in Python

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample documents
documents = [
    "The cat sat on the mat",
    "The dog chased the cat",
    "The cat climbed the tree"
]

# Create a TfidfVectorizer
vectorizer = TfidfVectorizer()

# Fit and transform the documents
tfidf_matrix = vectorizer.fit_transform(documents)

# Convert to a dense matrix and print
print(tfidf_matrix.toarray())

# Get the feature names (terms)
print(vectorizer.get_feature_names_out())
```

Output:

```
[[0.27116066 0.         0.         0.         0.4591149  0.4591149
  0.4591149  0.54232132 0.        ]
 [0.30523155 0.51680194 0.         0.51680194 0.         0.
  0.         0.61046311 0.        ]
 [0.30523155 0.         0.51680194 0.         0.         0.
  0.         0.61046311 0.51680194]]
['cat' 'chased' 'climbed' 'dog' 'mat' 'on' 'sat' 'the' 'tree']
```

AUC (Area Under the Curve)

The AUC, or Area Under the Curve, is a performance measurement for classification problems at various threshold settings. AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. The higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s.

ROC (Receiver Operating Characteristics)

ROC, or Receiver Operating Characteristics, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

1. True Positive Rate (TPR)

True Positive Rate, also known as Sensitivity or Recall, is the ratio of correctly predicted positive observations to all the actual positives. It is calculated as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

where:

- TP = True Positives
- FN = False Negatives

2. False Positive Rate (FPR)

False Positive Rate is the ratio of incorrectly predicted positive observations to all the actual negatives. It is calculated as:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

where:

- FP = False Positives
- TN = True Negatives

AUC-ROC Curve

The AUC-ROC curve is used for binary classification to visualize the performance of a classifier. A higher AUC value indicates a better performing model. An AUC value of 1 represents a perfect model, while an AUC value of 0.5 represents a model that performs no better than random guessing.

F1 Score

The F1 Score is the harmonic mean of Precision and Recall. It considers both false positives and false negatives and is especially useful when the class distribution is imbalanced. The F1 Score can be interpreted as a weighted average of Precision and Recall, where an F1 Score reaches its best value at 1 and worst score at 0.

1. Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It is calculated as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

2. Recall

Recall (Sensitivity) is the ratio of correctly predicted positive observations to all the actual positives. It is calculated as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

3. Calculating F1 Score

The F1 Score is calculated as:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Example in Python

```
from sklearn.metrics import roc_auc_score, f1_score
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Create a sample dataset
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train a RandomForest classifier
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_train, y_train)

# Predict probabilities
y_pred_prob = classifier.predict_proba(X_test)[:, 1]

# Predict classes
y_pred = classifier.predict(X_test)

# Calculate AUC-ROC
auc = roc_auc_score(y_test, y_pred_prob)
print("AUC-ROC:", auc)

# Calculate F1 Score
f1 = f1_score(y_test, y_pred)
print("F1 Score:", f1)
```

Output:

```
AUC-ROC: 0.9825
F1 Score: 0.9685
```