```
import nltk
```

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import numpy as np
```

```
# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
stop_words.update(['is', "n't"])  # Add 'is' and 'n't' to stopwords
stemmer = PorterStemmer()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# Function to preprocess text based on provided parameters
def preprocess_text(text, to_stem, to_remove_stop_words):
  words = text.lower().split(' ')  # Tokenize and lowercase
  if to_stem:
    words = [stemmer.stem(word) for word in words]  # Stemming
  if to_remove_stop_words:
    words = [word for word in words if word not in stop_words]  # Remove stopwords
  return " ".join(words)
```

```
# Function to encode text using different vector encodings
def encode_text(text, to_stem=False, to_remove_stop_words=False, encoding_type='one-hot'):
  preprocessed_text = preprocess_text(text, to_stem, to_remove_stop_words)
  vectorizer = None
  if encoding_type == 'one-hot':
    vectorizer = CountVectorizer(binary=True)
  elif encoding_type == 'bag of words':
    vectorizer = CountVectorizer()
  elif encoding_type == 'tf':
    vectorizer = CountVectorizer()
  elif encoding_type == 'tfXidf':
    vectorizer = TfidfVectorizer()

  if vectorizer:
    vectorized_text = vectorizer.fit_transform([preprocessed_text])
    return vectorized_text.toarray()[0]  # Convert to dense array
  else:
    return None
```

```
def cosine_similarity(vector1, vector2):
    # Pad the shorter vector with zeros to match dimensions
    if len(vector1) < len(vector2):
        vector1 = np.pad(vector1, (0, len(vector2) - len(vector1)), mode='constant')
    elif len(vector2) < len(vector1):
        vector2 = np.pad(vector2, (0, len(vector1) - len(vector2)), mode='constant')

    # Normalize vectors
    norm_vector1 = vector1 / np.linalg.norm(vector1)
    norm_vector2 = vector2 / np.linalg.norm(vector2)

    # Compute dot product
    dot_product = np.dot(norm_vector1, norm_vector2)

    return dot_product
```

```
# Example test data
sentence1 = "A dog eats a cat"
sentence2 = "A cat hasn't chased after a dog"
```

```
# Test function with different scenarios
def test_encoding(scenario):
  print(f"Scenario: {scenario}")
```

```
  print("Sentence 1:", sentence1)
  print("Sentence 2:", sentence2)

  encoding_scenarios = {
      'stop-word removal increases recall': (False, True, 'one-hot'),
      'stop-word removal decreases precision': (False, True, 'one-hot'),
      'stemming increases recall': (True, False, 'one-hot'),
      'stemming decreases precision': (True, False, 'one-hot'),
      'bag of words has higher precision than one-hot encoding': (False, False, 'bag of words'),
      'tf has higher precision than bag of words': (False, False, 'tf'),
      'tfXidf has higher precision than tf': (False, False, 'tfXidf')
      }

  to_stem, to_remove_stop_words, encoding_type = encoding_scenarios[scenario]

  # Print preprocessed texts
  print("Preprocessed Sentence 1:", preprocess_text(sentence1, to_stem, to_remove_stop_words))
  print("Preprocessed Sentence 2:", preprocess_text(sentence2, to_stem, to_remove_stop_words))


  vec1 = encode_text(sentence1, to_stem, to_remove_stop_words, encoding_type)
  vec2 = encode_text(sentence2, to_stem, to_remove_stop_words, encoding_type)

  similarity_before = cosine_similarity(vec1, vec2)

  # Modify one sentence to simulate the effect
  if to_remove_stop_words:
      sentence1_modified = preprocess_text(sentence1, to_stem, False)  # Keep stopwords
      vec1_modified = encode_text(sentence1_modified, to_stem, False, encoding_type)
  else:
      sentence1_modified = preprocess_text(sentence1, to_stem, True)  # Remove stopwords
      vec1_modified = encode_text(sentence1_modified, to_stem, True, encoding_type)

  similarity_after = cosine_similarity(vec1_modified, vec2)

  print("Similarity Before:", similarity_before)
  print("Similarity After:", similarity_after)
  print()
```

```
# Test each scenario
test_encoding('stop-word removal increases recall')
test_encoding('stop-word removal decreases precision')
test_encoding('stemming increases recall')
test_encoding('stemming decreases precision')
test_encoding('bag of words has higher precision than one-hot encoding')
test_encoding('tf has higher precision than bag of words')
test_encoding('tfXidf has higher precision than tf')
```

```
Scenario: stop-word removal increases recall
Sentence 1: A dog eats a cat
Sentence 2: A cat hasn't chased after a dog
Preprocessed Sentence 1: dog eats cat
Preprocessed Sentence 2: cat chased dog
Similarity Before: 1.0000000000000002
Similarity After: 1.0000000000000002

Scenario: stop-word removal decreases precision
Sentence 1: A dog eats a cat
Sentence 2: A cat hasn't chased after a dog
Preprocessed Sentence 1: dog eats cat
Preprocessed Sentence 2: cat chased dog
Similarity Before: 1.0000000000000002
Similarity After: 1.0000000000000002

Scenario: stemming increases recall
Sentence 1: A dog eats a cat
Sentence 2: A cat hasn't chased after a dog
Preprocessed Sentence 1: a dog eat a cat
Preprocessed Sentence 2: a cat hasn't chase after a dog
Similarity Before: 0.7745966692414834
Similarity After: 0.7745966692414834

Scenario: stemming decreases precision
Sentence 1: A dog eats a cat
Sentence 2: A cat hasn't chased after a dog
Preprocessed Sentence 1: a dog eat a cat
Preprocessed Sentence 2: a cat hasn't chase after a dog
Similarity Before: 0.7745966692414834
```

```
Similarity After: 0.7745966692414834

Scenario: bag of words has higher precision than one-hot encoding
Sentence 1: A dog eats a cat
Sentence 2: A cat hasn't chased after a dog
Preprocessed Sentence 1: a dog eats a cat
Preprocessed Sentence 2: a cat hasn't chased after a dog
Similarity Before: 0.7745966692414834
Similarity After: 0.7745966692414834

Scenario: tf has higher precision than bag of words
Sentence 1: A dog eats a cat
Sentence 2: A cat hasn't chased after a dog
Preprocessed Sentence 1: a dog eats a cat
Preprocessed Sentence 2: a cat hasn't chased after a dog
Similarity Before: 0.7745966692414834
Similarity After: 0.7745966692414834

Scenario: tfXidf has higher precision than tf
Sentence 1: A dog eats a cat
Sentence 2: A cat hasn't chased after a dog
Preprocessed Sentence 1: a dog eats a cat
Preprocessed Sentence 2: a cat hasn't chased after a dog
Similarity Before: 0.7745966692414836
Similarity After: 0.7745966692414836
```