TEXT CLASSIFICATION USING NAÏVE BAYES

```
import nltk
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
from nltk.classify import NaiveBayesClassifier
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split

nltk.download('movie_reviews')
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess(text):
    tokens = word_tokenize(text.lower())
    tokens = [lemmatizer.lemmatize(token) for token in tokens if token not in stop_words and token.isalpha()]
    return dict(nltk.FreqDist(tokens))

pos_reviews = [(movie_reviews.raw(fileid), 'positive') for fileid in movie_reviews.fileids('pos')]
neg_reviews = [(movie_reviews.raw(fileid), 'negative') for fileid in movie_reviews.fileids('neg')]
tot_rev = pos_reviews + neg_reviews
processed_data = [(preprocess(text), category) for (text, category) in tot_rev]

train_data, val_data = train_test_split(processed_data, test_size=0.2, random_state=42)

# Train NB Classifier on training data
classifier = NaiveBayesClassifier.train(train_data)

new_text = ["The movie was amazing", "the movie was terrible", "The movie was awful"]

for text in new_text:
    new_features = preprocess(text)
    predicted_category = classifier.classify(new_features)
    print(f"The predicted category for '{text}' is '{predicted_category}'")
```

OUTPUT

```
The predicted category for 'The movie was amazing' is 'positive'
The predicted category for 'the movie was terrible' is 'negative'
The predicted category for 'The movie was awful' is 'negative'
```

TEXT CLASSIFICATION USING LOGISTIC REGRESSION

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

def preprocess(text):
    ps = PorterStemmer()
    stop_words = set(stopwords.words('english'))
    words = [word_tokenize(sentence) for sentence in text]
    filtered_words = [[ps.stem(word) for word in tokenized if word not in stop_words and word.isalpha()] for
tokenized in words]
    filtered_sentences = [' '.join(sentence) for sentence in filtered_words]
    return filtered_sentences

sentences = ["The food is tasty", "the quality of food is low", "i will never recommend their food",
             "I got sick after having their food", "I was in cloudnine after tasting their food",
             "My favourite is their desserts", "the food was not cooked properly"]
classes = [1, 0, 0, 0, 1, 1, 0]
test_sentences = ["food is not cooked properly", "I feel sick after having food", "I love their desserts",
                  "was in cloudnine after tasting their food"]

vectorizer = CountVectorizer()
sentences = preprocess(sentences)
vect1 = vectorizer.fit_transform(sentences)

# Splitting data for testing
# train_data, test_data, train_labels, test_labels = train_test_split(vect1, classes, test_size=0.2,
random_state=42)

nb = LogisticRegression()
nb.fit(vect1, classes)

test_sentences = preprocess(test_sentences)
vect2 = vectorizer.transform(test_sentences)

pred_classes = nb.predict(vect2)
print(pred_classes)
```

OUTPUT

```
[0 0 1 1]
```