EX.NO.: 16
DATE: 14.03.2025

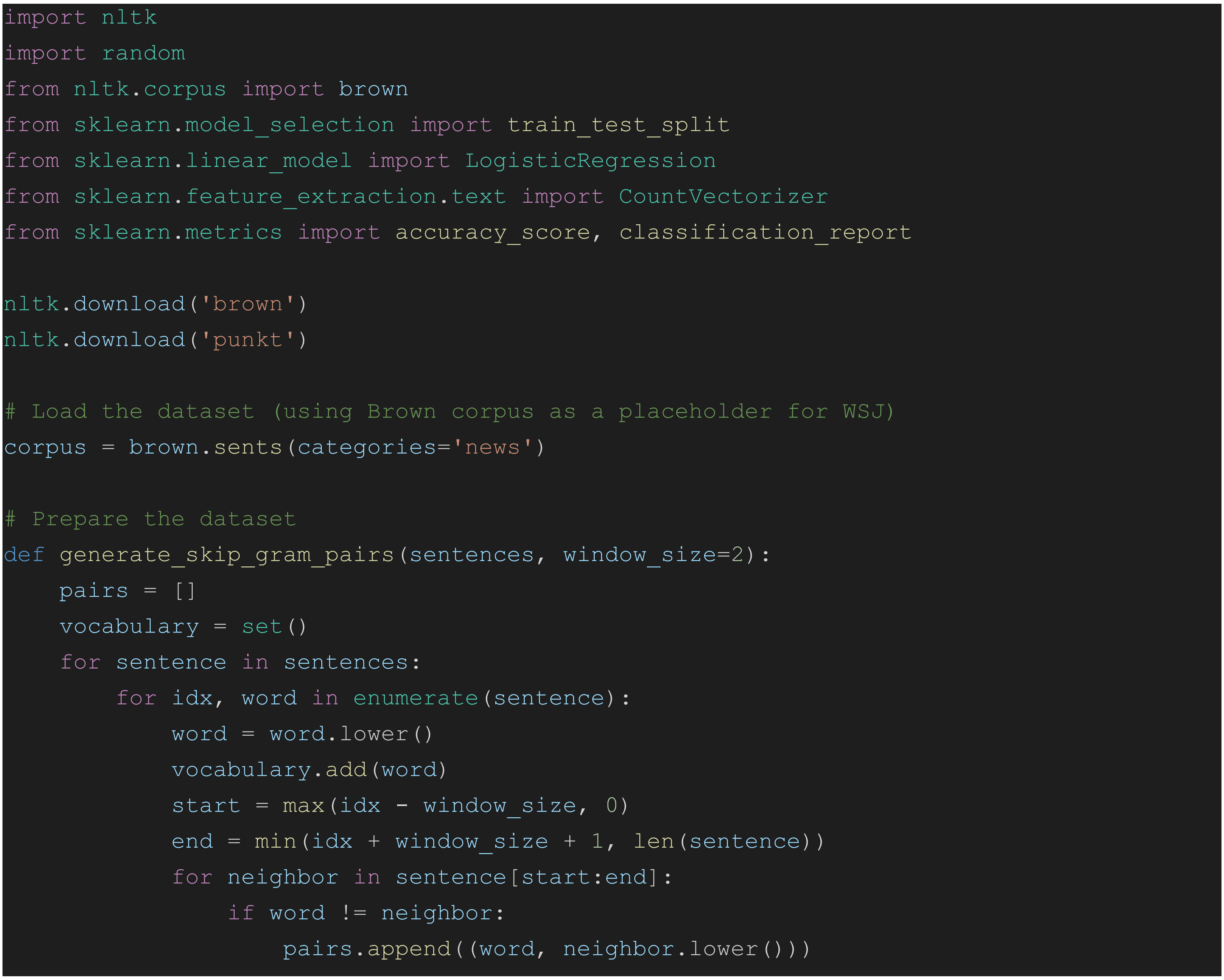
LOGISTIC REGRESSION - SKIP GRAM

Train a Logistic Regression classifier using Skip-Gram pairs to predict the probability of two words appearing nearby in text.

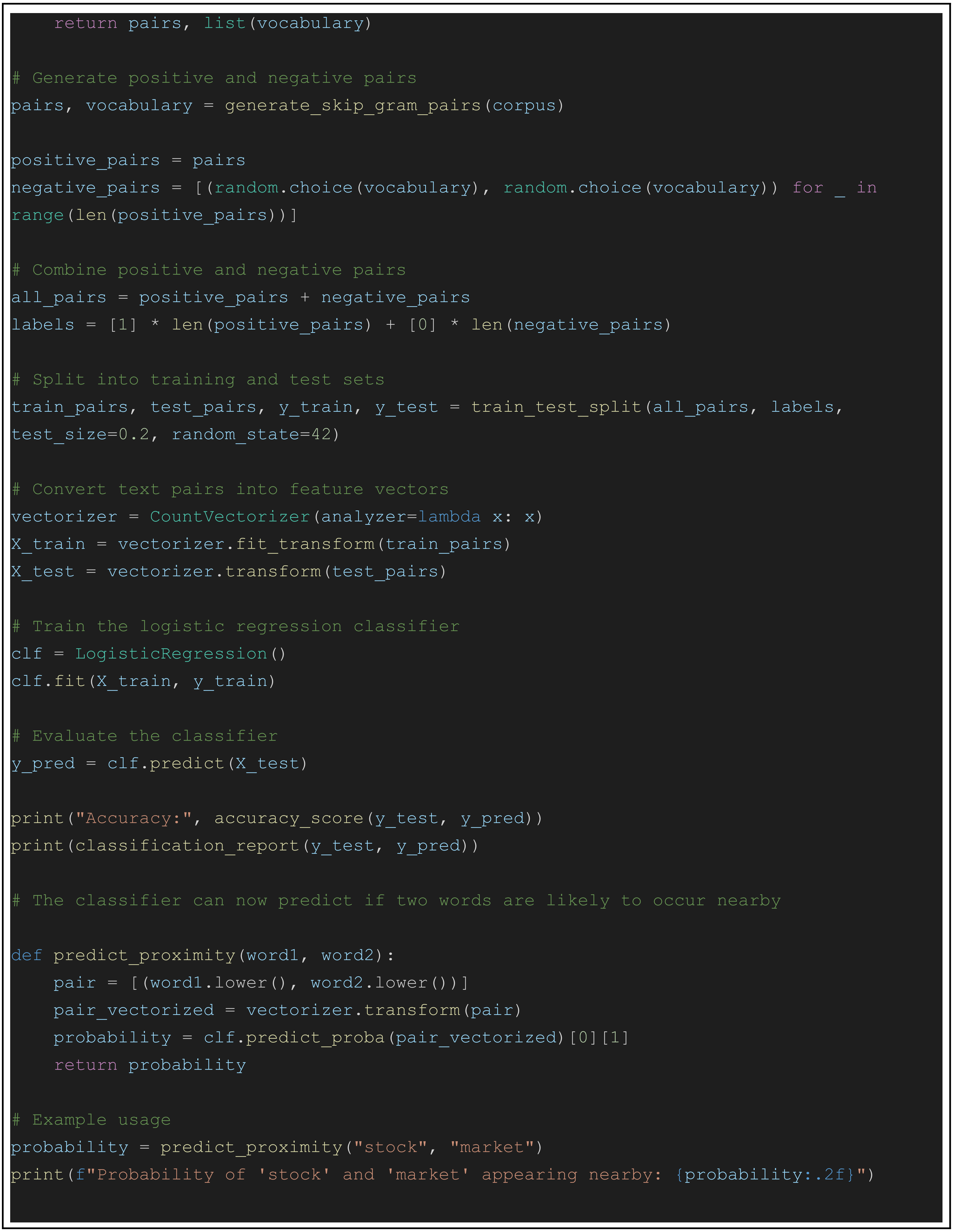
PROCEDURE:

1. Load the dataset (Reuters corpus as a proxy for WSJ)
2. Tokenize the text
3. Flatten the list
4. Train a lightweight Word2Vec Skip-Gram model
5. Generate Skip-Gram pairs
6. Function to get word embeddings
7. Prepare feature vectors
8. Generate binary labels (1 for actual pairs, 0 for negative sampling)
9. Add negative samples for balancing
10. Combine positive and negative samples
11. Split data into training and testing sets
12. Train Logistic Regression model
13. Evaluate accuracy

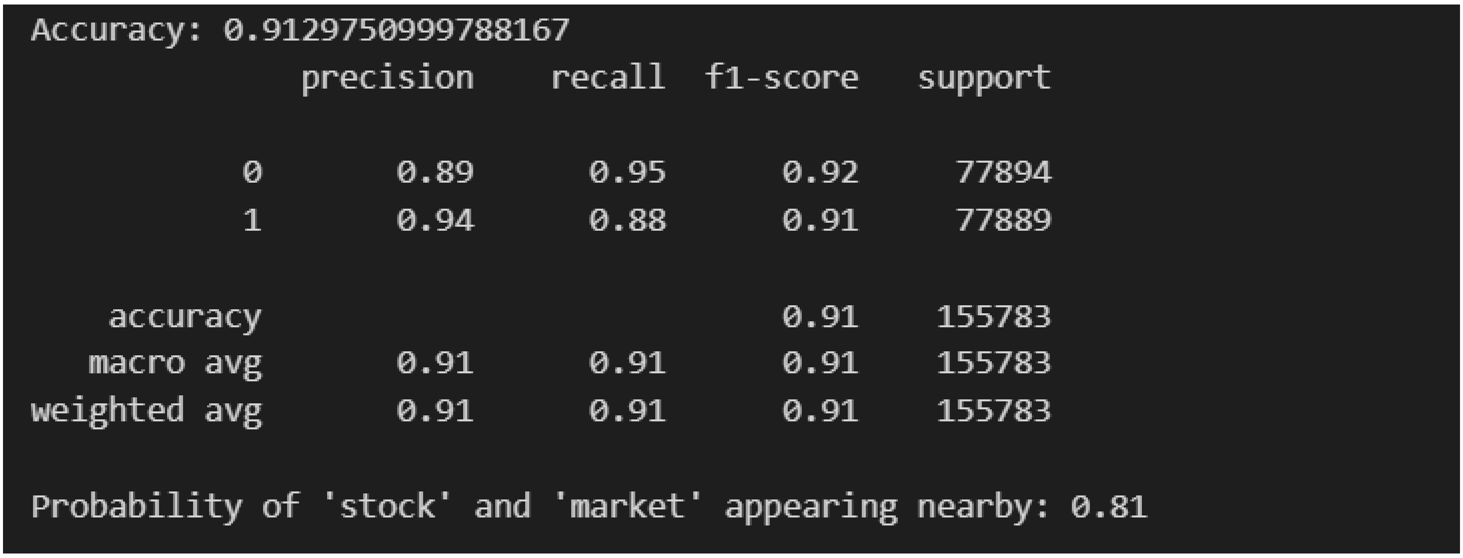
CODE AND OUTPUT



```
import nltk
import random
from nltk.corpus import brown
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, classification_report

nltk.download('brown')
nltk.download('punkt')

# Load the dataset (using Brown corpus as a placeholder for WSJ)
corpus = brown.sents(categories='news')

# Prepare the dataset
def generate_skip_gram_pairs(sentences, window_size=2):
    pairs = []
    vocabulary = set()
    for sentence in sentences:
        for idx, word in enumerate(sentence):
            word = word.lower()
            vocabulary.add(word)
            start = max(idx - window_size, 0)
            end = min(idx + window_size + 1, len(sentence))
            for neighbor in sentence[start:end]:
                if word != neighbor:
                    pairs.append((word, neighbor.lower()))
```



```
    return pairs, list(vocabulary)

# Generate positive and negative pairs
pairs, vocabulary = generate_skip_gram_pairs(corpus)

positive_pairs = pairs
negative_pairs = [(random.choice(vocabulary), random.choice(vocabulary)) for _ in
range(len(positive_pairs))]

# Combine positive and negative pairs
all_pairs = positive_pairs + negative_pairs
labels = [1] * len(positive_pairs) + [0] * len(negative_pairs)

# Split into training and test sets
train_pairs, test_pairs, y_train, y_test = train_test_split(all_pairs, labels,
test_size=0.2, random_state=42)

# Convert text pairs into feature vectors
vectorizer = CountVectorizer(analyzer=lambda x: x)
X_train = vectorizer.fit_transform(train_pairs)
X_test = vectorizer.transform(test_pairs)

# Train the logistic regression classifier
clf = LogisticRegression()
clf.fit(X_train, y_train)

# Evaluate the classifier
y_pred = clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# The classifier can now predict if two words are likely to occur nearby

def predict_proximity(word1, word2):
    pair = [(word1.lower(), word2.lower())]
    pair_vectorized = vectorizer.transform(pair)
    probability = clf.predict_proba(pair_vectorized)[0][1]
    return probability

# Example usage
probability = predict_proximity("stock", "market")
print(f"Probability of 'stock' and 'market' appearing nearby: {probability:.2f}")
```



```
Accuracy: 0.9129750999788167
              precision    recall  f1-score   support

           0       0.89      0.95      0.92     77894
           1       0.94      0.88      0.91     77889

    accuracy                           0.91    155783
   macro avg       0.91      0.91      0.91    155783
weighted avg       0.91      0.91      0.91    155783

Probability of 'stock' and 'market' appearing nearby: 0.81
```

INFERENCE

The Logistic Regression classifier successfully learns the proximity relationship between words based on Skip-Gram pairs. A higher accuracy indicates that the model effectively captures the semantic relationship between words in the corpus.