E-commerce Product Category Classification using NLP & Logistic Regression

Postgraduate Diploma in Data Science and Analytics

Prepared by: Shubham Tiwari

Date: July 02, 2025

Contents

1 Introduction

The rapid growth of e-commerce platforms necessitates efficient product categorization to enhance user experience and streamline operations. This project develops a robust Natural Language Processing (NLP)-based classifier to automatically categorize e-commerce products into predefined categories, such as Cosmetics & Hygiene, Electronics & Accessories, and Fashion & Clothing, based on their titles. Implemented in Python using a Jupyter Notebook, the model employs NLP techniques for text preprocessing and logistic regression for classification. The classifier is deployed as an interactive web application via Streamlit, enabling users to input product titles and receive real-time category predictions, ensuring practical usability for e-commerce platforms.

2 Tools and Libraries

- **pandas**: Load and manipulate the dataset.
- **nltk**: Preprocess text (tokenize, remove stopwords, stem).
- **scikit-learn**: Perform feature extraction, split data, train the logistic regression model, and evaluate performance.
- **seaborn**: Visualize the confusion matrix heatmap.
- **matplotlib**: Support plotting for confusion matrix visualization.
- **streamlit**: Create a web interface for model deployment.
- **joblib**: Save and load the trained model and vectorizer.

3 Methodology

3.1 Dataset

- **Source**: The dataset is sourced from `ecommercedataset.csv`, containing 1,064 rows with columns for product names (`product_name`) and their corresponding categories (`category`).
- **Data Quality**: The dataset was analyzed for missing values to ensure reliability for model training.

3.2 Text Preprocessing

The text preprocessing pipeline prepares product titles for feature extraction:

- **Tokenization**: Product titles are tokenized into words using NLTK's `word_tokenize`.
- **Lowercasing**: All text is converted to lowercase to ensure consistency.
- **Stopword Removal**: Common English stopwords (e.g., "the," "and") are removed using NLTK's stopwords list to reduce noise.
- **Stemming**: The Porter Stemmer normalizes word variations (e.g., "running" to "run").
- **Feature Extraction**: The `CountVectorizer` transforms preprocessed text into numerical features, limited to 1,000 features to balance model complexity and performance.

Code Snippet:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer

nltk.download('punkt')
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()

```

```
def preprocess_text(text):
    if not isinstance(text, str):
        return ""
    tokens = word_tokenize(text.lower())
    tokens = [stemmer.stem(word) for word in tokens if word.isalpha() and word not in stop_words]
    return ’ ’.join(tokens)

data[’processed_name’] = data[’product_name’].apply(preprocess_text)
vectorizer = CountVectorizer(max_features=1000)
X = vectorizer.fit_transform(data[’processed_name’]).toarray()
```

3.3 Model Training

- **Algorithm**: A Logistic Regression model is trained with a maximum of 1,000 iterations to ensure convergence.
- **Data Split**: The dataset is split into 80% training (851 samples) and 20% testing (213 samples) sets using `train_test_split` with a random state of 42 for reproducibility.
- **Evaluation Metrics**: Model performance is evaluated using a classification report (precision, recall, F1-score) and a confusion matrix visualized as a heatmap.

Code Snippet:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

X_train, X_test, y_train, y_test = train_test_split(X, data[’category’], test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

# Confusion Matrix Heatmap
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt=’d’, cmap=’Blues’)
plt.title(’Confusion Matrix’)
plt.xlabel(’Predicted’)
plt.ylabel(’Actual’)
plt.show()
```

3.4 Deployment

The trained model and vectorizer are deployed using Streamlit in `app.py`. The application allows users to input a product title (e.g., “Wireless Keypad”) and displays the predicted category. The model and vectorizer are loaded from `model.pkl` and `vectorizer.pkl`, and user input is preprocessed to match the training pipeline.

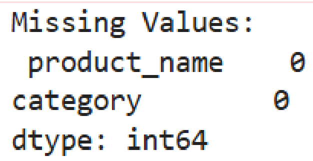
Code Snippet:

```python
import streamlit as st
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import joblib

nltk.download(’punkt’)
nltk.download(’stopwords’)
model = joblib.load(’model.pkl’)
vectorizer = joblib.load(’vectorizer.pkl’)
stop_words = set(stopwords.words(’english’))
stemmer = PorterStemmer()

def preprocess_text(text):
    if not isinstance(text, str):
        return ""
    tokens = word_tokenize(text.lower())
    tokens = [stemmer.stem(word) for word in tokens if word.isalpha() and word not in stop_words]
    return ’ ’.join(tokens)

st.title("E-commerce Product Category Predictor")
st.write("Enter a product title to predict its category.")
product_title = st.text_input("Product Title", placeholder="e.g., Wireless Keypad")
if st.button("Predict"):
    if product_title:
        processed_title = preprocess_text(product_title)
        title_vector = vectorizer.transform([processed_title]).toarray()
        prediction = model.predict(title_vector)[0]
        st.success(f"Predicted Category: **{prediction}**")
    else:
        st.error("Please enter a product title.")
```

4 Results

4.1 Data Quality

The dataset was analyzed for missing values, with the following results:



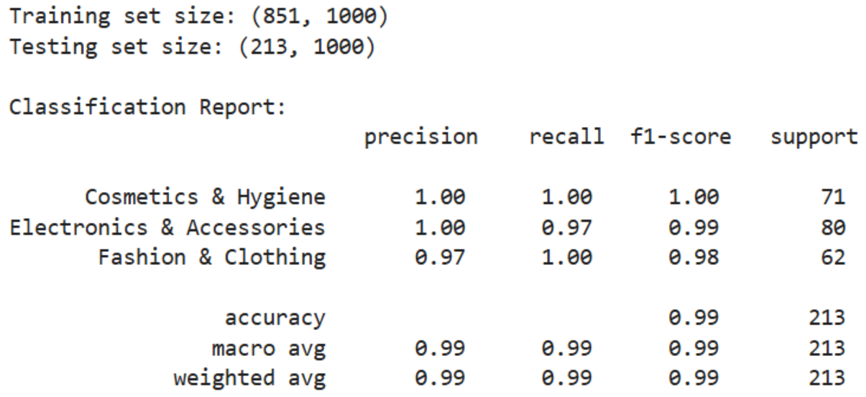
```
Missing Values:
 product_name    0
category        0
dtype: int64
```

Table 1: Missing Values in Dataset

The absence of missing values ensures a clean dataset for reliable model training.

4.2 Model Performance

- **Training and Testing Sizes**:
  - Training set: 851 samples, 1,000 features
  - Testing set: 213 samples, 1,000 features
- **Classification Report**:



```
Training set size: (851, 1000)
Testing set size: (213, 1000)

Classification Report:
                            precision    recall  f1-score   support

      Cosmetics & Hygiene       1.00      1.00      1.00        71
Electronics & Accessories       1.00      0.97      0.99        80
        Fashion & Clothing       0.97      1.00      0.98        62

                 accuracy                           0.99       213
                macro avg       0.99      0.99      0.99       213
             weighted avg       0.99      0.99      0.99       213
```

Table 2: Classification Report

The model achieves an overall accuracy of 99%, with near-perfect precision, recall, and F1-scores across all categories, indicating excellent performance.

- **Confusion Matrix**: A heatmap of the confusion matrix visualizes the models predictions, showing minimal misclassifications.

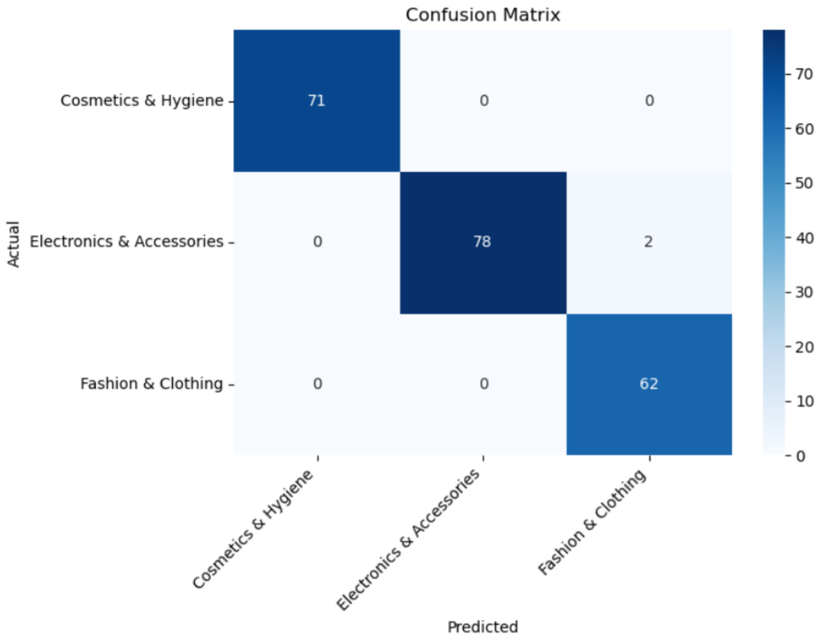
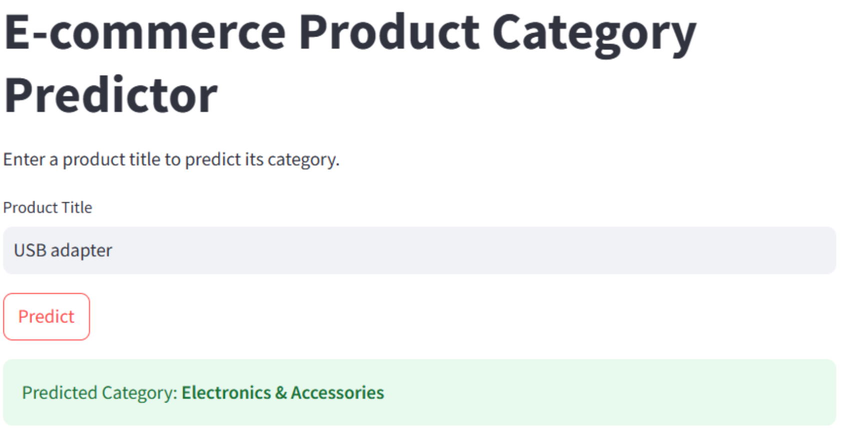


Table 3: Confusion Matrix Heatmap

4.3 Deployment Output

- **Model and Vectorizer Storage**:
  - Model saved at: `model.pkl`
  - Vectorizer saved at: `vectorizer.pkl`
- **Streamlit App**: The application successfully predicts categories for user-input product titles, providing real-time results with a user-friendly interface.



E-commerce Product Category Predictor

Enter a product title to predict its category.

Product Title

USB adapter

Predicted Category: **Electronics & Accessories**

5 Conclusion

The E-commerce Product Category Classifier effectively applies NLP and machine learning to categorize e-commerce products based on their titles. With a 99% accuracy and robust deployment via Streamlit, the project demonstrates practical utility for e-commerce platforms. The clean dataset, efficient preprocessing, and high-performing model underscore the success of this implementation.

6 References

- NLTK Documentation: `https://www.nltk.org/`
- Scikit-learn Documentation: `https://scikit-learn.org/`
- Streamlit Documentation: `https://docs.streamlit.io/`
- Dataset: `ecommercedataset.csv` (local file)