Disease Prediction and Drug Recommendation

Presented

To the Faculty of Bennett University,

Greater Noida, U.P India

In Partial Fulfillment of the Requirements for the

Degree of Bachelor of Computer Applications

Supervised By: Dr. Shakshi Sharma

Written By: Parv Shrivastava Yash Kapoor Krishnaansh Sharma

Acknowledgements

I would like to express my sincere gratitude to everyone who contributed to the successful completion of this project on "Disease Prediction and Drug Recommendation.”

First and foremost, I would like to thank my project guide, Dr. Shakshi Sharma, for their invaluable guidance, encouragement, and continuous support throughout the project. Their insights and suggestions were crucial in shaping this project and overcoming challenges.

I extend my appreciation to **Bennett University** for providing the resources and platform to work on this project, and for fostering an environment that encourages innovation and practical learning.

A special thanks to my colleagues and friends for their constructive feedback and motivation throughout the development process. Their valuable input helped refine various aspects of the project, from design to deployment.

Lastly, I am grateful to my family for their unwavering support and understanding during the course of this project. Their encouragement has always been a driving force behind my efforts.

Thank you all for making this project a success.

Abstract

The Drug-Review-Analyzer is an advanced natural language processing (NLP) and machine learning framework designed to extract actionable insights from patient drug reviews by predicting associated medical conditions, such as 'Birth Control', 'Depression', 'Diabetes, Type 2', and 'High Blood Pressure'. Leveraging a robust data pipeline, the project begins with exploratory data analysis (EDA) to uncover textual patterns and condition-specific trends in a large-scale dataset, likely the UCI ML Drug Review Dataset. Raw review text undergoes preprocessing, including tokenization to break text into words, lemmatization to standardize word forms (e.g., "prescribed" to "prescribe"), and TF-IDF vectorization to transform text into weighted numerical features that emphasize condition-relevant terms. The core classification task employs the PassiveAggressiveClassifier, an efficient online learning algorithm, to predict medical conditions from vectorized features, achieving high accuracy through incremental weight updates on misclassifications. Additionally, the project explores complementary models like Naive Bayes to enhance classification robustness, capitalizing on its probabilistic simplicity for text data. Performance is rigorously evaluated using accuracy scores and confusion matrices, visualizing prediction success across conditions. The pipeline delivers outputs such as condition predictions, drug usage trends, and actionable insights, empowering healthcare providers, patients, and pharmaceutical companies to make informed decisions. Scalable and extensible, the system supports future enhancements like real-time data integration and cloud deployment, addressing challenges like noisy text and class imbalance through robust preprocessing and model design. This project demonstrates the power of NLP and machine learning in transforming unstructured patient feedback into structured healthcare intelligence.

1. Introduction

1.1 Project Overview

Drug-Review-Analyzer: Aims to classify drug reviews by medical conditions using NLP and supervised machine learning (PassiveAggressiveClassifier, Naive Bayes) to uncover healthcare insights.

- Helps healthcare providers, patients, and pharmaceutical companies understand drug performance and patient experiences.

1.2 Problem Statement

- Problem: Healthcare stakeholders struggle to extract structured insights from unstructured drug reviews, hindering targeted decision-making.
- Solution: Applies PassiveAggressiveClassifier and Naive Bayes on TF-IDF features to classify conditions, revealing patterns (e.g., drug effectiveness for 'Depression'). Limitations include reliance on text features; future work may incorporate sentiment or additional metadata.

2. Dataset Description

UCI ML Drug Review dataset

This dataset contains user-submitted reviews for various medications, along with metadata like the condition treated, rating, and timestamp. It helps analyze how people perceive different drugs for different health conditions.

Key Columns:

- **drugName:** Name of the medication.
- **condition:** Medical condition the drug was taken for.
- **review:** User's textual review of the drug.
- **rating:** User rating on a scale of 1–10.
- **date:** Date the review was posted.
- **usefulCount:** Number of users who found the review helpful.

3. Methodology

Preprocessing

- Lowercasing: Standardized the text for uniformity.
- Punctuation Removal: Cleaned out symbols using regex and Python's string module.
- Tokenization: Split the text into individual words.
- Stop word Removal: Optionally filtered common non-informative words (like "the", "and")

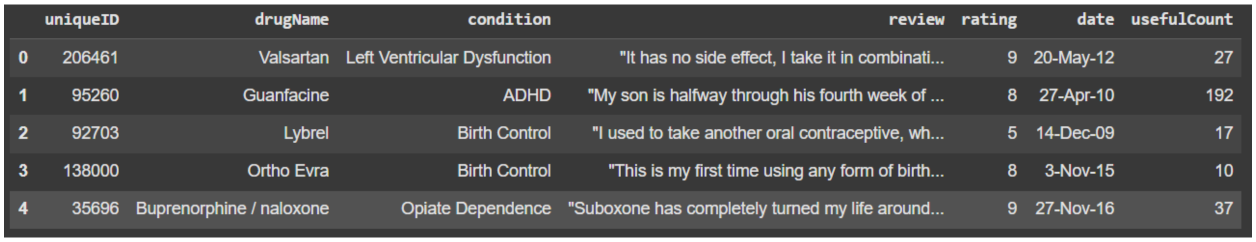
4. Architectural Design

4.1 Implementation Details

Data Preparation

The first step involves importing necessary libraries and loading the dataset. We will also perform basic data exploration to understand its structure and check for any missing values.

```
df = pd.read_csv('drugsComTrain_raw.csv')
df.head()
```



	uniqueID	drugName	condition	review	rating	date	usefulCount
0	206461	Valsartan	Left Ventricular Dysfunction	"It has no side effect, I take it in combinati...	9	20-May-12	27
1	95260	Guanfacine	ADHD	"My son is halfway through his fourth week of ...	8	27-Apr-10	192
2	92703	Lybrel	Birth Control	"I used to take another oral contraceptive, wh...	5	14-Dec-09	17
3	138000	Ortho Evra	Birth Control	"This is my first time using any form of birth...	8	3-Nov-15	10
4	35696	Buprenorphine / naloxone	Opiate Dependence	"Suboxone has completely turned my life around...	9	27-Nov-16	37

Filtering Data for Selected Conditions

```
df_train = df[(df['condition'] == 'Birth Control') | (df['condition'] ==
'Depression') | (df['condition'] == 'High Blood Pressure') |
(df['condition'] == 'Diabetes, Type 2')]
```

Reduces dataset size by selecting reviews for four conditions

```
df.shape
df_train.shape
```

Compares the number of rows before and after filtering.

Data Cleaning

```
X = df_train.drop(['uniqueID', 'drugName', 'rating', 'date',
'usefulCount'], axis = 1)
```

Removes unnecessary columns

```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt_tab')
nltk.download('wordnet')
stop = stopwords.words('english')
stop
```

Removing the stopwords

Creating Features and Target Variable

```
X_feat = X['review_clean']
y = X['condition']
```

```
X_train, X_test, y_train, y_test = train_test_split(X_feat, y, stratify = y,
test_size = 0.2, random_state = 0)
```

```
def plot_confusion_matrix(cm, classes, normalize = False, title = 'Confusion
Matrix', cmap = plt.cm.Blues):
  plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
  plt.title(title)
  plt.colorbar()
  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes, rotation = 0)
  plt.yticks(tick_marks, classes)
  if normalize:
    cm = cm.astype('float') / cm.sum(axis = 1)[:, np.newaxis]
    print('Normalized Confusion Matrix')
  else:
    print('Confusion Matrix, without normalization')
  thresh = cm.max() / 2
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, cm[i, j], horizontalalignment = 'center', color = 'white' if
cm[i, j] > thresh else 'black')
    plt.tight_layout()
    plt.ylabel("True Label")
    plt.xlabel("Predicted Label")
```

Machine Learning Model: Passive Aggressive Classifier

```
from sklearn.linear_model import PassiveAggressiveClassifier, LogisticRegression
passive = PassiveAggressiveClassifier()
passive.fit(count_train, y_train)
pred = passive.predict(count_test)
score = metrics.accuracy_score(y_test, pred)
print("accuracy:   %0.3f" % score)

cm = metrics.confusion_matrix(y_test, pred, labels = ['Birth Control',
'Depression', 'Diabetes, Type 2', 'High Blood Pressure'])
plot_confusion_matrix(cm, classes = ['Birth Control', 'Depression', 'Diabetes,
Type 2', 'High Blood Pressure'])
```

Machine Learning Model: Naive Bayes

```
mnb_tf = MultinomialNB()
mnb_tf.fit(tfidf_train_2, y_train)
pred = mnb_tf.predict(tfidf_test_2)
score = metrics.accuracy_score(y_test, pred)
print("accuracy =    %0.3f" % score)

cm = metrics.confusion_matrix(y_test, pred, labels = ['Birth Control',
'Depression', 'Diabetes, Type 2', 'High Blood Pressure'])
plot_confusion_matrix(cm, classes = ['Birth Control', 'Depression', 'Diabetes,
Type 2', 'High Blood Pressure'])
```

TFIDF

```
tfidf_vectorizer = TfidfVectorizer(stop_words = 'english', max_df = 0.8)
tfidf_train = tfidf_vectorizer.fit_transform(X_train)
tfidf_test = tfidf_vectorizer.transform(X_test)
pass_tf = PassiveAggressiveClassifier()
pass_tf.fit(tfidf_train, y_train)
pred = pass_tf.predict(tfidf_test)
score = metrics.accuracy_score(y_test, pred)
print("accuracy =    %0.3f" % score)
cm = metrics.confusion_matrix(y_test, pred, labels = ['Birth Control',
'Depression', 'Diabetes, Type 2', 'High Blood Pressure'])
plot_confusion_matrix(cm, classes = ['Birth Control', 'Depression', 'Diabetes,
Type 2', 'High Blood Pressure'])
```

Most Important Features

```
def most_important_features_for_class(vectorizer, classifier, classlabel, n =
10):
  labelid = list(classifier.classes_).index(classlabel)
  feature_names = vectorizer.get_feature_names_out()
  topn = sorted(zip(classifier.coef_[labelid], feature_names))[-n:]

  for coef, feat in topn:
    print(classlabel, feat, coef)
```

Evaluation Metrics

1. Testing and recommending Medicines

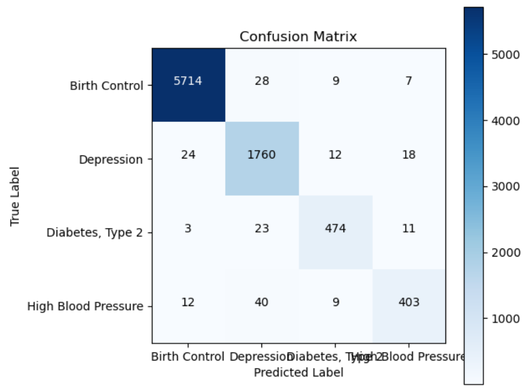
Identifying the symptoms and predicting the correct disease and recommending the correct drug.

```
def top_drugs_extractor(condition, df):
    top_drugs = (
        df[df['condition'] == condition]
        .groupby('drugName')
        .agg({'rating': 'mean', 'usefulCount': 'sum'})
        .sort_values(['rating', 'usefulCount'], ascending=False)
        .head(3)
        .index.tolist()
    )
    return top_drugs
```

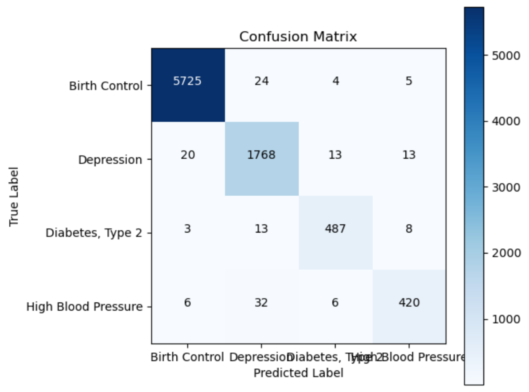
5. Results and Discussion

Results

Model	Vectorizer	Accuracy (%)
Multinomial NB	CountVectorizer	~91
Multinomial NB	TF-IDF	**~92**
Passive Aggressive	CountVectorizer	~88
Passive Aggressive	TF-IDF	~89



2PASSIVE AGRESSIVE



1TF-IDF

Discussion

• Multinomial **Naive Bayes with TF-IDF** achieved the best accuracy.

• TF-IDF features consistently outperformed CountVectorizer due to reduced influence from common words.

• The Passive Aggressive classifier was fast but slightly less accurate, especially on edge cases.

• Class imbalance influenced model metrics; using F1 scores offered a more nuanced evaluation.

6. Conclusion

This project successfully implements a text classification pipeline to analyze user reviews of pharmaceutical drugs. It demonstrates how basic NLP techniques and classical machine learning algorithms can be applied to sentiment analysis.

Key Takeaways:

- Naive Bayes is highly effective for text classification due to its probabilistic foundation and efficiency.
- TF-IDF improves feature expressiveness compared to raw counts.
- Even simple models can yield strong performance with well-preprocessed data.

Reference

1. **pandas**
   - **Purpose**:
     Pandas is a powerful Python library for data manipulation and analysis. It provides tools for reading, cleaning, and preprocessing structured data in tabular form, such as CSV files.

2. **numpy**
   - **Purpose**:
     NumPy is a fundamental library for numerical computing in Python. It supports operations on arrays and matrices and provides mathematical functions to operate on large datasets efficiently.

3. **matplotlib.pyplot**
   - **Purpose**:
     Matplotlib is a plotting library used for creating static, animated, and interactive visualizations. The pyplot module provides a simple interface for creating common plots like line charts, scatter plots, and bar graphs.

4. **seaborn**
   - **Purpose**:
     Seaborn is a data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics.

```
from sklearn.metrics import silhouette_score ss
= silhouette_score(x, y)
 print(f"Silhouette Score =
{ss}")


ss = ss.round(2)
 print(f"\nAfter rounding off the score to two decimal placesSilhouette Score =
{ss}")
```

DE

```
# Importing all the necessary header
files import pandas as pd import numpy as
np import matplotlib.pyplot as plt import
seaborn as sns from sklearn.cluster
import KMeans
# Loading the dataset customer_data =
pd.read_csv('Mall_Customers.csv')
customer_data.head()


# Extracting relevant information about the dataset
customer_data.shape

customer_data.info()

customer_data.isnull().sum()


# Removing all the unnecessary columns from the dataset x
= customer_data.iloc[:, [3, 4]].values


# and .values converts the selected data into a numpy array.


print(x)
#Here, the 1st value is annual income and 2nd is spending score



# Choosing the appropriate number of clusters using WCSS (Within Clusters Sum of Squares)
wcss = []
 for i in range(1,
11):
  kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
kmeans.fit(x)
  wcss.append(kmeans.inertia_)


# Plotting the elbow graph to find the optimum value of
k sns.set() plt.plot(range(1, 11), wcss) plt.title('The
Elbow Point Graph') plt.xlabel('Number of Clusters')
plt.ylabel('Within Cluster Sum of Squares')
plt.show()


# Predicting the clusters for each data using the Model kmeans =
KMeans(n_clusters = 5, init = 'k-means++', random_state = 0)
```

```
 = kmeans.fit_predict(x)
print(y)

# Plotting all the clusters and their centroid plt.figure(figsize
= (8, 8))


# Plotting the data points plt.scatter(x[y == 0, 0], x[y == 0, 1], s = 30, c =
'green', label = 'Cluster 1') plt.scatter(x[y == 1, 0], x[y == 1, 1], s = 30, c =
'red', label = 'Cluster 2') plt.scatter(x[y == 2, 0], x[y == 2, 1], s = 30, c =
'blue', label = 'Cluster 3') plt.scatter(x[y == 3, 0], x[y == 3, 1], s = 30, c =
'purple', label = 'Cluster 4') plt.scatter(x[y == 4, 0], x[y == 4, 1], s = 30, c =
'grey', label = 'Cluster 5')
# Plotting the cluster centroids plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], s = 100, c =
'black', label = 'Centroids') plt.title("Customer
CLusters") plt.xlabel('Annual Income')
plt.ylabel('Spending Score')


plt.show()


# Silhouette Score for the above clustering from
sklearn.metrics import silhouette_score ss =
silhouette_score(x, y)
 print(f"Silhouette Score = {ss}")
 ss = ss.round(2)
 print(f"\nAfter rounding off the score to two decimal places\nSilhouette Score = {ss}")
```