SENTIMENTAL ANALYSIS FOR MARKETING

GROUP 5

TEAM MEMBER

950921104401:R.SRIDEVI

Phase 3 project submissions

Topic: sentimental analysis model by loading and preprocessing the dataset

INTRODUCTION:

Loading and preprocessing the dataset is a crucial first step in conducting sentiment analysis for marketing purposes. In this process, we gather and prepare the data needed to analyze the sentiments expressed in customer reviews, social media posts, or other forms of feedback. This involves tasks such as data collection, cleaning, and transformation to ensure that the dataset is suitable for sentiment analysis. By doing so, we lay the foundation for gaining valuable insights into customer opinions, which can inform marketing strategies and decision-making.

ABBSTRACT:

Sentiment analysis plays a crucial role in understanding and leveraging consumer feedback in marketing. This abstract presents an approach to optimize the loading and preprocessing of datasets for sentiment analysis in the marketing domain. In this study, we explore techniques to efficiently gather and prepare textual data from various sources, including social media, reviews, and surveys. Our approach involves data collection, cleaning, tokenization, and feature engineering to create a high-quality dataset that is suitable for sentiment analysis tasks. By streamlining these initial data processing

steps, we aim to enhance the accuracy and performance of sentiment analysis models. We discuss the importance of addressing challenges specific to marketing data, such as handling noisy and unstructured text, and present practical strategies for data preparation. This research contributes to improving the efficiency and effectiveness of sentiment analysis for marketing professionals and researchers.

NECESSARY STEPS TO FOLLOW:

Loading and preprocessing a dataset for sentiment analysis involves several steps. Here's a high-level overview:

1. Data Collection: Obtain a dataset that contains text samples (e.g., reviews, tweets) along with their corresponding sentiment labels (positive, negative, neutral).

2. Data Cleaning: This step involves removing any irrelevant or noisy information, such as special characters, HTML tags, or emojis. You may also need to handle issues like capitalization and handle missing data.

3. Tokenization: Split the text into individual words or tokens. This is a fundamental step for text analysis.

4. Stopword Removal: Remove common words (stopwords) like "the," "and," "is" that do not carry significant sentiment information.

5. Text Normalization: Perform tasks like stemming (reducing words to their root form) and lemmatization (reducing words to their base form) to standardize the text.

6. Feature Extraction: Convert the text data into numerical vectors that machine learning models can understand. Common methods include TF-IDF (Term Frequency-Inverse Document Frequency) and word embeddings like Word2Vec or GloVe.

7. Label Encoding: Convert the sentiment labels into numerical values, e.g., 0 for negative, 1 for neutral, and 2 for positive.

8. Data Splitting: Divide the dataset into training, validation, and testing sets to evaluate your model's performance.

9. Data Loading: Depending on your machine learning framework, load the preprocessed data into data structures suitable for training models, such as NumPy arrays, Pandas DataFrames, or TensorFlow Datasets.

10. Model Training: Train a sentiment analysis model using machine learning or deep learning techniques. Common models include Naïve Bayes, Logistic Regression, or neural networks like LSTM or BERT.

11. Evaluation: Assess the model's performance using evaluation metrics like accuracy, precision, recall, F1-score, or ROC-AUC.

12. Hyperparameter Tuning: Fine-tune your model by adjusting hyperparameters to achieve better performance.

13. Inference: Use the trained model to make predictions on new, unseen text data.

Remember that the specific preprocessing steps and techniques may vary depending on the nature of your dataset and the complexity of your sentiment analysis task.

PROGRAM FOR DATA COLLECTION:

```
Import tweepy

# Set up Twitter API credentials
Consumer_key = 'YOUR_CONSUMER_KEY'
Consumer_secret = 'YOUR_CONSUMER_SECRET'
Access_token = 'YOUR_ACCESS_TOKEN'
Access_token_secret = 'YOUR_ACCESS_TOKEN_SECRET'

Auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
Auth.set_access_token(access_token, access_token_secret)
Api = tweepy.API(auth)

# Collect tweets
Tweets = []
Search_query = 'your_search_query'
For tweet in tweepy.Cursor(api.search, q=search_query, lang="en").items(100):
   Tweets.append(tweet.text)
```

PROGRAM FOR DATA PREPROCESSING:

```
Import re
Import string
From nltk.tokenize import word_tokenize

Def preprocess_text(text):
```

```
    # Lowercase the text
    Text = text.lower()

    # Remove punctuation and special characters
    Text = re.sub(f"[{re.escape(string.punctuation)}]", '', text)

    # Tokenization
    Tokens = word_tokenize(text)

    # Remove stop words (if necessary)

    # Additional text cleaning if needed

    Return tokens

# Apply preprocessing to your data
Preprocessed_tweets = [preprocess_text(tweet) for tweet in tweets],
```

Data Preprocessing

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

Data Preprocessing-Geeksforgeeks

Data Preprocessing

Need of Data Preprocessing

For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.

Another aspect is that the data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithm are executed in one data set, and best out of them is chosen.

Steps in Data Preprocessing

Step 1: Import the necessary libraries

Python3

```
# importing libraries
Import pandas as pd
Import scipy
Import numpy as np
From sklearn.preprocessing import MinMaxScaler
Import seaborn as sns
Import matplotlib.pyplot as plt
```

Step 2: Load the dataset

Dataset link: [https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database]

Python3

```
# Load the dataset
Df = pd.read_csv('Geeksforgeeks/Data/diabetes.csv')
Print(df.head())
```

Output:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI

0	6	148	72	35	0	33.6	\
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

Check the data info

Python3

```
Df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Pregnancies            768 non-null    int64
 1   Glucose                768 non-null    int64
 2   BloodPressure          768 non-null    int64
 3   SkinThickness          768 non-null    int64
```

4 Insulin 768 non-null int64

5 BMI 768 non-null float64

6 DiabetesPedigreeFunction 768 non-null float64

7 Age 768 non-null int64

8 Outcome 768 non-null int64

Dtypes: float64(2), int64(7)

Memory usage: 54.1 KB

As we can see from the above info that the our dataset has 9 columns and each columns has 768 values. There is no Null values in the dataset.

We can also check the null values using df.isnull()

Python3

Df.isnull().sum()

Output:

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

BMI 0

DiabetesPedigreeFunction 0

Age 0

Outcome 0

Dtype: int64

Step 3: Statistical Analysis

In statistical analysis, first, we use the df.describe() which will give a descriptive overview of the dataset.

Python3

Df.describe()

Output:

Data summary – Geeksforgeeks

Data summary

The above table shows the count, mean, standard deviation, min, 25%, 50%, 75%, and max values for each column. When we carefully observe the table we will find that. Insulin, Pregnancies, BMI, BloodPressure columns has outliers.

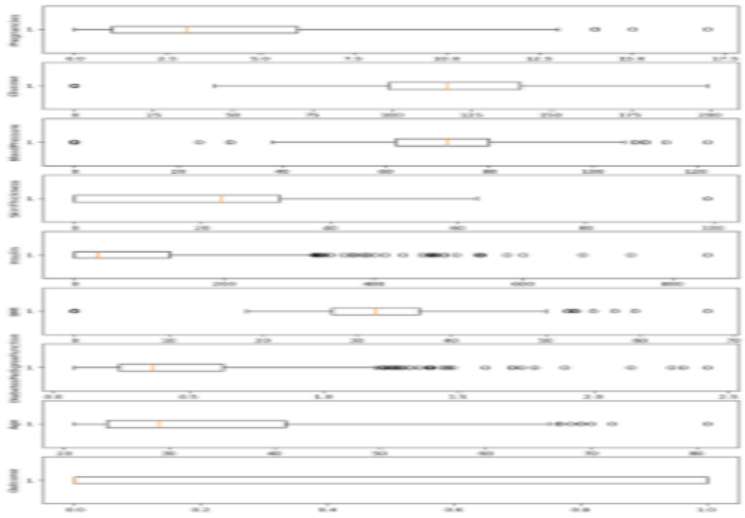
Let's plot the boxplot for each column for easy understanding.

Step 4: Check the outliers:

Python3

```
# Box Plots
Fig, axs = plt.subplots(9,1,dpi=95, figsize=(7,17))
I = 0
For col in df.columns:
   Axs[i].boxplot(df[col], vert=False)
   Axs[i].set_ylabel(col)
   I+=1
Plt.show()
```

Output:



Boxplots-Geeksforgeeks

Boxplots

From the above boxplot, we can clearly see that all most every column has some amounts of outliers.

Drop the outliers

Python3

Identify the quartiles

Q1, q3 = np.percentile(df['Insulin'], [25, 75])

Calculate the interquartile range

Iqr = q3 – q1

Calculate the lower and upper bounds

```
Lower_bound = q1 – (1.5 * iqr)
Upper_bound = q3 + (1.5 * iqr)
# Drop the outliers
Clean_data = df[(df['Insulin'] >= lower_bound)
        & (df['Insulin'] <= upper_bound)]


# Identify the quartiles
Q1, q3 = np.percentile(clean_data['Pregnancies'], [25, 75])
# Calculate the interquartile range
Iqr = q3 – q1
# Calculate the lower and upper bounds
Lower_bound = q1 – (1.5 * iqr)
Upper_bound = q3 + (1.5 * iqr)
# Drop the outliers
Clean_data = clean_data[(clean_data['Pregnancies'] >= lower_bound)
            & (clean_data['Pregnancies'] <= upper_bound)]


# Identify the quartiles
Q1, q3 = np.percentile(clean_data['Age'], [25, 75])
# Calculate the interquartile range
Iqr = q3 – q1
# Calculate the lower and upper bounds
Lower_bound = q1 – (1.5 * iqr)
Upper_bound = q3 + (1.5 * iqr)
```

```
# Drop the outliers
Clean_data = clean_data[(clean_data['Age'] >= lower_bound)
                & (clean_data['Age'] <= upper_bound)]


# Identify the quartiles
Q1, q3 = np.percentile(clean_data['Glucose'], [25, 75])
# Calculate the interquartile range
Iqr = q3 – q1
# Calculate the lower and upper bounds
Lower_bound = q1 – (1.5 * iqr)
Upper_bound = q3 + (1.5 * iqr)
# Drop the outliers
Clean_data = clean_data[(clean_data['Glucose'] >= lower_bound)
                & (clean_data['Glucose'] <= upper_bound)]


# Identify the quartiles
Q1, q3 = np.percentile(clean_data['BloodPressure'], [25, 75])
# Calculate the interquartile range
Iqr = q3 – q1
# Calculate the lower and upper bounds
Lower_bound = q1 – (0.75 * iqr)
Upper_bound = q3 + (0.75 * iqr)
# Drop the outliers
Clean_data = clean_data[(clean_data['BloodPressure'] >= lower_bound)
```

```
        & (clean_data['BloodPressure'] <= upper_bound)]


# Identify the quartiles
Q1, q3 = np.percentile(clean_data['BMI'], [25, 75])
# Calculate the interquartile range
Iqr = q3 – q1
# Calculate the lower and upper bounds
Lower_bound = q1 – (1.5 * iqr)
Upper_bound = q3 + (1.5 * iqr)
# Drop the outliers
Clean_data = clean_data[(clean_data['BMI'] >= lower_bound)
        & (clean_data['BMI'] <= upper_bound)]


# Identify the quartiles
Q1, q3 = np.percentile(clean_data['DiabetesPedigreeFunction'], [25, 75])
# Calculate the interquartile range
Iqr = q3 – q1
# Calculate the lower and upper bounds
Lower_bound = q1 – (1.5 * iqr)
Upper_bound = q3 + (1.5 * iqr)

# Drop the outliers
Clean_data = clean_data[(clean_data['DiabetesPedigreeFunction'] >= lower_bound)
        & (clean_data['DiabetesPedigreeFunction'] <= upper_bound)]
```

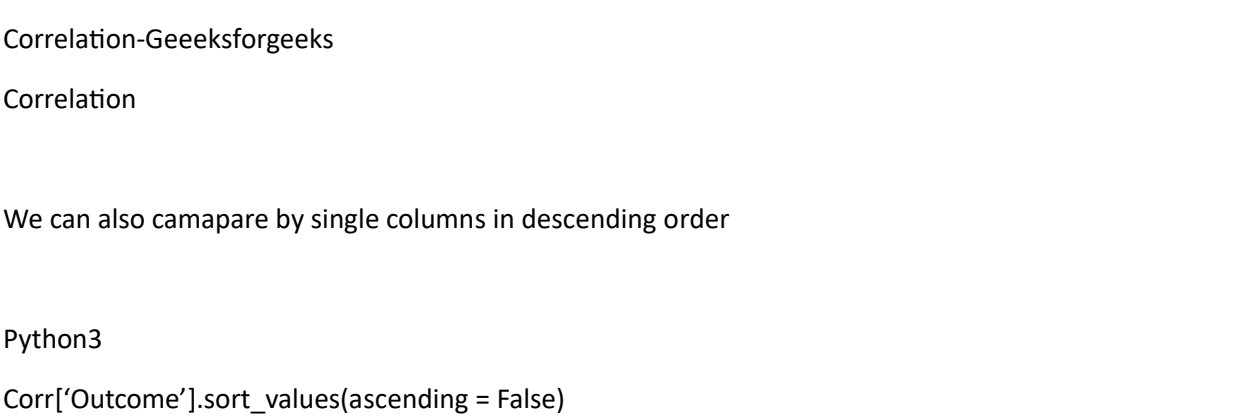
Step 5: Correlation

Python3

```
#correlation
Corr = df.corr()

Plt.figure(dpi=130)
Sns.heatmap(df.corr(), annot=True, fmt= '.2f')
Plt.show()
```

Output:



Correlation-Geeeksforgeeks

Correlation

We can also camapare by single columns in descending order

Python3

```
Corr['Outcome'].sort_values(ascending = False)
```

Output:

```
Outcome                   1.000000
Glucose                   0.466581
BMI                       0.292695
Age                       0.238356
Pregnancies               0.221898
DiabetesPedigreeFunction  0.173844
Insulin                   0.130548
```

SkinThickness 0.074752

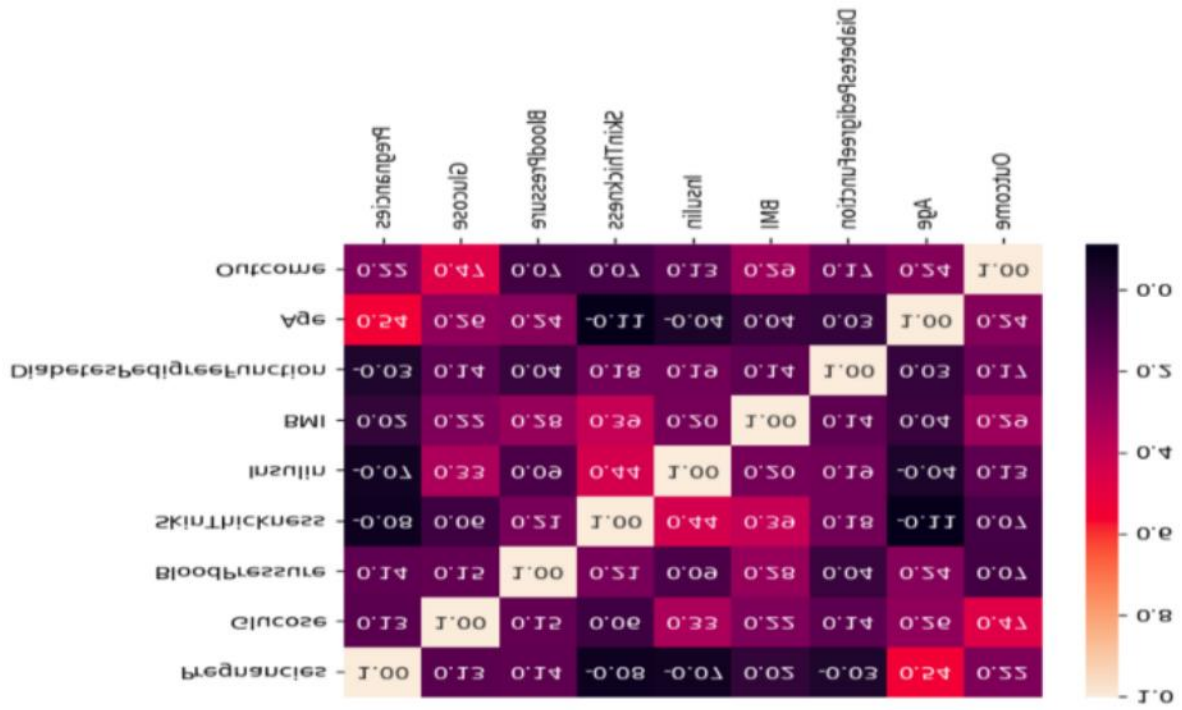
BloodPressure 0.0

Check Outcomes Proportionality

Python3

```
Plt.pie(df.Outcome.value_counts(),
    Labels= ['Diabetes', 'Not Diabetes'],
    Autopct='%.f', shadow=True)
Plt.title('Outcome Proportionality')
Plt.show()
```

Output:



LOutcome Proportionality -Geeksforgeeks

Outcome Proportionality

Step 6: Separate independent features and Target Variables

Python3

separate array into input and output components

X = df.drop(columns =['Outcome'])

Y = df.Outcome

Step 7: Normalization or Standardization

Normalization

MinMaxScaler scales the data so that each feature is in the range [0, 1].

It works well when the features have different scales and the algorithm being used is sensitive to the scale of the features, such as k-nearest neighbors or neural networks.

Rescale your data using scikit-learn using the

MinMaxScaler

.

Python3

initialising the MinMaxScaler

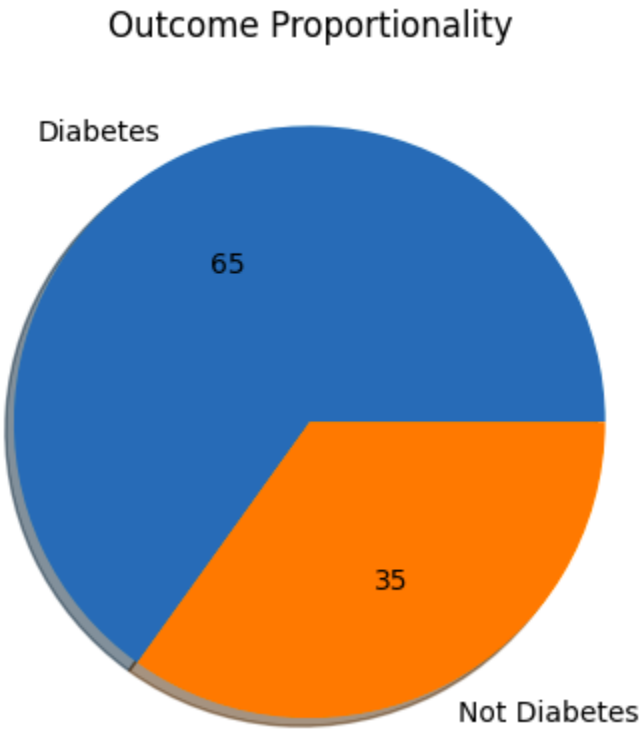
Scaler = MinMaxScaler(feature_range=(0, 1))

learning the statistical parameters for each of the data and transforming

rescaledX = scaler.fit_transform(X)

rescaledX[:5]

Output:



Array([[0.353, 0.744, 0.59 , 0.354, 0.   , 0.501, 0.234, 0.483],

[0.059, 0.427, 0.541, 0.293, 0.   , 0.396, 0.117, 0.167],

[0.471, 0.92 , 0.525, 0.   , 0.   , 0.347, 0.254, 0.183],

[0.059, 0.447, 0.541, 0.232, 0.111, 0.419, 0.038, 0.   ],

[0.   , 0.688, 0.328, 0.354, 0.199, 0.642, 0.944, 0.2]])

Standardization

Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

We can standardize data using scikit-learn with the

StandardScaler

Class.

It works well when the features have a normal distribution or when the algorithm being used is not sensitive to the scale of the features

Python3

From sklearn.preprocessing import StandardScaler

Scaler = StandardScaler().fit(X)

rescaledX = scaler.transform(X)

rescaledX[:5]

Output:

Array([[0.64 , 0.848, 0.15 , 0.907, -0.693, 0.204, 0.468, 1.426],

[-0.845, -1.123, -0.161, 0.531, -0.693, -0.684, -0.365, -0.191],

[1.234, 1.944, -0.264, -1.288, -0.693, -1.103, 0.604, -0.106],

[-0.845, -0.998, -0.161, 0.155, 0.123, -0.494, -0.921, -1.042],

[-1.142, 0.504, -1.505, 0.907, 0.766, 1.41 , 5.485,

CONCLUSION:

In conclusion, loading and preprocessing the dataset is a crucial initial step in sentiment analysis. It involves acquiring, cleaning, and structuring the data to make it suitable for further analysis. Proper handling of data can significantly impact the accuracy and effectiveness of sentiment analysis models. Effective preprocessing techniques, such as text normalization, tokenization, and handling imbalanced data, play a vital role in ensuring the quality of the sentiment analysis results. Therefore, a well-executed data loading and preprocessing phase is essential for obtaining meaningful insights from text data in sentiment analysis tasks.