SENTIMENT ANALYSIS FOR MARKETING

PHASE 4 PROJECT

BY:

THANUSRE A

SANGEETHA K

DHARSHINI T

SARUMATHY E

Sentiment Analysis Using Various ML Classifiers

In this document you will see text precessing on twitter data set and after that I have performed different Machine Learning Algorithms on the data such as Logistic Regression, RandomForestClassifier, SVC, Naive Bayes to classifiy positive and negative tweets.

Index

Introduction:

The process of building a sentiment analysis model is a criticalendeavor in the realm of real estate, finance, and property valuation.

Accurately estimating the price of a house is essential for buyers, sellers, and investors to make informed decisions. In this comprehensive guide, we will continue to delve deeper into the construction of a robust house price prediction model by focusing on three fundamental components: feature engineering, model training, and evaluation.

Feature engineering is the process of identifying and selecting the most relevant features from a dataset to improve the performance of a machine learning model. This is an important step in building a sentiment analysis model, as it can help to reduce overfitting and improve the generalization ability of the model.

Model training is the process of feeding the selected features to a machine learning algorithm and allowing it to learn the relationship between the features and the target variable (i.e., house price). Once the model is trained, it can be used to predict the house prices of new houses, given their features.

Model evaluation is the process of assessing the performance of a trained machine learning model on a held-out test set.

This is important to ensure that the model is generalizing well and that it is not overfitting the training data.

Overview of the process:

The following is an overview of the process of building a house price prediction model by feature engineering, model training, and evaluation:

1. Prepare the data: This includes cleaning the data, removing outliers, and handling missing values.
2. Perform feature engineering : This can be done using a variety of methods, such as correlation analysis, information gain, and recursive feature elimination.
3. Train the model: There are many different machine learning algorithms that can be used for house price prediction. Some popular choices include linear regression, random forests, and gradient boosting machines.
4. Evaluate the model: This can be done by calculating the mean squared error (MSE) or the root mean squared error (RMSE) of the model's predictions on the held-out test set.
5. Deploy the model: Once the model has been evaluated and found to be performing well, it can be deployed to production so that it can be used to predict the house prices of new houses.

Feature Engineering:

Now we're going to analyse the preprocessed data to get an understanding of it. We'll plot Word Clouds for Positive and Negative tweets from our dataset and see which words occur the most. Word-Cloud for Negative tweets.

In [24]:

```
plt.figure(figsize = (15,15))

wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(data[data.polarity == 0].processed_tweets))
```

plt.imshow(wc , interpolation = 'bilinear')

Out[24]:

<matplotlib.image.AxesImage at 0x7f08765ff290>

Word-Cloud for Positive tweets.

In [25]:

plt.figure(figsize = (15,15))

wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(data[data.polarity == 1].processed_tweets))

plt.imshow(wc , interpolation = 'bilinear')

Out[25]:

<matplotlib.image.AxesImage at 0x7f08765f77d0>

Vectorization and Splitting the data Storing input variable-processes_tweets to X and output variable-polarity to y

In [26]:

X = data['processed_tweets'].values

y = data['polarity'].values

In [27]:

print(X.shape)

print(y.shape)

(200000,)

(200000,) Convert text to word frequency vectors TF-IDF This is an acronym than stands for Term Frequency – Inverse Document Frequency which are the components of the resulting scores assigned to each word. Term Frequency: This summarizes how often a given word appears within a document. Inverse Document Frequency: This downscales words that appear a lot across documents.

In [28]:

#Convert a collection of raw documents to a matrix of TF-IDF features.

vector = TfidfVectorizer(sublinear_tf=True)

X = vector.fit_transform(X)

print(f'Vector fitted.')

print('No. of feature_words: ', len(vector.get_feature_names()))

Vector fitted.

No. of feature_words: 170137

In [29]:

print(X.shape)

print(y.shape)

(200000, 170137)

(200000,) Split train and test The Preprocessed Data is divided into 2 sets of data: Training Data: The dataset upon which the model would be trained on. Contains 80% data. Test Data: The dataset upon which the model would be tested against. Contains 20% data.

In [30]:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state=101)

In [31]:

print("X_train", X_train.shape)

print("y_train", y_train.shape)

print()

print("X_test", X_test.shape)

print("y_test", y_test.shape)

X_train (160000, 170137)

y_train (160000,)

X_test (40000, 170137)

y_test (40000,)

Model training:

In [13]:

%matplotlib inline

sns.countplot(data['polarity'])

Out[13]:

<AxesSubplot:xlabel='polarity', ylabel='count'>

In [14]:

Removing the unnecessary columns.

data.drop(['date','query','user','word count'], axis=1, inplace=True)

In [15]:

data.drop('id', axis=1, inplace=True)

In [16]:

data.head(10)

Out[16]:

polarity Text 722084 0 Leg hurts can hardly walk 1157616 1 @thoughtcloud Cool that'd be awesome! Thanks.... 1064123 1 @lilyroseallen sorry to keep saying 1483162 1 Woke up at 11. I feel awesome 501963 0 Sorry for lack of updates I don't have the int... 1210662 1 @ home about todo mums hair.....straightening... 1030149 1 @blindmonk ??????.. ? ??? ??? ????? ???????? ??? 390866 0 apparently its too early in the day to buy a c... 798832 0 @Tolsonii Oooooohhhhh how sad..... 702103 0 i'm sad. i'm bored and nobody wants to text me...

In [17]:

#Checking if any null values present

(data.isnull().sum() / len(data))*100

Out[17]:

polarity 0.0

text 0.0

dtype: float64

In [18]:

#convrting pandas object to a string type

data['text'] = data['text'].astype('str')

In [19]:

nltk.download('stopwords')

stopword = set(stopwords.words('english'))

print(stopword)

[nltk_data] Downloading package stopwords to /usr/share/nltk_data...

[nltk_data] Package stopwords is already up-to-date!

{'you', "that'll", 'too', 'that', 'ours', 'as', 'not', 'hadn', 'don', 'needn', 'do', 'them', 'yourself', 'because', 'once', 'more', 'aren', 'by', 'both', 'how', 'mustn', 'under', 'having', 'those', 'which', "mightn't", 'against', 'didn', 'will', 'won', 've', 'are', 'did', 't', 'above', 'haven', 'few', 'such', 'weren', 'ma', 'some', 'each', "needn't", 'll', 'our', 'at', 'in', 'shan', "mustn't", 'their', 'what', 'm', 'yourselves', 'mightn', 'just', 'down', 'further', "couldn't", 'again', 'there', 'had', 'during', 'where', 'own', 'over', 'y', "wouldn't", 'whom', 'if', 'him', 'to', 'same', 'yours', 'up', "you'd", "aren't", "hasn't", 'itself', "shouldn't", 'wouldn', 'the', 'has', 'other', "she's", 'should', 'from', 'your', 'no', 'doesn', 'a', "you'll", 'his', 'an', 'can', "it's", 'who', "haven't", 'about', 'me', 'am', 'doing', 'below', 's', 'does', 'herself', 'hasn', 'when', 'or', 're', 'myself', 'her', 'all', 'nor', "weren't", "doesn't", 'now', "won't", 'ourselves', "you're", 'most', 'but', 'very', 'she', 'have', "you've", 'been', 'shouldn', 'isn', 'they', 'for', 'on', 'were', 'wasn', 'these', "shan't", 'off', 'themselves', 'only', 'was', 'couldn', "hadn't", 'and', 'between', 'himself', 'any', "should've", 'my', "didn't", 'theirs', 'of', 'here', 'its', 'o', 'while', 'be', 'it', 'before', 'until', "isn't", 'i', 'he', 'hers', 'being', 'then', 'than', "don't", "wasn't", 'so', 'we', 'after', 'why', 'ain', 'into', 'through', 'out', 'this', 'd', 'with', 'is'}

In [20]:

nltk.download('punkt')

nltk.download('wordnet')

[nltk_data] Downloading package punkt to /usr/share/nltk_data...

[nltk_data] Package punkt is already up-to-date!

[nltk_data] Downloading package wordnet to /usr/share/nltk_data...

[nltk_data] Package wordnet is already up-to-date!

Out[20]:

True The Preprocessing steps taken are: „h Lower Casing: Each text is converted to lowercase. „h Removing URLs: Links starting with "http" or "https" or "www" are replaced by "". „h Removing Usernames: Replace @Usernames with word "". (eg: "@XYZ" to "") „h Removing Short Words: Words with length less than 2 are removed. „h Removing Stopwords: Stopwords are the English words which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. (eg: "the", "he", "have") „h Lemmatizing: Lemmatization is the process of converting a word to its base form. (e.g: ¡§wolves¡¨ to ¡§wolf¡¨)

Model Evaluation:

Model evaluating function

In [32]:

```
def model_Evaluate(model):

#accuracy of model on training data

acc_train=model.score(X_train, y_train)

#accuracy of model on test data

acc_test=model.score(X_test, y_test)

print('Accuracy of model on training data : {}'.format(acc_train*100))

print('Accuracy of model on testing data : {} \n'.format(acc_test*100))

# Predict values for Test dataset

y_pred = model.predict(X_test)

# Print the evaluation metrics for the dataset.

print(classification_report(y_test, y_pred))

# Compute and plot the Confusion matrix

cf_matrix = confusion_matrix(y_test, y_pred)
```

```
categories = ['Negative','Positive']

group_names = ['True Neg','False Pos', 'False Neg','True Pos']

group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten() /
np.sum(cf_matrix)]

labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]

labels = np.asarray(labels).reshape(2,2)

sns.heatmap(cf_matrix, annot = labels, cmap = 'Reds',fmt = '',

xticklabels = categories, yticklabels = categories)

plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)

plt.ylabel("Actual values" , fontdict = {'size':14}, labelpad = 10)

plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

Logistic Regression

In [33]:

```
lg = LogisticRegression()

history=lg.fit(X_train, y_train)

model_Evaluate(lg)
```

Accuracy of model on training data : 83.460625

Accuracy of model on testing data : 77.065

Accuracy of model on testing data : 77.065

precision

recall

f1-score

support

0

0.78

0.75

0.77

19967

1

0.76

0.79

0.78

20033

accuracy

0.77

40000

macro avg

0.77

0.77

0.77

40000

weighted avg

0.77

0.77

0.77

40000

Linear SVM

In [34]:

svm = LinearSVC()

svm.fit(X_train, y_train)

model_Evaluate(svm)

Accuracy of model on training data : 93.191875

Accuracy of model on testing data : 76.0025

precision

recall

f1-score

support

0

0.77

0.74

0.76

19967

0.75

0.78

0.76

20033

accuracy

0.76

40000

macro avg

0.76

0.76

0.76

40000

weighted avg

0.76

0.76

0.76

40000

Random Forest

In [35]:

```
rf = RandomForestClassifier(n_estimators = 20, criterion = 'entropy',
max_depth=50)
```

```
rf.fit(X_train, y_train)
```

```
model_Evaluate(rf)
```

Accuracy of model on training data : 74.87125

Accuracy of model on testing data : 70.1975

precision

recall

f1-score

support

0

0.74

0.62

0.68

19967

1

0.67

0.78

0.72

20033

accuracy

0.70

40000

macro avg

0.71

0.70

0.70

40000

weighted avg

0.71

0.70

0.70

40000

Naive Bayes

In [36]:

nb = BernoulliNB()

nb.fit(X_train, y_train)

model_Evaluate(nb)

Accuracy of model on training data : 86.86874999999999

Accuracy of model on testing data : 75.69749999999999

precision

recall

f1-score

support

0

0.75

0.78

0.76

19967

1

0.77

0.73

0.75

20033

accuracy

0.76

40000

macro avg

0.76

0.76

0.76

40000

weighted avg

0.76

0.76

0.76

40000

Project Conclusion: Sentiment

Project Conclusion: Sentiment Analysis of Twitter Data

In this project, our team embarked on a journey to perform sentiment analysis on a Twitter dataset. Our primary goal was to understand and implement text preprocessing techniques and apply various machine learning algorithms to classify tweets as positive or negative sentiment.

Feature Engineering: We considered techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) to convert text data into numerical features, which are suitable for machine learning algorithms. This enabled us to represent text data quantitatively.

Model training: With our preprocessed dataset in hand, we proceeded to implement various machine learning algorithms for sentiment classification. We experimented with the following algorithms:

Model Evaluation: After implementing these algorithms, we assessed their performance through various evaluation metrics, such as accuracy, precision, recall, F1-score, and ROC curves. This evaluation process allowed us to compare the algorithms and determine their effectiveness in sentiment analysis.

Conclusion:

In conclusion, this project was an insightful exploration of sentiment analysis on Twitter data. Our team successfully executed text preprocessing, a critical step in preparing unstructured text data for machine learning. We implemented a range of machine learning algorithms, each with its unique characteristics, to classify tweets as positive or negative.

Through rigorous performance evaluation, we identified the strengths and weaknesses of each algorithm in handling this specific sentiment classification task. This knowledge will be invaluable for future projects and real-world applications where understanding customer sentiment is crucial. Sentiment analysis has a wide range of applications, from customer feedback analysis to brand reputation management and social media monitoring, and our project marks a significant step in harnessing the power of machine learning in these areas.

As we continue to refine our models and explore advanced techniques, we are excited about the potential of sentiment analysis and its impact on understanding public sentiment and customer experiences. This project serves as a solid foundation for future endeavors in the realm of natural language processing and sentiment analysis.