VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

REPORT

Social Network Analytics

Name: Aditya Panditrao

RegistrationNo: 22MCB0032

Course Name: Social Network Analytics

Course Code: MCSE618L

TABLE OF CONTENTS

INTRODUCTION

Sentiment analysis, also known as opinion mining, is a subfield of natural language processing (NLP) that involves the extraction and analysis of subjective information from text data. It aims to determine the sentiment or emotional tone behind a piece of text, whether it is positive, negative, or neutral.

The primary goal of sentiment analysis is to understand and interpret the attitudes, opinions, and emotions expressed in textual data. It involves various techniques and approaches, including machine learning, deep learning, and linguistic analysis, to automatically classify and quantify the sentiment expressed in text.

Sentiment analysis has wide-ranging applications in various domains. Some common use cases include:

1. **Social media monitoring**: Analyzing user-generated content on platforms like Twitter, Facebook, and Instagram to gauge public opinion, brand sentiment, or customer feedback.

2. **Customer feedback analysis**: Analyzing customer reviews, surveys, or feedback forms to understand customer satisfaction, identify issues, and improve products or services.

3. **Brand reputation management**: Monitoring online discussions and news articles to assess the overall sentiment towards a brand or product and take appropriate actions to maintain a positive reputation.

4. **Market research:** Analyzing textual data from customer surveys, forums, or product reviews to gain insights into consumer preferences, market trends, and competitor analysis.

Sentiment analysis can be performed using various techniques, including rule-based approaches, machine learning algorithms (**such as Naive Bayes, Support Vector Machines, or Random Forests**), and deep learning models (**such as Recurrent Neural Networks or Transformer-based models like BERT**).

Overall, sentiment analysis enables organizations to gain valuable insights from large volumes of textual data, understand customer sentiments, make data-driven decisions, and enhance customer experience and satisfaction.

NAIVE BAYES ALGORITHM

The Naïve Bayes classifier algorithm is a popular algorithm used for classification tasks, especially in text classification. It is based on Bayes' theorem and assumes that all features are independent of each other.

Here is an outline of the steps involved in implementing the Naïve Bayes classifier algorithm:

1. **Data Preprocessing**: Preprocess your dataset, which may include steps like cleaning, tokenization, removing stopwords, and converting text into numerical representations such as bag-of-words or TF-IDF vectors.

2. **Split the Dataset**: Split your dataset into a training set and a test set. The training set will be used to train the Naïve Bayes classifier, and the test set will be used to evaluate its performance.

3. **Calculate Class Priors**: Calculate the prior probabilities of each class in the training set. This can be done by counting the occurrences of each class in the training data and dividing it by the total number of training instances.

4. **Calculate Feature Likelihoods**: For each feature (or word in the case of text classification), calculate the likelihood of that feature occurring given each class. This involves counting the occurrences of each feature in the instances of each class.

5. **Make Predictions:** For each instance in the test set, calculate the posterior probability of each class using Bayes' theorem and the likelihoods and priors calculated in the previous steps. The class with the highest posterior probability is assigned as the predicted class for that instance.

6. **Evaluate the Model:** Compare the predicted classes with the actual classes in the test set to evaluate the performance of the Naïve Bayes classifier. Common evaluation metrics for classification include accuracy, precision, recall, and F1 score.

The implementation details may vary depending on the specific programming language or library you are using. Python provides several libraries, such as scikit-learn, that offer ready-to-use implementations of the Naïve Bayes classifier algorithm.

SUPPORT VECTOR MACHINE (SVM)

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It is particularly effective in dealing with complex datasets and has been widely used in various domains, including text classification, image recognition, and bioinformatics.

The key idea behind SVM is to find an optimal hyperplane that separates the data points of different classes with the largest possible margin. In the case of linearly separable data, SVM aims to find the best decision boundary that maximizes the distance between the support vectors (data points closest to the decision boundary) of different classes.

Here are some key characteristics and concepts related to SVM:

1. Hyperplane: In SVM, a hyperplane refers to the decision boundary that separates the data points of different classes. For a binary classification problem, the hyperplane is a line in a 2D space, a plane in a 3D space, or a higher-dimensional surface in multidimensional spaces.

2. Support Vectors: Support vectors are the data points that lie closest to the decision boundary. These points play a crucial role in defining the optimal hyperplane. SVM constructs the decision boundary using these support vectors.

3. Kernel Trick: SVM can handle nonlinear classification tasks by mapping the input data into a higher-dimensional feature space using a kernel function. The kernel function computes the similarity between pairs of data points in the original space or the transformed feature space. It allows SVM to find nonlinear decision boundaries in the higher-dimensional space.

4. Regularization Parameter (C): SVM has a regularization parameter (C) that controls the trade-off between maximizing the margin and minimizing the classification errors. A smaller value of C allows for a larger margin but may tolerate more misclassifications, while a larger value of C aims to classify more points correctly at the expense of a narrower margin.

5. Soft Margin SVM: In cases where the data is not linearly separable, SVM allows for a soft margin by introducing a slack variable. This allows for some misclassifications but still aims to minimize the errors and maximize the margin.

SVM has several advantages, including its ability to handle high-dimensional data, effectiveness in handling small to medium-sized datasets, and good generalization performance. However, it can be computationally intensive for large datasets.

Python provides several libraries, such as scikit-learn, that offer SVM implementations with different kernels (linear, polynomial, radial basis function, etc.) and options for binary and multiclass classification as well as regression tasks. Overall, SVM is a powerful algorithm with a solid theoretical foundation that has proven to be effective in various machine learning tasks, especially in situations where data separation is not straightforward or when dealing with complex decision boundaries.

DECISION TREE

A Decision Tree is a supervised machine learning algorithm used for both classification and regression tasks. It is a flowchart-like model where each internal node represents a feature, each branch represents a decision, and each leaf node represents an outcome or prediction. It is called a "tree" because of its hierarchical structure, with branches representing different decisions and outcomes.

Key characteristics and concepts related to Decision Trees include:

1. Splitting Criteria: Decision Trees make decisions based on splitting criteria that measure the quality of a split. The most commonly used splitting criteria include Gini impurity and information gain. These criteria help determine the best feature and value to split the data at each node.

2. Attribute Selection: The decision tree algorithm selects the most informative attribute to split the data based on the splitting criteria. The goal is to find the attribute that maximizes the separation between different classes or minimizes the impurity within each split.

3. Pruning: Decision Trees are prone to overfitting, where they create complex trees that fit the training data too well but generalize poorly to new data. Pruning techniques such as cost complexity pruning (also known as minimal cost complexity pruning or alpha pruning) are used to prevent overfitting by trimming or merging branches of the tree.

4. Handling Categorical and Numerical Data: Decision Trees can handle both categorical and numerical features. For categorical features, each branch represents a different category, while for numerical features, different threshold values are used to split the data.

5. Interpretability: Decision Trees provide interpretable models that are easy to understand and visualize. The decision rules learned by the tree can be easily explained and used for inference.

6. Ensemble Methods: Decision Trees can be combined using ensemble methods such as Random Forests and Gradient Boosting. Ensemble methods improve the performance and robustness of Decision Trees by training multiple trees and combining their predictions.

Python provides several libraries, such as scikit-learn, that offer implementations of Decision Trees and related algorithms. These libraries provide various options for customizing the tree-building process, handling missing values, dealing with imbalanced datasets, and visualizing the resulting trees.

Decision Trees are widely used in various domains, including finance, healthcare, customer segmentation, and fraud detection. They are effective when the data has non-linear relationships and when interpretability and explain ability are important.

Overall, Decision Trees are powerful and versatile machine learning algorithms that offer interpretable models and can handle a wide range of tasks, making them a popular choice in the field of machine learning.

RANDOM FOREST

Random Forest is an ensemble learning method that combines multiple decision trees to create a more robust and accurate model. It is a supervised machine learning algorithm used for both classification and regression tasks.

Key characteristics and concepts related to Random Forest include:

1. Ensemble of Decision Trees: Random Forest builds an ensemble of decision trees, where each tree is trained on a random subset of the training data. Each tree in the ensemble makes an independent prediction, and the final prediction is determined by aggregating the predictions of all the trees.

2. Random Feature Selection: In addition to using random subsets of the training data, Random Forest also performs random feature selection at each split of the decision trees. This helps to introduce more diversity among the trees and reduce overfitting.

3. Bagging: Random Forest uses a technique called bagging (bootstrap aggregating) to create different subsets of the training data. Bagging involves randomly sampling the training data with replacement, which means that some instances may be repeated in the subsets while others may be left out. This helps to reduce the variance and improve the generalization of the model.

Voting for Classification and Averaging for Regression: In classification tasks, Random Forest uses majority voting among the individual trees to determine the final class label. For regression tasks, the predictions of all the trees are averaged to obtain the final regression value.

4. Out-of-Bag (OOB) Error: Random Forest uses the out-of-bag error estimate to evaluate the performance of the model during training. The out-of-bag error is calculated by evaluating the performance of each tree on the instances that were not included in its training subset. This provides a useful estimate of the model's performance without the need for a separate validation set.

5. Feature Importance: Random Forest can provide a measure of feature importance, which indicates the relative importance of each feature in making accurate predictions. Feature importance is calculated based on how much the accuracy of the model decreases when a particular feature is randomly permuted.

Random Forests are known for their high accuracy, robustness to noise and outliers, and resistance to overfitting. They are widely used in various domains, including finance, healthcare, image classification, and natural language processing. Random Forests also offer advantages such as handling high-dimensional data, handling both categorical and numerical features, and providing interpretable feature importance.

Overall, Random Forest is a powerful and versatile algorithm that combines the strength of multiple decision trees to create a more accurate and reliable predictive model.

BI-LSTM (Word2Vec)

BI-LSTM (Bidirectional Long Short-Term Memory) with Word2Vec is a deep learning model used for natural language processing (NLP) tasks, such as text classification, sentiment analysis, named entity recognition, and machine translation. It combines the power of bidirectional LSTM and Word2Vec word embedding's to capture contextual information and semantic relationships in text data.

Here is an overview of the components and working of the BI-LSTM (Word2Vec) model:

1. Word Embedding's (Word2Vec): Word2Vec is a popular word embedding technique that represents words as dense vectors in a continuous vector space. It captures the semantic meaning and relationships between words based on their co-occurrence patterns in a large corpus of text. Word2Vec provides a distributed representation of words, where similar words have similar vector representations.

2. Bidirectional LSTM (BI-LSTM): LSTM is a type of recurrent neural network (RNN) that can capture long-term dependencies and handle sequential data. A standard LSTM processes the input sequence in a forward direction, but a bidirectional LSTM processes the input sequence in both forward and backward directions simultaneously. This allows the model to capture both past and future context information for each word in the sequence.

3. Model Architecture: The BI-LSTM (Word2Vec) model typically consists of an embedding layer, a bidirectional LSTM layer, and one or more fully connected layers. The embedding layer maps each word in the input sequence to its corresponding Word2Vec vector representation. The bidirectional LSTM layer processes the embedded input sequence and produces a contextual representation for each word based on both the forward and backward contexts. The fully connected layers are used for classification or other NLP tasks, where the final output is generated.

4. Training: The BI-LSTM (Word2Vec) model is trained using labeled data, where the input sequences are paired with corresponding labels or targets. During training, the model learns to adjust its parameters to minimize a loss function, such as cross-entropy, which measures the dissimilarity between predicted and true labels. The optimization process involves backpropagation and gradient descent to update the model's parameters iteratively.

5. Inference: After training, the BI-LSTM (Word2Vec) model can be used for inference on new, unseen data. Given a new input sequence, the model processes it through the embedding and bidirectional LSTM layers, and then generates predictions or outputs based on the task at hand. For example, in text classification, the model can predict the class or category of a given text.

BI-LSTM with Word2Vec is a powerful combination for NLP tasks, as it leverages the strengths of both word embedding's and bidirectional LSTM to capture the context and meaning of words in a text sequence. This enables the model to achieve better performance and understanding of textual data. Implementation of BI-LSTM (Word2Vec) can be done using deep learning frameworks such as Tensor Flow or Porch, which provide the necessary tools and libraries for building and training such models.

BERT BASED MODEL

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art language model that utilizes a transformer architecture to capture contextual relationships and representations of words in a text. It has revolutionized various natural language processing (NLP) tasks, including text classification, named entity recognition, sentiment analysis, question answering, and more.

Here is an overview of the BERT-based model:

1. Pre-training: BERT is initially pre-trained on a large corpus of unlabeled text data, such as Wikipedia articles or books. During pre-training, BERT learns to predict missing words in a sentence using a technique called masked language modeling. It also learns to determine the relationship between two sentences using a task known as next sentence prediction. This pre-training process allows BERT to capture a deep understanding of the language and context.

2. Fine-tuning: After pre-training, BERT can be fine-tuned on specific downstream NLP tasks using labeled data. The model's parameters are further trained on task-specific datasets, enabling it to adapt to specific tasks such as sentiment analysis or text classification. Fine-tuning involves updating the model's weights while keeping the initial knowledge gained during pre-training.

3. Model Architecture: BERT consists of a multi-layer bidirectional transformer encoder. The transformer architecture allows BERT to capture dependencies between words in both directions, enabling it to understand the context of a word based on its surrounding words. BERT utilizes a stack of transformer layers to extract hierarchical representations of words, sentences, and documents.

4. Tokenization: BERT tokenizes input text into sub word units using a technique called Word Piece tokenization. This enables BERT to handle out-of-vocabulary words by breaking them into smaller sub word units. Each token is associated with an embedding vector that represents its meaning and context.

5. Attention Mechanism: BERT employs self-attention mechanisms within the transformer layers to focus on important words and relationships within a sentence. Self-attention allows BERT to weigh the relevance of different words when capturing their representations and contextual dependencies.

6. Training and Inference: During training, BERT minimizes a loss function specific to the downstream task, such as cross-entropy loss for classification. The model is optimized using backpropagation and gradient descent. After training, BERT can be used for inference by inputting a text sequence and obtaining the corresponding embedding’s or predictions based on the specific task.

BERT-based models have achieved state-of-the-art performance on various NLP benchmarks and tasks. Implementation of BERT can be done using popular deep learning frameworks such as Tensor Flow or Porch, which provide pre-trained BERT models and tools for fine-tuning them on specific tasks. The Hugging Face library is also commonly used for BERT-based models, as it provides pre-trained BERT models and a user-friendly API for working with BERT in Python.

DATASET USED

Dataset contained 34000 reviews which was splitted in 20 percent for test set and 80 percent for training the model. It consist of following attributes.

1. id: The unique identifier for each entry in the dataset.

2. name: The name of the product.

3. asins: The unique identifiers assigned to the products by Amazon Standard Identification Number (ASIN).

4. brand: The brand of the product.

5. categories: The categories or product classifications that the product belongs to.

6. keys: Keywords or additional information related to the product.

7. manufacturer: The manufacturer or company that produces the product.

8. reviews.date: The date when the review was posted.

9. reviews.dateAdded: The date when the review was added to the dataset.

10. reviews.dateSeen: The date range during which the review was seen.

11. reviews.didPurchase: Indicates whether the reviewer purchased the product or not.

12. reviews.doRecommend: Indicates whether the reviewer recommends the product or not.

13. reviews.id: The unique identifier for each review.

14. reviews.numHelpful: The number of users who found the review helpful.

15. reviews.rating: The rating given by the reviewer to the product.

16. reviews.sourceURLs: The URL of the source from where the review was obtained.

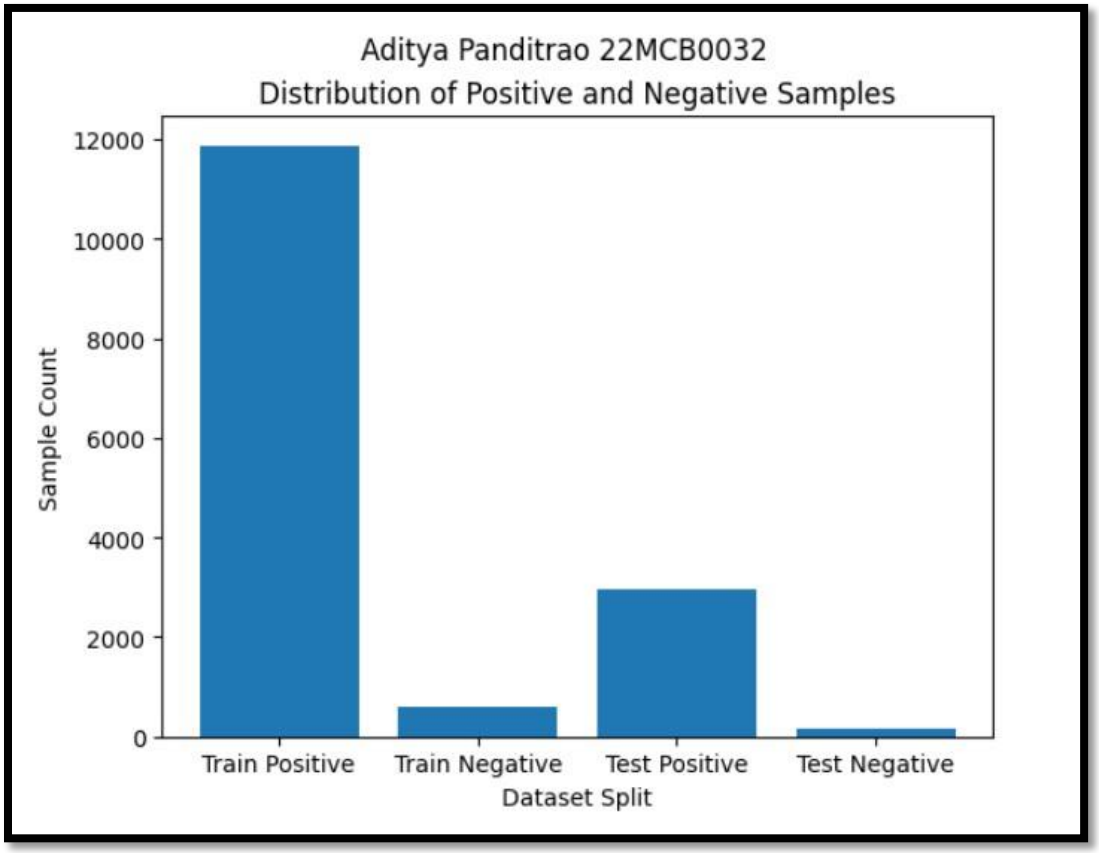
17. reviews.text: The text of the review.

18. reviews.title: The title or summary of the review.

19. reviews.userCity: The city of the reviewer.

20. reviews.userProvince: The province or state of the reviewer.

21. reviews.username: The username or identifier of the reviewer.



OUTCOMES

1) NAIVE BAYES

```
Result on training set :
Confusion matrix :
[[  411   175]
 [  364 11511]]
accuracy :  0.9567450445389616
Result on test set :
Confusion matrix :
[[  93   54]
 [ 109 2860]]
accuracy 0.9476893453145058
```

2) SVM

	precision	recall	f1-score	support
1.0	0.00	0.00	0.00	80
2.0	1.00	0.01	0.03	77
3.0	0.41	0.03	0.05	278
4.0	0.54	0.21	0.31	1756
5.0	0.74	0.97	0.84	4735
accuracy			0.72	6926
macro avg	0.54	0.24	0.24	6926
weighted avg	0.67	0.72	0.65	6926

3) BI-LSTM

Rating	Precision	Recall	F1-Score	Support
1	0.48	0.55	0.51	80
2	0.25	0.32	0.28	77
3	0.33	0.28	0.30	278
4	0.44	0.41	0.43	1756
5	0.82	0.84	0.83	4735

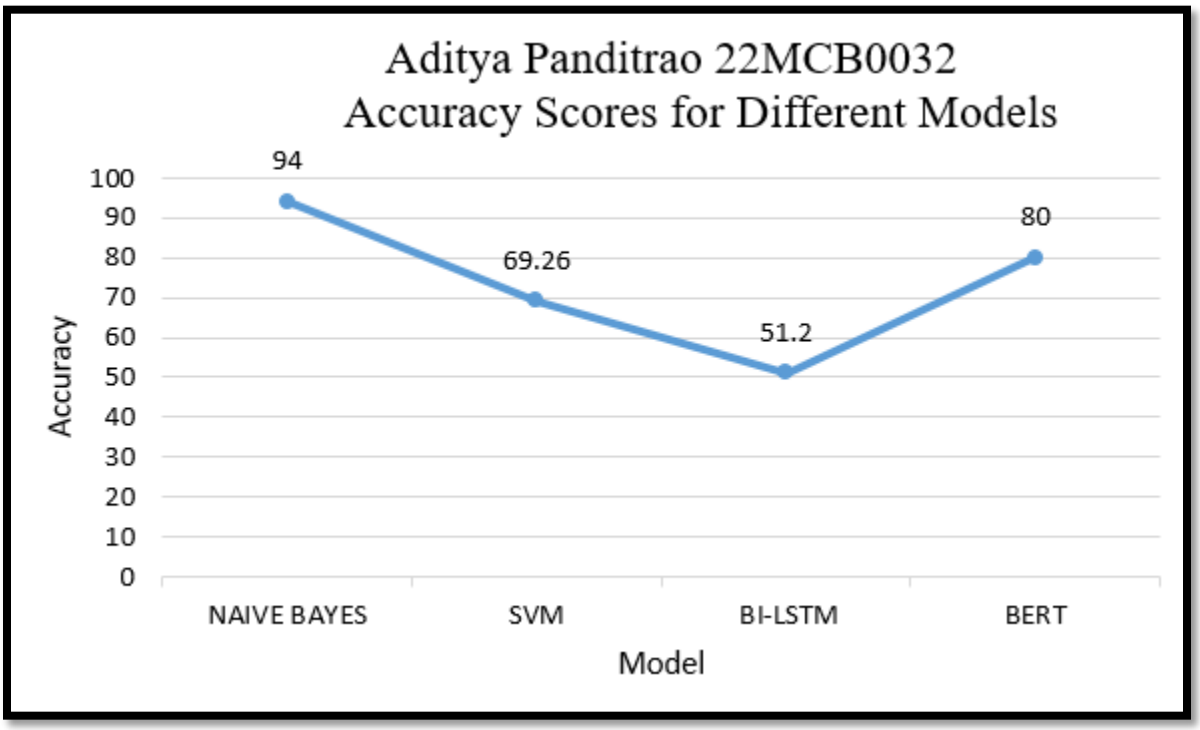
4) BERT BASED MODEL

Rating	Precision	Recall	F1-Score	Support
1	0.52	0.49	0.50	80
2	0.47	0.36	0.41	77
3	0.39	0.23	0.29	278
4	0.49	0.37	0.42	1756
5	0.80	0.92	0.86	4735

COMPARATIVE RESULT AND ANALYSIS

In our comparative analysis, we evaluated several models for sentiment analysis: Naive Bayes, SVM, BI-LSTM with Word2Vec, and a Transformer-based model with BERT-based word embedding.

Naive Bayes achieved an accuracy of 94% for SVM it is 69.26% and BI-LSTM gives accuracy of 51.2% and Bert based model gives accuracy of 80%.


Aditya Panditrao 22MCB0032
Accuracy Scores for Different Models
94
69.26
51.2
80
100
90
80
70
60
50
40
30
20
10
0
Accuracy
NAIVE BAYES
SVM
BI-LSTM
BERT
Model