UTD- Fall 2023
CS 6375 Machine Learning
Project Report
(PPA220001)

Project Title: Sarcasm Detection

Abstract:
Sarcasm is a complex linguistic phenomenon that often involves conveying the opposite of the literal meaning through irony, humour, or satire. Accurate sarcasm detection is essential in various applications, including natural language processing, sentiment analysis, customer support, social media monitoring, and more. This project aims to develop a robust sarcasm detection system leveraging state-of-the-art machine learning and natural language processing techniques.
In this report, we present a comparative study of four machine learning models – Support Vector Machines (SVM), Naive Bayes, Decision Tree, and Neural Networks – for sarcasm detection. We evaluate and compare the performance of these models based on accuracy, precision, recall, and F1 score metrics using a common dataset.

1. Introduction:
Sarcasm is a form of verbal irony that conveys the opposite of its literal meaning, often with humorous or mocking intent. Detecting sarcasm in text is crucial for improving the accuracy of sentiment analysis and understanding user sentiments on social media platforms. This project aims to explore and compare the effectiveness of different machine-learning models in identifying sarcastic expressions.

2. Problem Statement:
Given an article headline detect whether it's sarcastic using machine learning classification models like Naïve Bayes, Neural Networks, and Natural Language Processing Techniques.

3. Data Description:
The dataset used consists of a list of news headlines.
Features of Dataset:

1. Headline: Headline of the news
2. Article_link: Link to the news article
3. Is_Sarcastic: 1 if the headline is sarcastic or 0 otherwise.

Data Size: 3 MB
Dataset link: [Kaggle Dataset] [1]

4. Methodology:

The methodology used to implement this project is as follows:

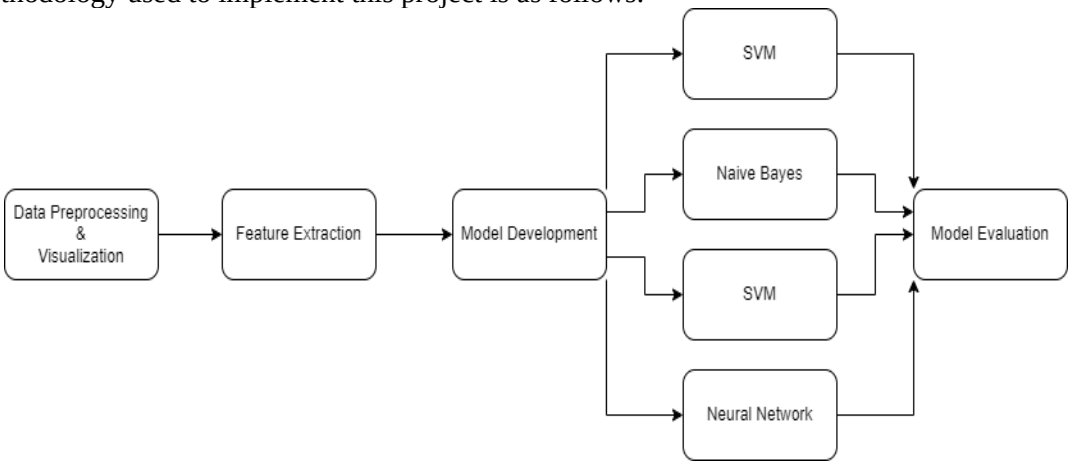

Figure 1: Methodology

4.1 Data Preprocessing & Visualization:

Since our input is text data it is important to clean the data before model development. Some basic data cleaning methods like removing stop words, digits, and special characters are implemented and data is visualized using word cloud to get patterns of words that occur frequently in sarcastic comments.

4.2 Feature Extraction:

Since we are working with text data, we will have to first convert the text data into a rich feature set for training purposes. We have used the count vectorizer technique in our project to convert the text data into a numerical feature vector.

4.3 Model Development:

To evaluate which classification model best fits for the Sarcasm detection problem we have tested the effectiveness of the below 4 models in identifying sarcastic expressions.

- SVM: Train a linear SVM model to find a hyperplane to maximize the margin between the two classes i.e. sarcastic and non–sarcastic.
- Naive Bayes: Use the Naive Bayse approach to find probabilities of each class (sarcastic/non-sarcastic) based on the occurrence of each word in the text document.
- Decision Tree: Train a decision tree using different splitting criteria so the nodes are split recursively till the pure leaf nodes are formed which can help in classification.
- Neural Network: Build a neural network with a different number of hidden layers and activation functions to find a model that has a good fit on our data.

4.4 Model Evaluation:

Evaluate and compare the performance of all the developed models using the below metrics.

a. *Accuracy = Number of correction Predictions / Total number of classifications*
b. *Precision = True sarcasm predictions / (True sarcasm predictions + False sarcasm prediction)*
c. *Recall = True sarcasm predictions / (True sarcasm predictions + False non-sarcasm prediction)*
d. *F1-Score* $= 2 \times \frac{Precision \times Recall}{Precision + Recall}$
e. *Confusion Matrix: A confusion matrix provides a tabular summary of a model's predictions, breaking down true positives, true negatives, false positives, and false negatives*.

5. Implementation:

5.1 Preprocessing:

Nltk library is used to perform the following preprocessing on our text data:

1. Stop words Removal
2. Removing digits and special characters
3. Removing any extra irrelevant data.

Preprocessing is important to avoid training on irrelevant data which can largely impact our models. Like for example the new headlines can have sometimes additional data written in brackets to give extra context. For example, a new headline may end with "(Continued on page 4)". Such extra data needs to be stemmed from input before training.

```
eat your veggies: 9 deliciously different recipes
'eat veggies deliciously different recipes'
```

Figure 2: Preprocessed Headline

5.2 Data Visualization & Analysis:

EDA is an important step as it serves as a quality assurance step by allowing us to verify the integrity and reliability of our dataset. This is crucial for ensuring that downstream analyses and models are built on trustworthy information.

Given our headline data, we have analyzed the distribution of words in both sarcastic and non-sarcastic comments using word clouds. The below word clouds give us a quick overview of the prominent words in both sarcastic and non-sarcastic comments.

Figure 3 Word cloud for sarcastic words

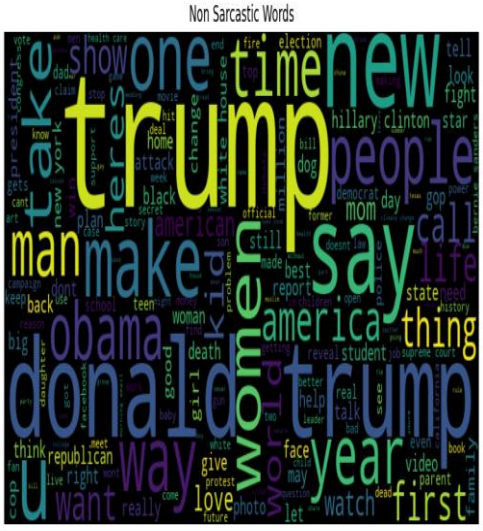

Figure 4 Word cloud for non-sarcastic words

We also verify that the data has a fair distribution of sarcastic and non-sarcastic comments to ensure that the model will not be biased.

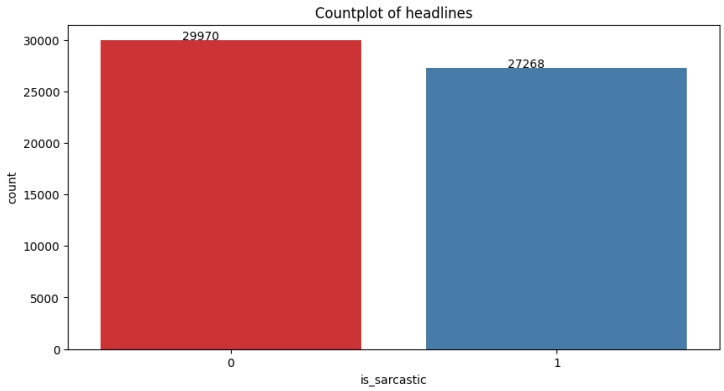

Figure 5: Countplot of headlines

As seen the data has a fair distribution of data with over 29970 non-sarcastic and 27268 sarcastic headlines.

5.3 Feature Extraction:

The Count Vectorizer technique is used to convert the text headlines into a matrix of token counts. It represents the number of occurrences of each word(token) in each headline.

	Number of Input Rows	**Number of Features**
Original Data	34342	1
Post Feature Extraction	34342	26207

```
Extracted feature set shape: (34342, 26207)

Orignal Input: briefly talk honestly weddings
Extracted Feature Set: [[0 0 0 ... 0 0 0]]
```

Figure 6 Output of Feature Extraction

The feature extraction converted our data into a rich feature set with 26207 attributes. Count Vectorizer is one of the simple feature extraction techniques. There are other feature extraction techniques like TFIDF which are complex and are useful in capturing context between words. However, since the focus of this project is to evaluate the machine learning models, we will do it using a simple Count Vectorizer technique.

5.4 Model Development:

5.4.1 SVM (Support Vector Classification):

- SVM aims at finding the best hyperplane that separates the two datasets.
- SVM uses multiple kernel functions to determine the hyperplane based on the nature of the dataset.
    - **Linear Kernel ($K(x,y)=x^T \cdot y$):**
      Useful when data is linearly seperable
    - **Polynomial Kernel ($K(x,y)=(x^T \cdot y+c)^d$):**
      Introduces non-linearity and is effective for capturing polynomial relationships in the data. However, higher degrees d can lead to overfitting.

- **Radial Basis Function (RBF) or Gaussian Kernel($K(x,y)=\exp(-(||x-y||^2/2\sigma^2)$):** Highly versatile and effective for capturing complex relationships. It is the most commonly used kernel and works well when there is no prior knowledge about the data.

We trained our dataset using all the above kernel functions to understand how the kernel functions affect our model fit.

Kernel Function	Linear	Polynomial (d=3)	Radial Basis
Train Score	0.995486576204065	0.9762681264923417	0.9883233358569682
Test Score	0.9067959468902865	0.8382250174703005	0.9104647099930119

As our data set is generated from text data, we don't have prior knowledge about the degree of the data. As seen the RBF kernel function as known works best in this case capturing complex relationships between the word tokens.

The RBF kernel implicitly maps input features into a higher-dimensional space. This can be beneficial when the relationships between features are not easily expressed in the original feature space. The SVM learns a decision boundary in this transformed space, allowing for more expressive modeling.

5.4.2 Naive Bayes

- Naive Bayes is a probabilistic machine learning method based on Bayes theorem.
- Use the Naive Bayse approach to find probabilities of each class (sarcastic/non-sarcastic) based on the occurrence of each word in the text document.
- Test with both the below 2 variants of the model.
    - Bernoulli NB: This model assumes that the features are binary.
    - Multinomial NB: Assumes that features are discrete.

Model	Bernoulli	Multinomial
Train Score	0.9322986430609749	0.9310756508065925
Test Score	0.8721174004192872	0.870020964360587

As seen since our data is discrete but sparse in nature (count of tokens in each headline), both Bernoulli and Multinomial NB have around the same accuracy and performance over the data.

Effects of Laplace Smoothening:

- Smoothening techniques are used to handle zero probability cases where certain words are missing in the training data. This helps with model generalization.
- With Laplace smoothening we can set the probability of unseen data from 0 (same as no smoothening applied) to 1.

We tested how the model behaves for different smoothening probabilities as below.

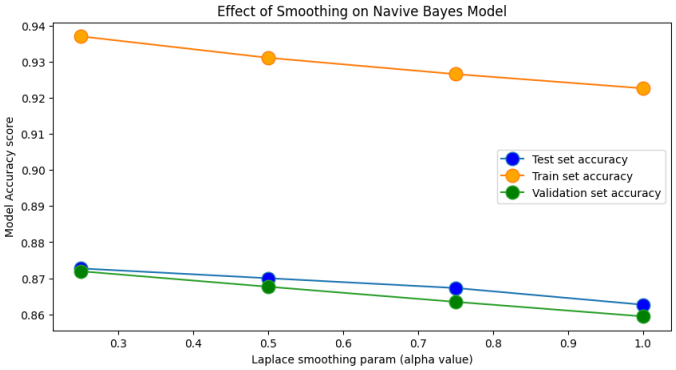

Figure 7 Laplace Smoothening

Interestingly, for our dataset, we notice that as the alpha value (probability used for laplace smoothening) increases both test and train accuracy decreases. This could be because our dataset is rich and has a good representation of all features, so setting alpha too high, might over-smooth the probabilities, essentially giving too much weight to the smoothing term and not relying enough on the actual counts from the training data. This could lead to a loss of information and a decrease in predictive performance.

To conclude, alpha = 0.5 looks like a good smoothening value for our data, as it won't overfit nor will over smoothen the data.

5.4.3 Decision Trees

- Decision Trees recursively partitions the dataset based on features selected using splitting criteria like Gini Index. It grows the tree structure until the lead node represents the final prediction or decision.
- Test the model with different splitting criteria like entropy, and Gini index and understand how the tree grows.

Splitting Criteria	Gini (no threshold)	Entropy (no threshold)	Gini (threshold = 4)	Gini (threshold =50)	Gini (threshold =500)
Train Score	1	1	0.581	0.714	0.917
Test Score	0.886	0.889	0.580	0.69	0.823
Tree Depth	1742	1754	4	50	500

As seen, for decision trees with depth over 500, the test accuracy doesn't improve much but the model starts to overfit on the training data increasing the train score.
Also notice that the decision between the impurity measure (Gini, Entropy) doesn't have any major impact on the model. However, we see that Gini tends to converge slightly faster with a tree depth of 1742, whereas Entropy grows the tree further to a depth of 1754.

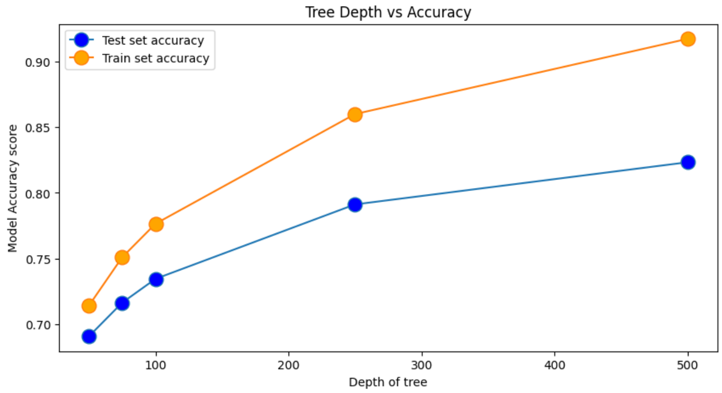

Figure 8: Tree Depth vs Accuracy

As seen from the figure as the depth of the tree increases the model starts to overfit the train data, but the test score doesn't improve much.

5.4.4 Neural Networks

Step 1: Word Embedding

Neural networks have an internal embedding layer that converts text data into word embedding. It does so by transforming word indices into dense vectors so it can capture the semantic relation between the words.

So far, we used the feature vector extracted using Count Vectorizer to train the models. But since the Neural network has its own embedding layer, let's covert the text data into a sequence of word indices which the model will further map into dense vectors.

1. Create a word to index mapping.
2. Replace each word in a sentence with its corresponding index.
3. Add padding so that all the sentences are converted into equal-length sequence.

Example:
Input (Text): 'briefly talk honestly weddings'
Output(Text Sequence): [2883, 246, 5696, 4813]
Padded Text Sequence:[[2883 246 5696 4813 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0]

Step 2: Develop the neural network

To determine the number of layers and parameters like dropout rate, epoch in our model, we tested the dataset using different model variants on our validation dataset as follows:

<table>
<tr><th>Model</th><th>Train Score</th><th>Validation Score</th></tr>
<tr><td><table><tr><th>Layer</th><th>Activation Function / Dropout Rate</th><th>Num of Nodes</th></tr><tr><td>Hidden</td><td>ReLU</td><td>24</td></tr><tr><td>Output</td><td>Sigmoid</td><td>1</td></tr></table></td><td>0.9248</td><td>0.8767</td></tr>
</table>

<table><tr><th>Layer</th><th>Activation Function / Dropout Rate</th><th>Num of Nodes</th></tr><tr><td>Hidden</td><td>ReLU</td><td>30</td></tr><tr><td>Dropout</td><td>0.5</td><td></td></tr><tr><td>Hidden</td><td>ReLU</td><td>20</td></tr><tr><td>Dropout</td><td>0.3</td><td></td></tr><tr><td>Output</td><td>Sigmoid</td><td>1</td></tr></table>	0.9979	0.9205
<table><tr><th>Layer</th><th>Activation Function / Dropout Rate</th><th>Num of Nodes</th></tr><tr><td>Hidden</td><td>ReLU</td><td>40</td></tr><tr><td>Dropout</td><td>0.2</td><td></td></tr><tr><td>Hidden</td><td>ReLU</td><td>20</td></tr><tr><td>Dropout</td><td>0.2</td><td></td></tr><tr><td>Hidden</td><td>ReLU</td><td>10</td></tr><tr><td>Dropout</td><td>0.2</td><td></td></tr><tr><td>Output</td><td>Sigmoid</td><td>1</td></tr></table>	0.9964	0.9211

We observe that adding more than 3 layers wont much affect the validation score, but might result in overfitting. So, we can conclude that the network with 3 layers is a good fit for our model.

Effect on Epoch:

The epoch parameter determines the number of passes the model makes over the training data set to learn the parameters. We ran the model for 15 epochs to understand how the loss and accuracy are affected with each pass.

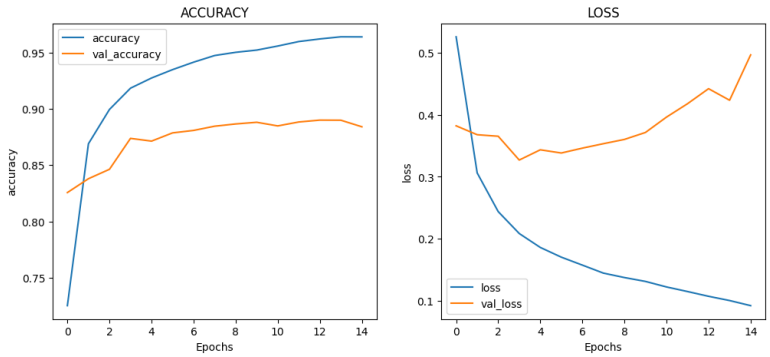

Figure 9: Epoch Effect

As seen from the figure after over 5 passes the validation loss starts to increase indicating overfitting. So, we can conclude that our model will run best with epoch = 5.

6. Evaluation

Now that we have developed our models, we will evaluate and compare them using different metrics.

6.1 Accuracy:

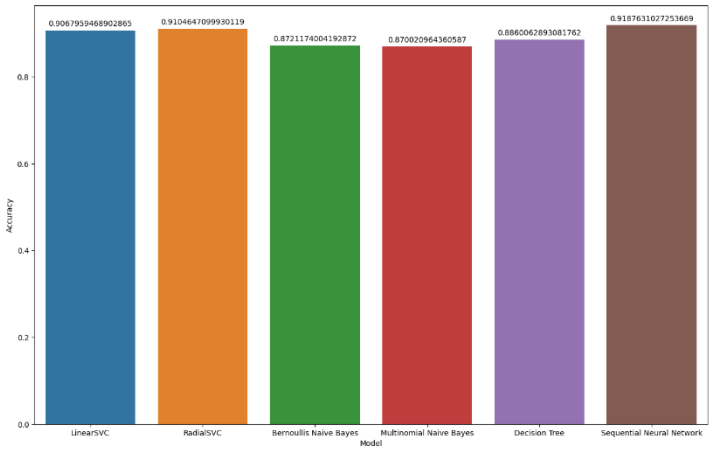

Figure 10: Accuracy

From the Accuracy measure, we can see that the neural network model and SVC with Radial kernel better fit the dataset. Other models aren't that bad either with accuracy greater than 85%.

6.2 Precision:

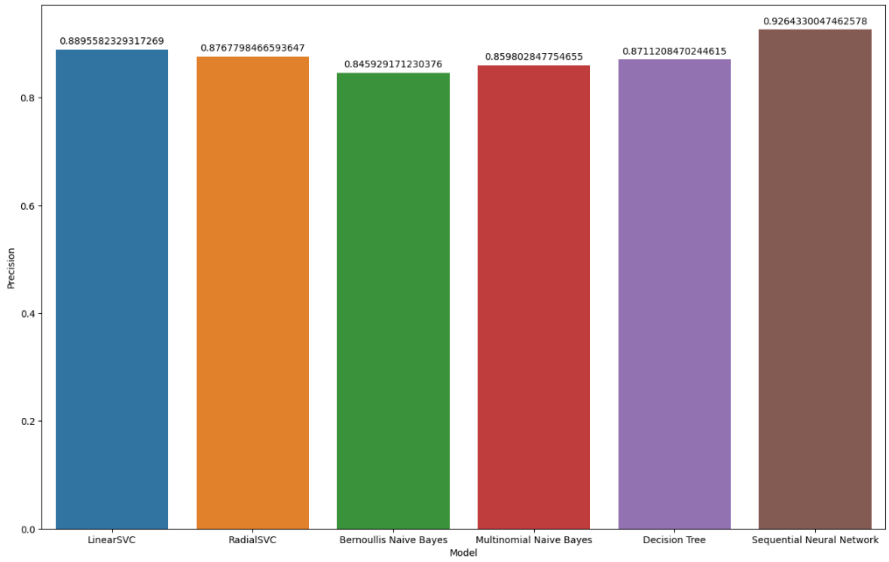

Figure 11: Precision

Again, we notice that the Neural network has better precision when compared to other models. Also, notice that though SVC with Radial Kernel had high accuracy its precisions is not that good indicating that the model doesn't accurately predict sarcastic comments.

6.3 Recall:

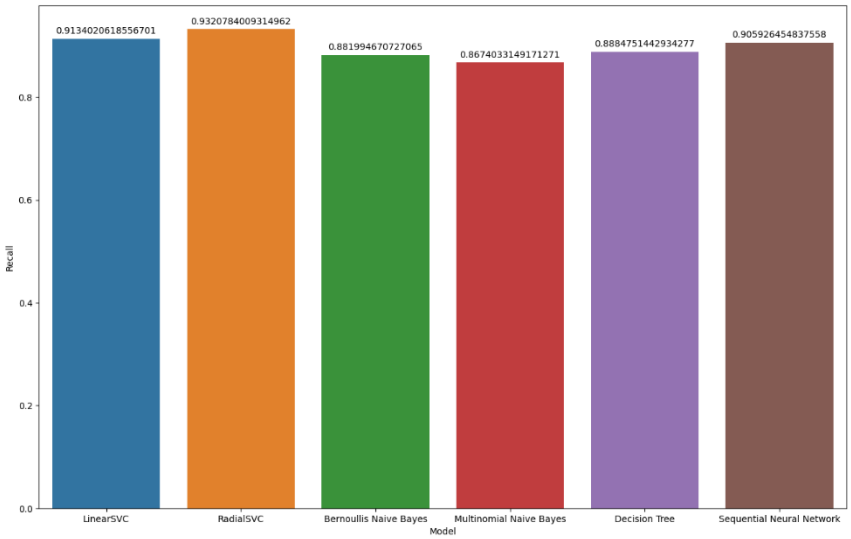

Figure 12: Recall

Both linear and Radial SVC have a higher recall, indicating that is good at predicting non-sarcastic data, but not that good at predicting sarcastic comments. Also, again neural network has a pretty decent recall as well.

6.4 F1-Score:

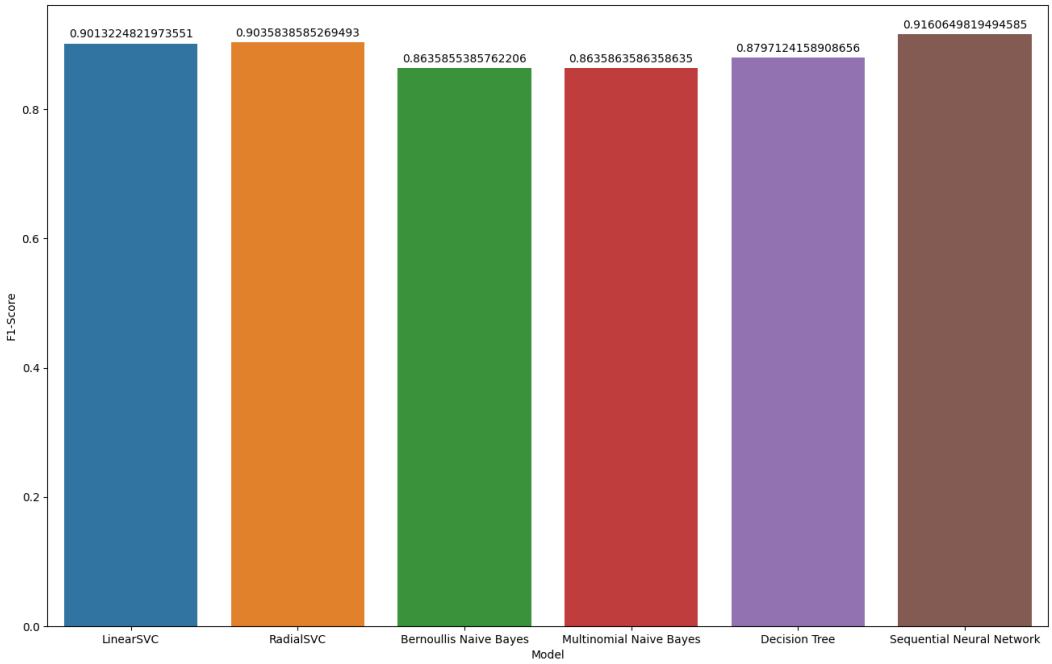

Figure 13: F1- Score

Since F1-Score gives the mean of precision and recall we see good values for both the SVC models and Neural network.

6.5 Confusion matrix:

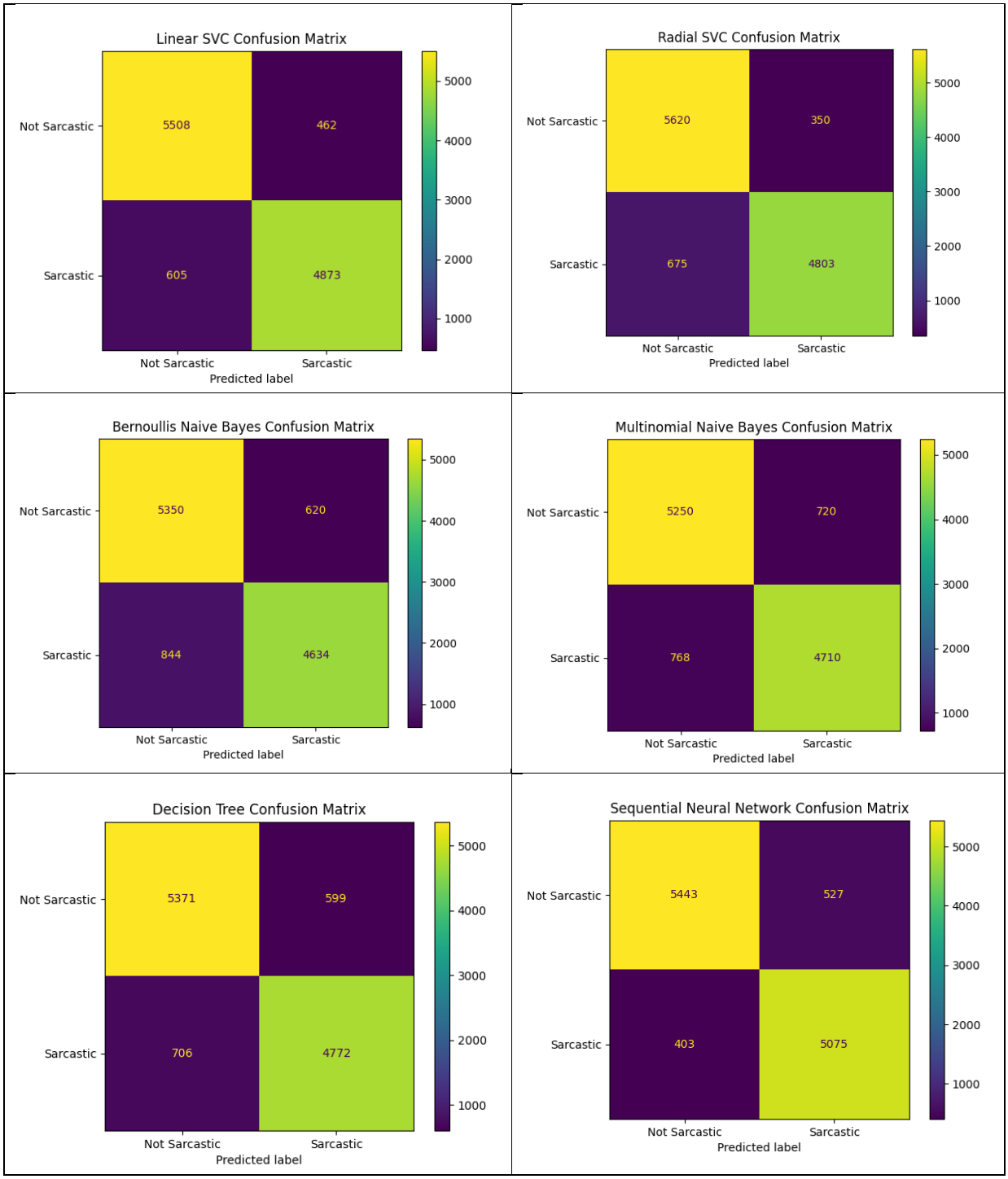

Figure 14: F1- Confusion matrix

From the confusion matrix, we can observe how each model is at classifying both sarcastic and non-sarcastic comments. All the models have correctly classified over 5000 non-sarcastic headlines and over 4000 sarcastic headlines. However, to conclude which works best let's look at the misclassified values which is the least for Neural Networks i.e 403+527 =930

Model selection:

From the above evaluation metrics, we can conclude that Neural networks work the best for our sarcasm detection data set.

7. Results

Now that we have evaluated the neural network to the best model developed so far, let's test it with some unseen data out of the dataset, to see how good it performs.

Test Input	Output
hello how are you?	Not sarcastic
Enraged Cow Injures Farmer with Ax	Sarcastic
Machine learning is easy	Not sarcastic
Juvenile Court to Try Shooting Defendant	Sarcastic
Red Tape Holds Up New Bridges	Sarcastic
Kids Make Nutritious Snacks	Not sarcastic
Man Struck By Lightning Faces Battery Charge	Not sarcastic

As seen from the output the model predicts the output pretty much accurately. Except last input "Man Struck By Lightning Faces Battery Charge" the model has correctly classified all the inputs.

Conclusion:

Accurate sarcasm detection has numerous applications and can significantly enhance the quality of communication analysis and decision-making in various domains. As a part of this project, we developed different classification models and compared their performance. Based on the metrics we concluded that Neural Networks worked best for our problem statement. This is because the embedding layer helps it capture the dependencies between the words/ tokens in a sentence. Interestingly we also observed how powerful a simple SVM classifier can be given the right kernel function. Naïve Bayes didn't work that well comparatively, it might have to do with its conditionally independent assumptions. In sarcasm detection, the presence of certain words or phrases might be strongly dependent on the context, making the independence assumption less realistic. Similarly, even the Decision tree didn't perform well due to overfitting given the rich feature set.

The models in this project are all trained over the feature set extracted using Count Vectorization. There are more complex feature extraction techniques like TFIDF, which also captures the context between the words. The discussed models could have a very different performance with different feature extraction techinque depending on the data captured in the extracted features.

In summary, sarcasm detection is a multifaceted problem that demands sophisticated models capable of contextual understanding and nuanced interpretation of language. The pursuit of more effective solutions continues to be an active area of research, with the potential to significantly enhance sentiment analysis and natural language understanding applications.

References:

1. Misra, Rishabh and Prahal Arora. "Sarcasm Detection using News Headlines Dataset." AI Open (2023).
2. Pre-trained Word Embedding using Glove in NLP models, Accessed on: 06 October 2023, [Online] Available: https://www.geeksforgeeks.org/pre-trained-word-embedding-using-glove-in-nlp-models/