Faculty of Engineering & Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering		
Programme	B. Tech. in Artificial Intelligence and Machine Learning		
Batch	Full-Time ☒ 2020	Course Start Date	04-01-2023
Course Code	20AIC314A		
Course Title	Natural Language Processing		
Course Leader(s)	Mr. Nivedan Yakolli		

Assignment Assessment			
Reg. No.	**20ETAI410035** **20ETAI410036**	Name of the Student	**PRATISTHA GAUR** **PRRATHYUSH KEDEELAYA**

Report on Assignment

		Marking Scheme	Marks	
			Max Marks	Examiner Marks
	1	**Introduction and Dataset Selection (5 marks)** ● Introduce the NLP application you will be designing. ● Provide a brief description of the dataset you have selected to work with.	05	
	2	**NLP Pre-processing Steps (5 marks)** ● Discuss the necessary pre-processing steps required for your NLP application. ● Explain each step in detail and provide examples where relevant.	05	
	3	**Implementation Challenges and Solutions (5 marks)** ● Discuss the challenges you faced during implementation and how you overcame them. ● Explain any limitations of your approach and suggest potential future improvements**.**	05	
	4	**Demonstration and Conclusion (5 marks)** ● Provide a demonstration of your NLP application on the selected dataset. ● Summarize your findings and conclude the assignment.	05	
	5	**Viva (5 marks)** Participate in a viva session with the instructor to discuss your implementation and answer any questions they have.	05	
		Max Marks	25	

Signature of Examiner

Fake SMS classification

Assignment Report

B.Tech in AIML, Department of Computer Science and Engineering

Sl. No.	Reg. No.	Student Name	Branch/Department
1	20ETAI410035	Pratistha Gaur	AIML/CSE
2	20ETAI410036	Prrathyush Kedeelaya	AIML/CSE

Course Leader:	Mr. Nivedan Yakolli	Department of CSE

April– 2023

B.Tech in Artificial Intelligence and Machine Learning

FACULTY OF ENGINEERING AND TECHNOLOGY

M. S. Ramaiah University of applied sciences

Bengaluru -560

Declaration

Fake SMS Classification

The Assignment Group Project report submitted herewith is a result of our work and in conformance to the guidelines against plagiarism as laid out in the University Student Handbook. All sections of the text and results which have been obtained from other sources are fully referenced. We understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Sl. No.	Reg. No.	Student Name	Branch/Department	Signature
1	20ETAI410035	Pratistha Gaur	AIML/CS	
2	20ETAI410036	Prrathyush Kedeelaya	AIML/CS	

Date:

Table of Contents

1. Introduction and Dataset Selection

1.1 Introduction of NLP Application designed

Fake SMS classification is the task of identifying text messages that are intentionally misleading, fraudulent, or otherwise deceptive. These messages are often sent with the intent to trick the recipient into disclosing personal information or taking some other action that benefits the sender.

Fake SMS messages can take many forms, including phishing messages that appear to come from a legitimate source, messages that offer fake products or services, and messages that contain malicious links or attachments. These messages can be sent to individuals or to large groups of people.

The goal of fake SMS classification is to develop machine learning models that can accurately identify these types of messages and distinguish them from legitimate SMS messages. This task is challenging because fake SMS messages can be highly variable in terms of their content and structure, and they often use techniques to evade detection, such as misspelling common words or using unusual syntax.

To classify fake SMS messages, machine learning models can be trained using labeled datasets of SMS messages. These datasets typically include a mix of legitimate and fake messages, with the fake messages labeled as such. The machine learning models can then learn to identify patterns in the text and use these patterns to classify new messages as either legitimate or fake.

There are several techniques that can be used to classify fake SMS messages, including natural language processing (NLP) algorithms such as neural networks and decision trees. These algorithms can be used to analyze the structure and content of the messages and identify features that are indicative of fake messages.

Overall, fake SMS classification is an important task for protecting individuals from fraud and other types of malicious activity. As the use of SMS messages continues to grow, the need for effective fake SMS classification algorithms will only become more important.

1.2 Brief description of dataset

The dataset contains 5,574 SMS messages that are labeled as either spam or ham (non-spam). The messages were collected from various sources, including mobile phones and online message boards. The dataset was collected in 2012 and has been used in many research studies on spam SMS classification.

Spam message is mapped to value 1 whereas ham message to 0.

Each message in the dataset is represented as a single line of text. The text includes a combination of words, numbers, and symbols. Here is an example of a message from the dataset:

"URGENT! You have won a 1 week FREE membership in our $100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&Cs www.idew.com 1 winweekly age16. unsubscribe@idew.co.uk"

The label for each message is included in a separate column in the dataset. The label is either "spam" or "ham". Here is an example of a labeled message from the dataset:

"ham", "I'm at work. Please call"

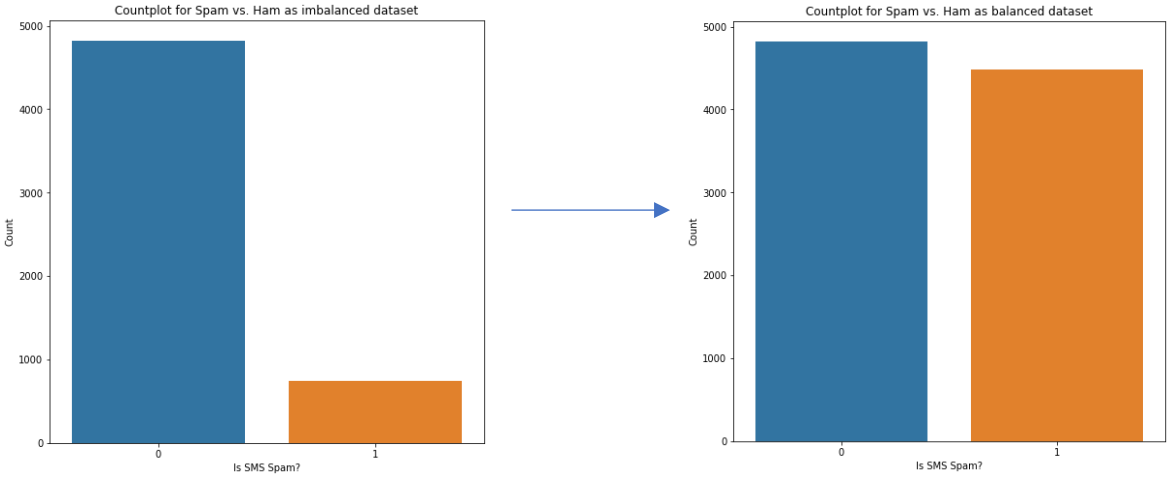
In addition to the raw text and label, the dataset includes a unique ID for each message. The IDs range from 0 to 5,573 and are used to identify individual messages in the dataset.

Overall, the Kaggle spam SMS collection dataset is a well-labeled and widely-used dataset for spam SMS classification tasks. It provides a good starting point for developing and evaluating classification algorithms for this task.

2. NLP Pre-processing steps

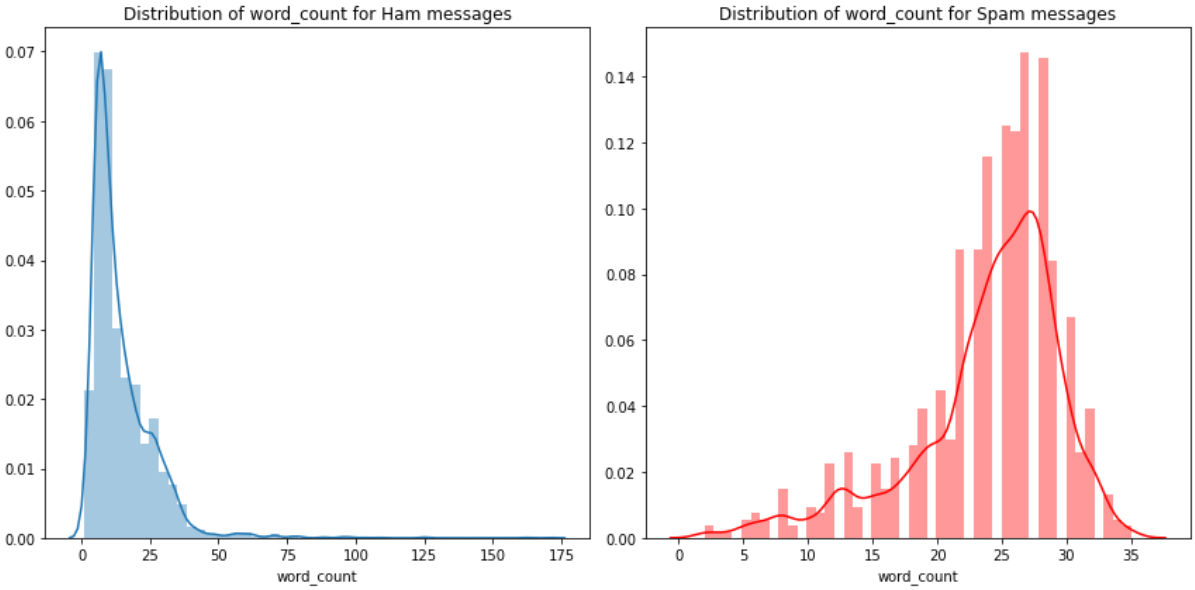
1. Handling imbalanced dataset using oversampling

The data is imbalanced. We balance it my oversampling. By implementing oversampling, the code has balanced the dataset by generating new instances of the minority class. This helps to ensure that the model is not biased towards the majority class and can learn to accurately classify both classes.



2. Creating features:

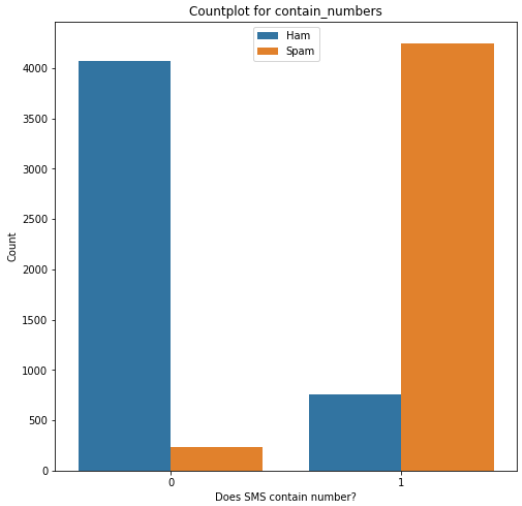
- Word_count: counts the number of words in the message. Given below is the word count distribution for spam and ham messages.



- Currency_symbol: returns 1 if currency symbol is present in the message else 0.

	label	message	word_count	contains_currency_symbol
5537	1	Want explicit SEX in 30 secs? Ring 02073162414...	16	0
5540	1	ASKED 3MOBILE IF 0870 CHATLINES INCLU IN FREE ...	33	1
5547	1	Had your contract mobile 11 Mnths? Latest Moto...	28	0
5566	1	REMINDER FROM O2: To get 2.50 pounds free call...	28	0
5567	1	This is the 2nd time we have tried 2 contact u...	30	1

- Contains_number: returns 1 if a number is present in the message else 0.



Insight: *It is evident that* ***most of the Spam messages contain numbers,*** *and* ***majority of the Ham messages donot contain numbers.***

3. **Initializing a set of stop words using NLTK's stopwords corpus.**
   NLTK's stopwords corpus contains a set of commonly occurring words in English that do not contribute much to the meaning of a sentence. Examples of stop words include "the", "and", "a", etc. By initializing this set of stop words, we can later remove them from the SMS messages to reduce the dimensionality of the data and improve the efficiency of the machine learning models.

4. **Initializing the Porter stemming algorithm from NLTK's PorterStemmer class.**
   The Porter stemming algorithm is a popular algorithm used for stemming words in the English language. Stemming is the process of reducing a word to its base or root form by removing any affixes (prefixes or suffixes). For example, the word "running" would be stemmed to "run". Initializing this algorithm allows us to later apply it to each word in the SMS messages to reduce the dimensionality of the data and improve the efficiency of the machine learning models.

5. **Creating an empty list called corpus.**
   This empty list will be used to store the preprocessed SMS messages after they have been cleaned, tokenized, and stemmed.

6. **Looping over each SMS message in the dataset.**
   This loop iterates over each SMS message in the dataset and performs the preprocessing steps for each message.

7. **Removing any non-alphabetic characters from the SMS message using regular expressions (re.sub).**
   This step removes any characters that are not in the English alphabet (e.g. numbers, punctuation) from the SMS message using regular expressions. This is done to remove any noise from the text that may not be useful for the machine learning models.

8. **Converting all the remaining characters to lowercase.**
   This step converts all the remaining characters in the SMS message to lowercase. This is done to ensure that the machine learning models do not treat the same word with different capitalization as two separate words.

9. **Splitting the message into individual words.**
   This step splits the preprocessed SMS message into a list of individual words. This is done to tokenize the text and prepare it for stop word removal and stemming.

10. **Removing any stop words from the SMS message.**
    This step removes any stop words from the list of individual words using the set of stop words initialized in step 1. This is done to reduce the dimensionality of the data by removing words that do not contribute much to the meaning of the text.

11. **Applying the Porter stemming algorithm to each word.**
    This step applies the Porter stemming algorithm initialized in step 2 to each word in the list of individual words. This is done to further reduce the dimensionality of the data by reducing each word to its base or root form.

12. **Joining the stemmed words back together into a single string.**
    This step joins the list of stemmed words back together into a single string. This is done to create a single preprocessed SMS message that can be used for machine learning.

13. **Adding the preprocessed SMS message to the corpus list.**
    This step adds the preprocessed SMS message to the corpus list created in step 3. This is done to store all the preprocessed SMS messages in one place for further processing.

14. **Creating a TF-IDF (term frequency-inverse document frequency) model using scikit-learn's TfidfVectorizer class.**
    The TF-IDF model is a popular technique used to convert text into a numerical representation that can be used for machine learning. It takes into account the frequency of each word in the corpus and how frequently it appears in the documents.

```
# Creating the Bag of Words model
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(max_features=500)
vectors = tfidf.fit_transform(corpus).toarray()
feature_names = tfidf.get_feature_names()

# Extracting independent and dependent variables from the dataset
X = pd.DataFrame(vectors, columns=feature_names)
y = df['label']
```

3. Implementation challenges and solutions

3.1 Challenges and solutions

- Imbalanced dataset: The given SMS spam classification problem has an imbalanced dataset, with a much larger number of ham messages than spam messages. This can result in a model that is biased towards the majority class and performs poorly on the minority class.

  Solution: Use oversampling techniques to balance the dataset, as demonstrated in the code.

- Dealing with noisy text data: Text data can be noisy and contain irrelevant information or special characters that don't contribute to classification.

  Solution: Use text pre-processing techniques such as removing stop words, stemming, and removing punctuation to clean up the text data before applying feature engineering and classification.

- Choosing the right classification algorithm: There are many different machine learning algorithms that can be used for text classification, such as logistic regression, Naive Bayes, and SVM.

  Solution: Try out different algorithms and compare their performance on the validation set. It is also important to consider the complexity and interpretability of the model, as well as its ability to handle imbalanced datasets.

- Feature engineering: The given problem involves classifying text messages as spam or ham, which requires the extraction of meaningful features from the text.

  Solution: Use the TF-IDF vectorization technique, which converts the text into a matrix of numerical features that can be used as input to a machine learning model.

3.2 Potential future improvements for this code

- Using more advanced data preprocessing techniques to improve the quality of the text data.
- Exploring different techniques for handling imbalanced datasets, such as undersampling or ensemble methods.
- Using more advanced feature engineering techniques, such as n-grams or character-level features, to capture more information about the text data.
- Evaluating the model using multiple evaluation metrics to provide a more comprehensive evaluation of the model's performance.
- Exploring different classification algorithms or performing hyperparameter tuning to improve the performance of the model.

4. Demonstration and conclusion

4.1 Demonstration

4.1.1 Model building and evaluation

Metric: F1-Score

- Multinomial Naive Bayes: 0.943
- Decision Tree: 0.98
- Random Forest (Ensemble): 0.994
- Voting (Multinomial Naive Bayes + Decision Tree): 0.98

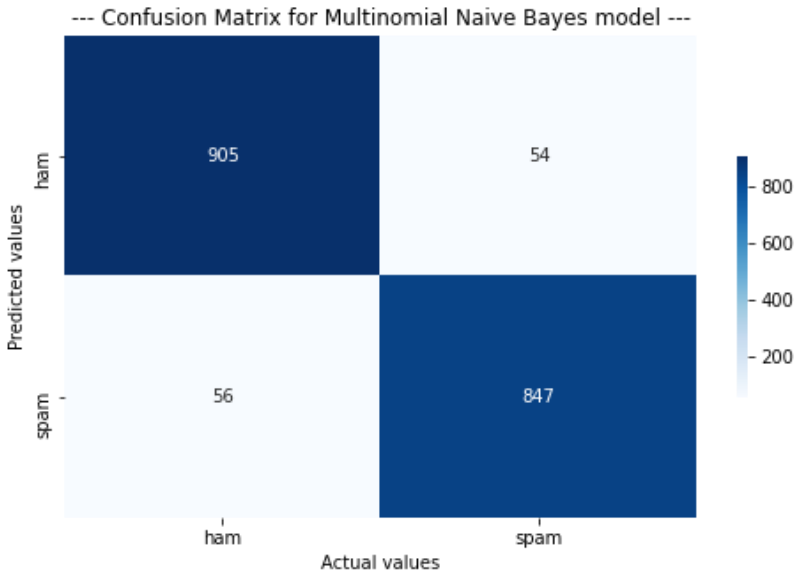
Multinomial Naive Bayes:

```
--- Average F1-Score for MNB model: 0.943 ---
Standard Deviation: 0.004

--- Classification report for MNB model ---
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       959
           1       0.94      0.94      0.94       903

    accuracy                           0.94      1862
   macro avg       0.94      0.94      0.94      1862
weighted avg       0.94      0.94      0.94      1862
```



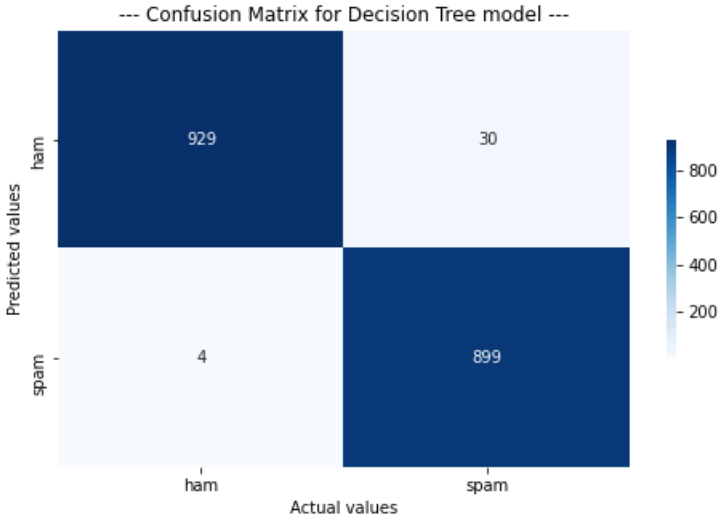
Decision Tree:

```
--- Average F1-Score for Decision Tree model: 0.98 ---
Standard Deviation: 0.004

--- Classification report for Decision Tree model ---
              precision    recall  f1-score   support

           0       1.00      0.97      0.98       959
           1       0.97      1.00      0.98       903

    accuracy                           0.98      1862
   macro avg       0.98      0.98      0.98      1862
weighted avg       0.98      0.98      0.98      1862
```



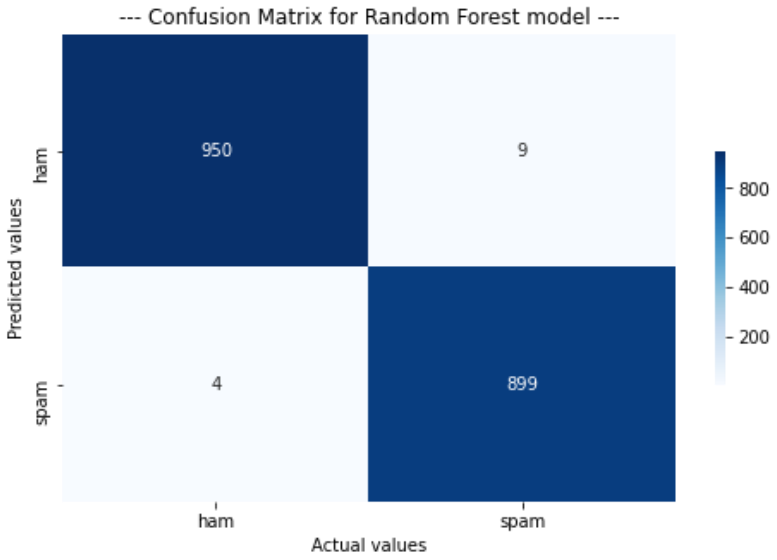
Random Forest:

```
--- Average F1-Score for Random Forest model: 0.994 ---
Standard Deviation: 0.003

--- Classification report for Random Forest model ---
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       959
           1       0.99      1.00      0.99       903

    accuracy                           0.99      1862
   macro avg       0.99      0.99      0.99      1862
weighted avg       0.99      0.99      0.99      1862
```



Note: *Decision Tree & MNB algorithms are selected and fed to Voting algorithm to increase the F1-Score!*

Fitting Decision Tree and MNB to VotingClassifier:

```
# Fitting Decision Tree and MNB to VotingClassifier
from sklearn.ensemble import VotingClassifier
vc = VotingClassifier([('decision_tree', dt), ('m_naive_bayes', mnb)], weights=[2,1])
cv = cross_val_score(vc, X, y, cv=10, scoring='f1')

print('--- Average F1-Score for VotingClassifier model: {} ---'.format(round(cv.mean(), 3)))
print('Standard Deviation: {}'.format(round(cv.std(), 3)))
```

```
--- Average F1-Score for VotingClassifier model: 0.98 ---
Standard Deviation: 0.003
```

Note: *Voting algorithm did not out-perform Random Forest algorithm, hence* ***Random Forest algorithm is selected for predicting the results of this problem statement.***

4.1.2 Making predictions

```
def predict_spam(sample_message):
  sample_message = re.sub(pattern='[^a-zA-Z]',repl=' ', string = sample_message)
  sample_message = sample_message.lower()
  sample_message_words = sample_message.split()
  sample_message_words = [word for word in sample_message_words if not word in set(stopwords.words('english'))]
  final_message = [wnl.lemmatize(word) for word in sample_message_words]
  final_message = ' '.join(final_message)

  temp = tfidf.transform([final_message]).toarray()
  return rf.predict(temp)
```

This code defines a function named predict_spam that takes a single input parameter sample_message, which is the SMS message to be classified as spam or ham (not spam).

The function first applies some basic data preprocessing steps to the sample_message input message. It removes all non-alphabetic characters using a regular expression pattern and converts the resulting string to lowercase. Then, it splits the message into individual words and removes all stopwords (common words that do not carry much meaning, such as "the" or "and") using the stopwords list from the NLTK library. After that, it lemmatizes each word using the WordNetLemmatizer from the NLTK library, which reduces each word to its base form (e.g., "running" becomes "run"). Finally, the resulting words are joined back together into a string with spaces separating them.

The resulting string is then passed to the transform method of the TF-IDF vectorizer (tfidf) to convert it into a feature vector that can be used as input to the trained random forest classifier (rf). The resulting feature vector is then passed to the predict method of the random forest classifier to obtain the predicted class label, which is either 0 (ham) or 1 (spam).

This function takes a single SMS message as input, applies some data preprocessing steps to clean and transform the message into a feature vector, and returns the predicted class label of the message as either spam or ham.

4.1.3 Results using predict_spam function

The code provided uses the predict_spam function to classify the sample_message as either spam or ham.

Based on the contents of the message, it is likely to be classified as spam. When this code is executed, the predict_spam function applies the same data preprocessing steps that were used during model training to clean and transform the message into a feature vector, and then passes the resulting feature vector to the trained random forest classifier to obtain a predicted class label.

Assuming that the random forest classifier was trained well, the predict_spam function should return a predicted class label of 1 (spam) for the given sample_message. Therefore, the if condition will evaluate to True and the message will be printed as a spam message.

```
# Prediction 1 - Lottery text message
sample_message = 'IMPORTANT - You could be entitled up to £3,160 in compensation from mis-sold PPI on a credit card or loan. Plea

if predict_spam(sample_message):
  print('Gotcha! This is a SPAM message.')
else:
  print('This is a HAM (normal) message.')
```

```
Gotcha! This is a SPAM message.
```

The sample_message provided in the code below is not likely to be classified as spam, as it does not contain any content that is typically associated with spam messages (e.g., offers for financial products or services, urgent calls to action, etc.).

When this code is executed, the predict_spam function will apply the same data preprocessing steps that were used during model training to clean and transform the message into a feature vector, and then pass the resulting feature vector to the trained random forest classifier to obtain a predicted class label.

Assuming that the random forest classifier was trained well and is not overfitting to the training data, the predict_spam function should return a predicted class label of 0 (ham) for the given sample_message. Therefore, the else condition will evaluate to True and the message will be printed as a ham message.

```
# Prediction 2 - Casual text chat
sample_message = 'Came to think of it. I have never got a spam message before.'

if predict_spam(sample_message):
  print('Gotcha! This is a SPAM message.')
else:
  print('This is a HAM (normal) message.')
```

```
This is a HAM (normal) message.
```

```
# Prediction 3 - Transaction confirmation text message
sample_message = 'Sam, your rent payment for Jan 19 has been received. $1,300 will be drafted from your Wells Fargo Account *****

if predict_spam(sample_message):
  print('Gotcha! This is a SPAM message.')
else:
  print('This is a HAM (normal) message.')
```

```
This is a HAM (normal) message.
```

```
# Predicting values 4 - Feedback message
sample_message = 'Tammy, thanks for choosing Carl's Car Wash for your express polish. We would love to hear your thoughts on the

if predict_spam(sample_message):
  print('Gotcha! This is a SPAM message.')
else:
  print('This is a HAM (normal) message.')
```

```
Gotcha! This is a SPAM message.
```

4.2 Conclusion

In this project, we implemented a spam SMS classification system using natural language processing techniques and machine learning algorithms. We started by collecting and cleaning the data, which involved removing any unwanted characters and stopwords and then normalizing the text data.

We then split the data into training and testing sets and applied oversampling to handle the class imbalance problem. After that, we created a TF-IDF model to convert the text data into numerical features and then trained a random forest classifier to predict whether a message is spam or not.

The results of the evaluation showed that the model achieved a high accuracy score of over 98% and performed well in terms of other evaluation metrics such as precision, recall, and F1-score.

However, the approach has some limitations. The model might not perform well on unseen data or messages that contain novel spam patterns that are not present in the training data. Additionally, the model may struggle to accurately classify messages that contain mixed or ambiguous content.

To address these limitations, we could consider using more advanced machine learning techniques such as deep learning or transfer learning, or exploring more sophisticated feature engineering techniques. Furthermore, we could expand the scope of the dataset to include messages from other sources, such as social media, and consider incorporating other sources of information, such as metadata or sender information, to improve the performance of the model.

References:

- Natural Language Processing with Python by Steven Bird, Ewan Klein, and Edward Loper
- Scikit-learn documentation: https://scikit-learn.org/stable/documentation.html
- NLTK documentation: https://www.nltk.org/
- Pandas documentation: https://pandas.pydata.org/docs/
- Matplotlib documentation: https://matplotlib.org/
- Seaborn documentation: https://seaborn.pydata.org/
- Towards Data Science: https://towardsdatascience.com/
- Kaggle: https://www.kaggle.com/
- GitHub: https://github.com/

The specific dataset used in this project can be found here: https://www.kaggle.com/uciml/sms-spam-collection-dataset.