```
Task 1: SMS Classifier Project

Author: Bandana Prakash
Batch :June
Domain: Data Science

Overview
Task 1 propelled me into the world of text classification, where I tackled the intricate task of distinguishing spam from non-spam SMS messages. Armed with Python prowess and NLP techniques, I crafted a model to sift through messages with precision. The goal? To streamline communication and enhance efficiency!

Project Details
Developed a model to distinguish spam from non-spam SMS messages.
Utilized Python prowess and NLP techniques.
Aimed to streamline communication and enhance efficiency.
```

In [24]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import confusion_matrix
from wordcloud import WordCloud
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")

import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/bandanaprakash/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/bandanaprakash/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]: True

Reading the Dataset

In [3]:
```
df = pd.read_csv('SMS.csv').rename(columns={'sms':'text'})
df.head()
```

Out[3]:

	text	label
0	Go until jurong point, crazy.. Available only ...	0
1	Ok lar... Joking wif u oni...\n	0
2	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	U dun say so early hor... U c already then say...	0
4	Nah I don't think he goes to usf, he lives aro...	0

Exploratory Data Analysis

In [4]:
```
df.shape
```

Out[4]: (5574, 2)

In [5]:
```
df.isna().sum()
```

Out[5]:
```
text     0
label    0
dtype: int64
```

In [6]:
```
df.duplicated().sum()
```

Out[6]:
```
403
```

In [7]:
```
df = df.drop_duplicates()
df.shape
```

Out[7]:
```
(5171, 2)
```

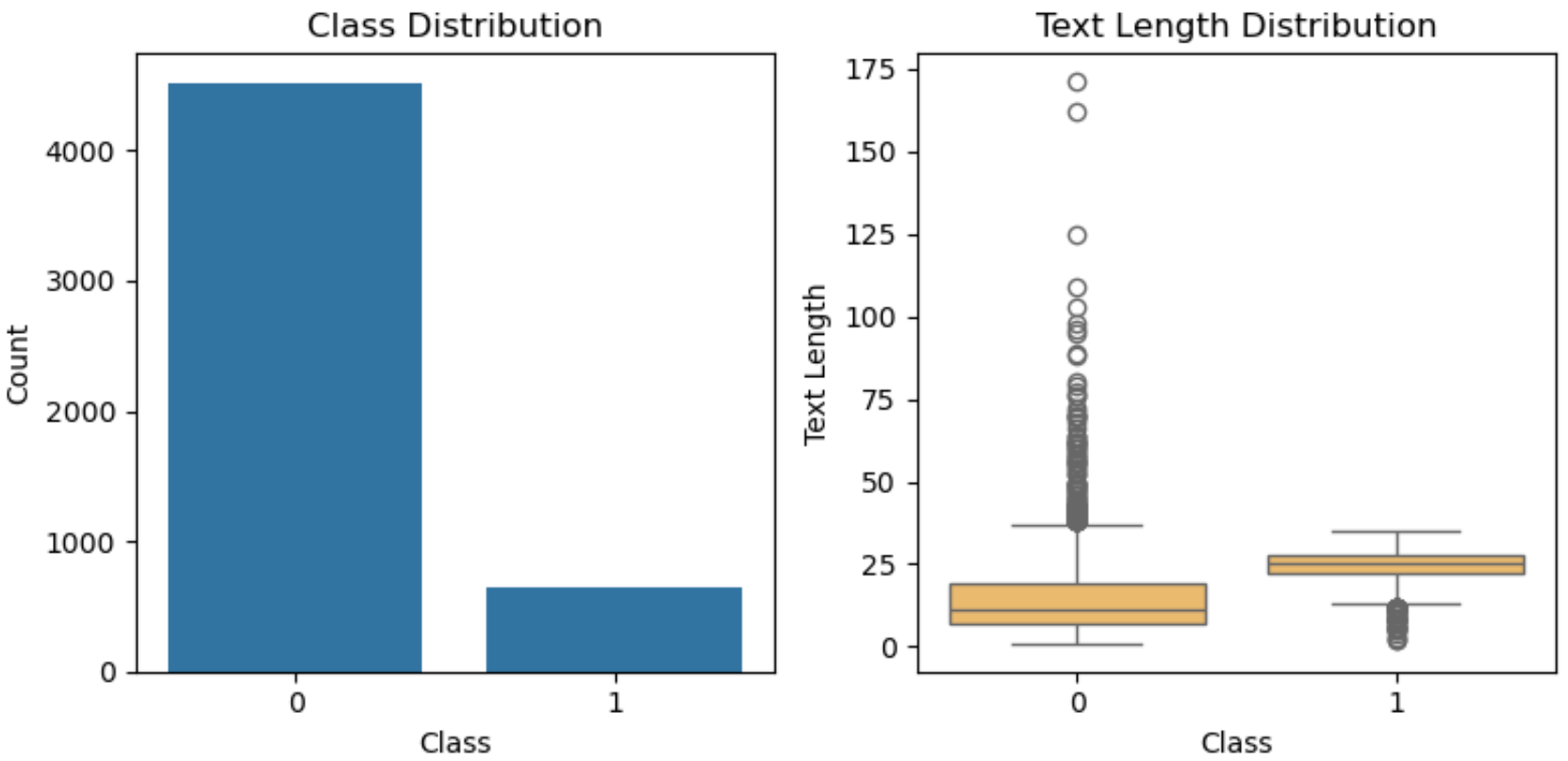
In [8]:
```
df.text.str.isspace().sum()
```

Out[8]:
```
0
```

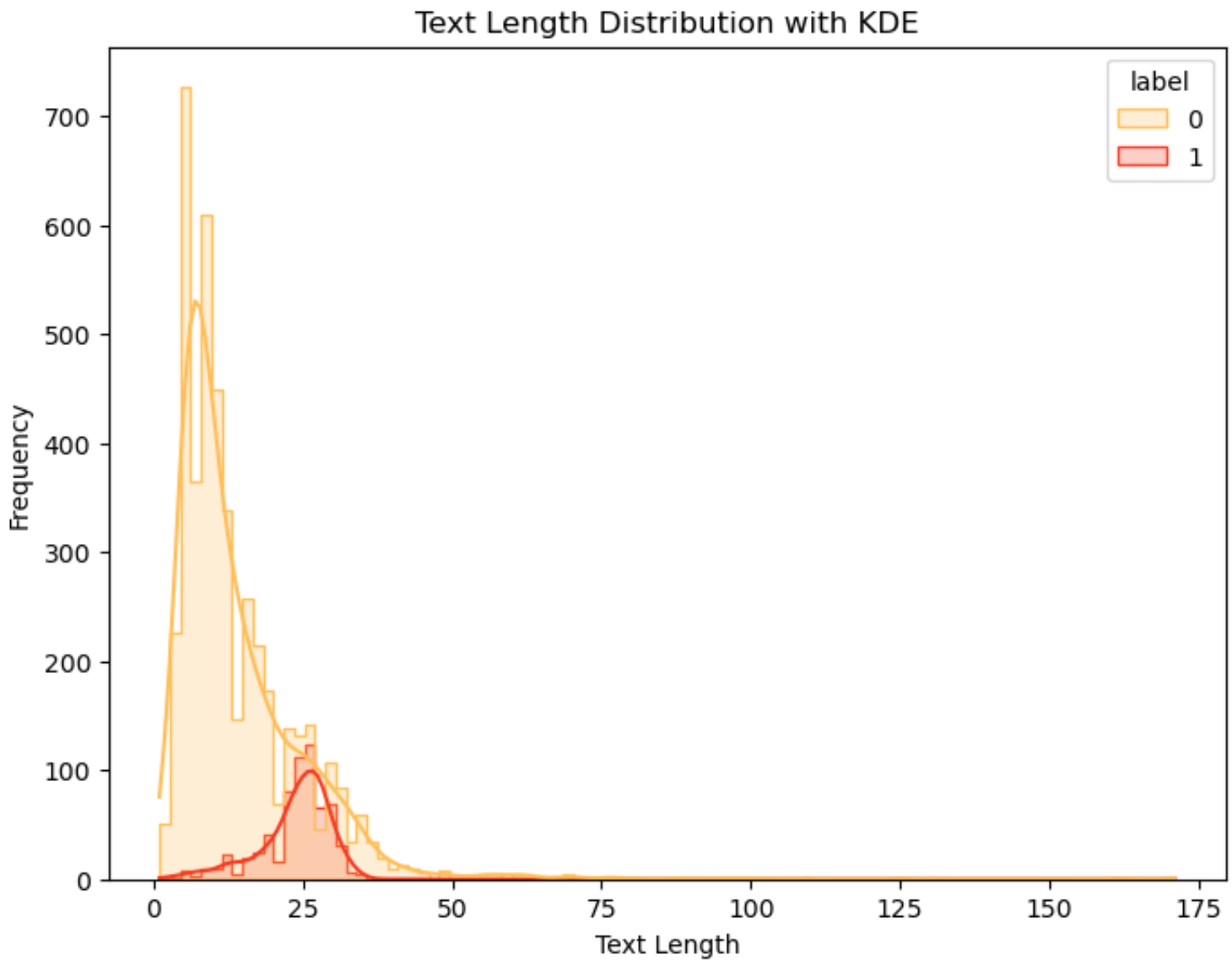
Visualization

In [9]:
```
red_palette = sns.color_palette("YlOrRd", 2)
red_palette_c = sns.color_palette("YlOrRd", as_cmap=True)
```

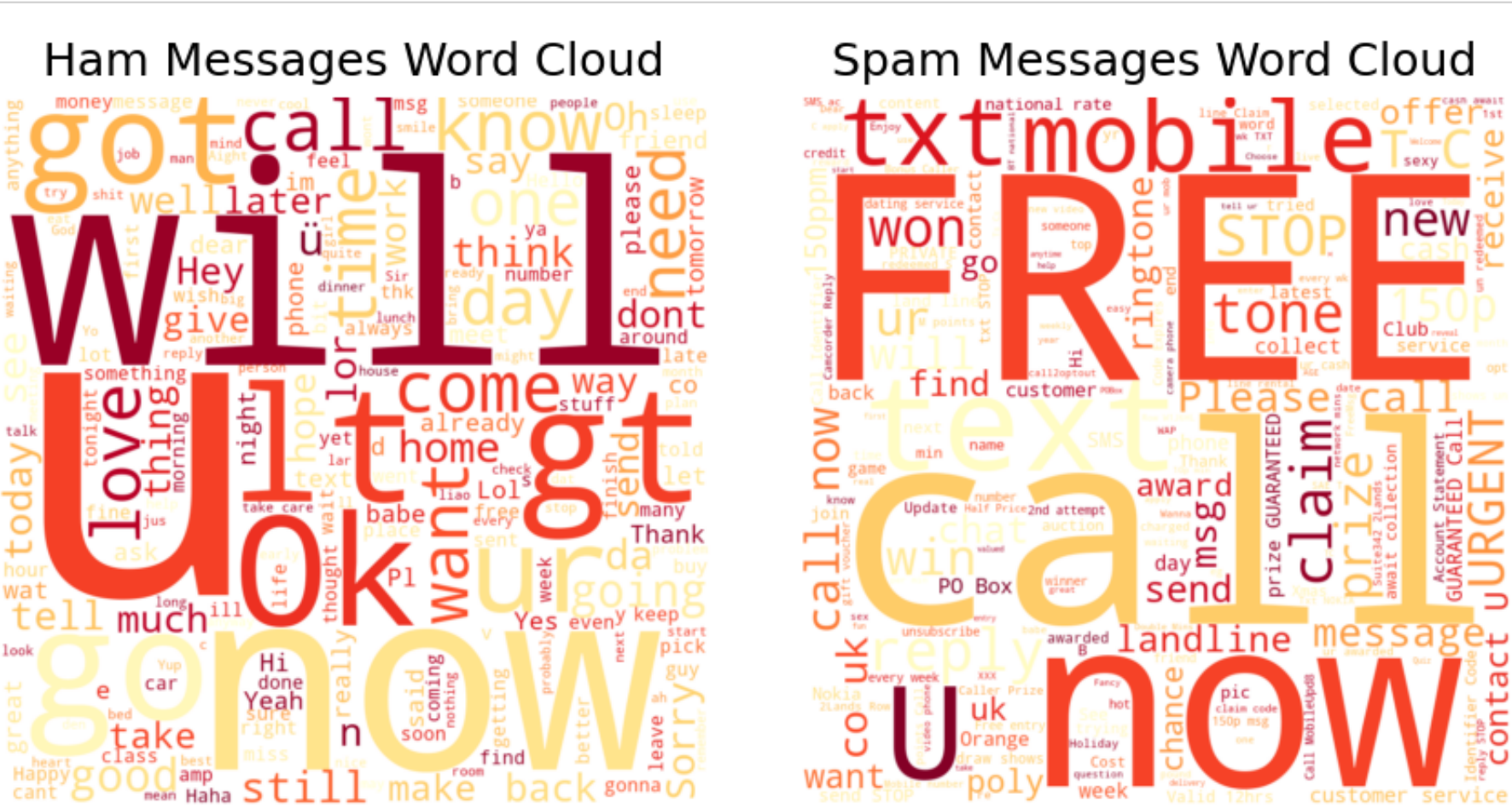
In [10]:
```
plt.figure(figsize=(8, 4), dpi=100)
plt.subplot(1, 2, 1)
sns.set_palette(red_palette)
sns.countplot(x='label', data=df)
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
sns.set_palette(red_palette)
df['text_length'] = df['text'].apply(lambda x: len(x.split()))
sns.boxplot(x='label', y='text_length', data=df)
plt.title('Text Length Distribution')
plt.xlabel('Class')
plt.ylabel('Text Length')
plt.tight_layout()
plt.show()
```



In [11]:
```
plt.figure(figsize=(8, 6), dpi=100)
sns.set_palette(red_palette)
sns.histplot(data=df, x='text_length', hue='label', kde=True, element='step')
plt.title('Text Length Distribution with KDE')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.show()
```



In [12]:
```
ham_text = " ".join(df[df['label'] == 0]['text'])
spam_text = " ".join(df[df['label'] == 1]['text'])

ham_wordcloud = WordCloud(width=800, height=800, background_color='white', colormap=red_palette_c).gener
spam_wordcloud = WordCloud(width=800, height=800, background_color='white', colormap=red_palette_c).gene

ham_image = ham_wordcloud.to_array()
spam_image = spam_wordcloud.to_array()

plt.figure(figsize=(6, 3), dpi=150)

plt.subplot(1, 2, 1)
plt.imshow(ham_image, interpolation='bilinear')
plt.title('Ham Messages Word Cloud')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(spam_image, interpolation='bilinear')
plt.title('Spam Messages Word Cloud')
plt.axis('off')

plt.tight_layout()
plt.show()
```



Pre-processing

In [13]:
```
def preprocess_text(text):
    words = word_tokenize(text) #Tokenization
    words = [word.lower() for word in words if word.isalnum()] #to Lowercase
    words = [word for word in words if word not in stopwords.words("english")] #Remove Stopwords
    return " ".join(words) #Concate tokens
```

In [14]:
```
df['text'] = df['text'].apply(preprocess_text)
```

TF-IDF Vectorization and Multinomial Naive Bayes Classification

In [15]:
```
tfidf_vectorizer = TfidfVectorizer(max_features=1000, ngram_range=(1, 2))
X = tfidf_vectorizer.fit_transform(df['text']).toarray()
y = df['label']
```

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [17]:
```
sklearn_classifier = MultinomialNB(alpha=.1) #alpha=0.1 is more accurate for our model
sklearn_classifier.fit(X_train, y_train)
```

Out[17]:
```
MultinomialNB(alpha=0.1)
```

NLTK Classifier Wrapper

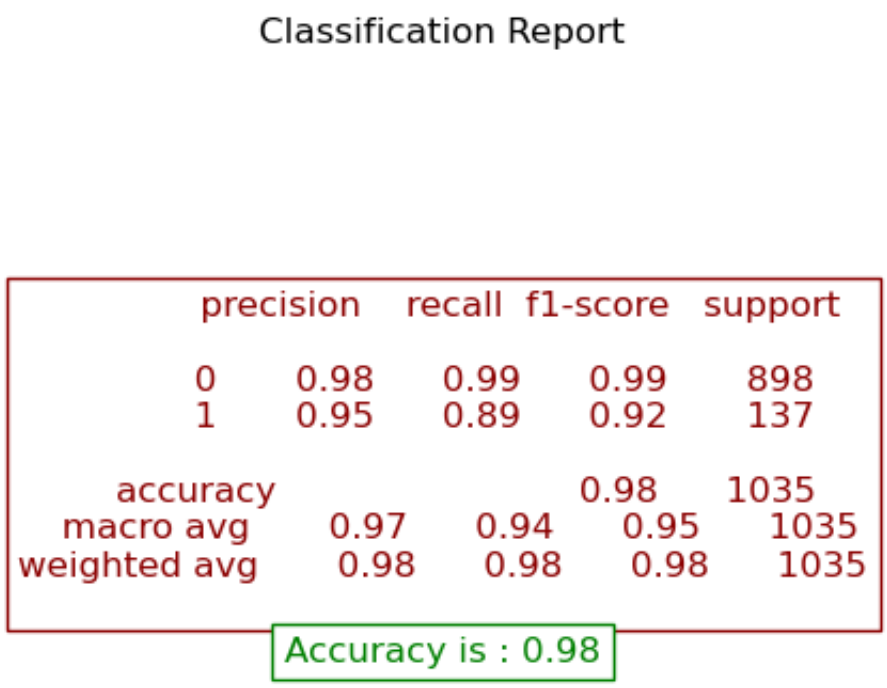
In [18]:
```
class SklearnNLTKClassifier(nltk.classify.ClassifierI): #Constructor
    def __init__(self, classifier):
        self._classifier = classifier

    def classify(self, features): #Predict for one feature
        return self._classifier.predict([features])[0]

    def classify_many(self, featuresets): #Predict for multiple features
        return self._classifier.predict(featuresets)

    def prob_classify(self, features): #Shows error for not implementating
        raise NotImplementedError("Probability estimation not available.")

    def labels(self): #return labels
        return self._classifier.classes
```

In [19]:
```
nltk_classifier = SklearnNLTKClassifier(sklearn_classifier)
```

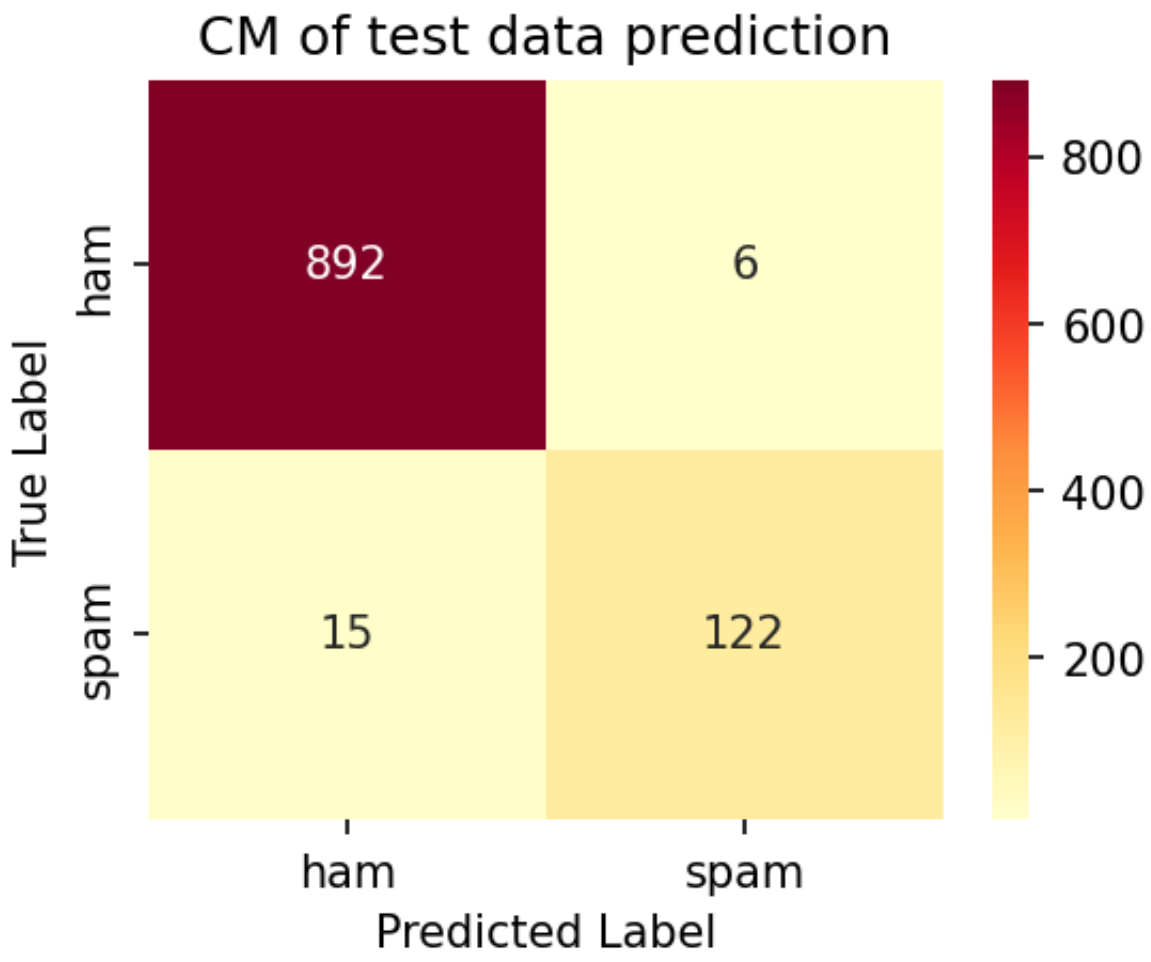
Prediction for test data

In [20]:
```
y_pred = nltk_classifier.classify_many(X_test)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
acc = f"Accuracy is : {accuracy:.2f}"
```

In [21]:
```
plt.figure(figsize=(8, 6), dpi=100)
plt.text(0.5, 0.6, report, fontsize=12, color='darkred', ha='center', va='center', bbox=dict(facecolor='
plt.text(0.5, 0.4, acc, fontsize=12, color='Green', ha='center', va='center', bbox=dict(facecolor='white
plt.title('Classification Report')
plt.axis('off')
plt.show()
```



In [22]:
```
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(4, 3), dpi=150)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='YlOrRd', xticklabels=['ham', 'spam'], yticklabels=['
plt.title('CM of test data prediction')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



In [23]:
```
from sklearn.metrics import roc_auc_score
r_a_score = roc_auc_score(y_test, y_pred)
print("ROC-AUC-Score:", r_a_score)
```

```
ROC-AUC-Score: 0.9419147172142475
```