Spam Classification

SVM

PRESENTATION BY:
NATHAN CHRISTOPHER MENON

DATAPLAY

Agenda

01. Objective

02. Collection and Preparing

03. Dataset

04. Feature Engineering

05. Model creation, training and evaluation

DATAPLAY

Objective

We will use Support Vector Machine for:

- Classifying the messages as spam or not spam(ham)
- Implementing basic Natural Language Processing,
- By using the Term Frequency-Inverse Document Frequency and Count Vectorizer

Step one: Importing Libraries

- Pandas for data manipulation
- Sklearn for setting up the train-test split, importing CountVectorizer, Support Vector Machine, classification report and confusion matrix
- Imblearn for importing SMOTE for resampling the data
- NumPy for numerical computations

```
import pandas as pd
from sklearn.model_selection import
train_test_split
from sklearn.feature_extraction.text
import CountVectorizer
from sklearn.svm import SVC
from sklearn.metrics import
classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
import numpy as np
```

DATAPLAY

Step two: Loading the Dataset

Here, we import the csv file named spam.csv into a pandas dataframe df

- encoding = 'utf-8': specifies the file's encoding, i.e, the way of reading and writing the data from the csv file should be in the widely used 'UTF-8' format
- encoding_erros = 'ignore': Ignores any encoding errors that might occur while reading the file

```
#Load dataset
df = pd.read_csv('spam.csv', encoding =
'utf-8', encoding_errors = 'ignore')
#overview of the dataset
df.head()
```

DATAPLAY

Dataset Overview

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

DATAPLAY

Step three: Preparing the Data

We just want two columns v1 and v2 of the data as the other features dont contribute much information

```
#Checking column names (assuming 'v1' is the label and 'v2' is the message)
df.columns = ['label','message','Unnamed:2','Unnnamed:3','Unnamed:4']
df = df[['label','message']] # only keep the necessary columns
df
df['label'].value_counts(normalize = True) #we want to know the proportion of spam and ham values
```

DATAPLAY

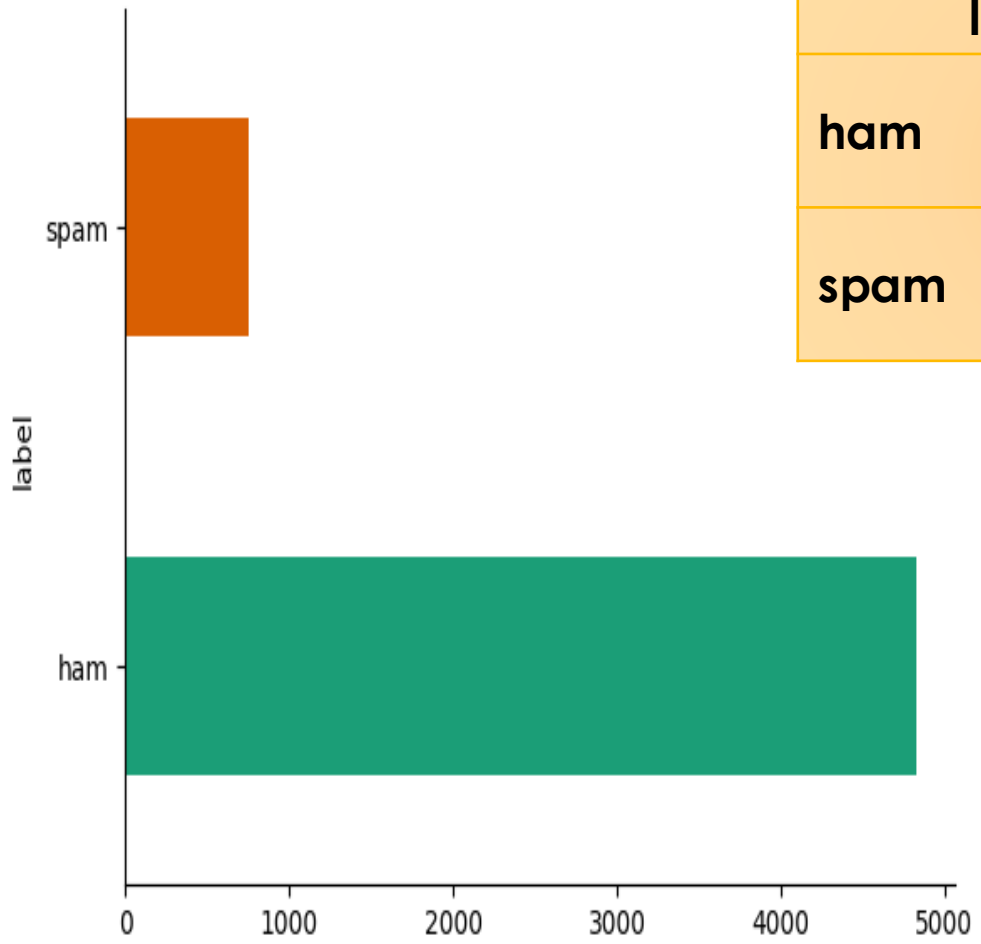
Dataset Required

5572 rows × 2 columns

index	label	message
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives around here though

DATAPLAY

Dataset Distribution



label	proportion
ham	0.865937
spam	0.134063

DATAPLAY

Step four: Splitting the Data into training and test sets

The train_test_split function is used to split the dataset into training and testing sets

- test_size = 0.25: specifies that 25% of the data will be used for training and 75% of the data will be used for testing
- stratify = df['label]: Ensures that the proportion of spam vs ham messages is maintained in both training and test sets for efficient training and prediction

```
#Stratified train-test split
X_train, X_test, y_train, y_test =
train_test_split(df['message'],df['label']
,test_size = 0.25,
stratify=df['label'],random_state = 42)
```

DATAPLAY

Step five: Vectorizing the test data

We vectorize the data so as to convert the words(categorical values) to numbers(continuous values) so as to allow the model to efficiently process the data

1. TfidfVectorizer(stop_words='english',ngram_range=(1,3),max_df=0.2): Initializes a vectorizer, which converts the text into a matrix of token counts based on their importance, ignoring the less relevant words like is,are etc.

2. fit_transform(X_train): learns the vocabulary from the training set and transforms the training text data into a sparse matrix of token counts

3. transform(X_test): Transforms the test text data into the same matrix format, using the vocabulary learnt from the training data

```
#using the tfidf vectorizer
tfidf_vectorizer =
TfidfVectorizer(stop_words='english',ngram_
range=(1,3),max_df=0.2)
X_train_count =
tfidf_vectorizer.fit_transform(X_train)
X_test_count =
tfidf_vectorizer.transform(X_test)
```

DATAPLAY

Step five: Vectorizing the test data continued….

```
X_train_count.todense() #view of the sparse matrix
y_train.value_counts(normalize=True) #checking whether the proportion is maintained
```

label	proportion
ham	0.865997
spam	0.134003

DATAPLAY

Step six: Applying SMOTE to Handle Class Imbalance

Smote(Synthetic Minority Over-Sampling Technique) is used to resample,i.e, feed the similar data again and again for the model to learn effectively. It addresses class imbalance in datasets. The same data is not fed rather an interpolation between minority class examples are fed.

- SMOTE(sampling_strategy='minority',random_state = 42): Creates an instance of SMOTE with a random state for reproducibility
- fit_resample(X_train_count, y_train): Applies SMOTE to the training data:

1. X_train_count = The vectorized text data of the training set
2. y_train = The labels(spam/ham) of the training set
3. X_train_smote, y_train_smote: Outputs the new balanced training set after adding synthetic examples to the minority class

DATAPLAY

Step six: Applying SMOTE to Handle Class Imbalance continued…

```
#Apply SMOTE to the training set

smote =
SMOTE(sampling_strategy='minority',random_s
tate = 42)

X_train_smote, y_train_smote =
smote.fit_resample(X_train_count, y_train)

y_train_smote.value_counts(normalize=True)
#equal proportions of data
```

label	proportion
ham	0.5
spam	0.5

DATAPLAY

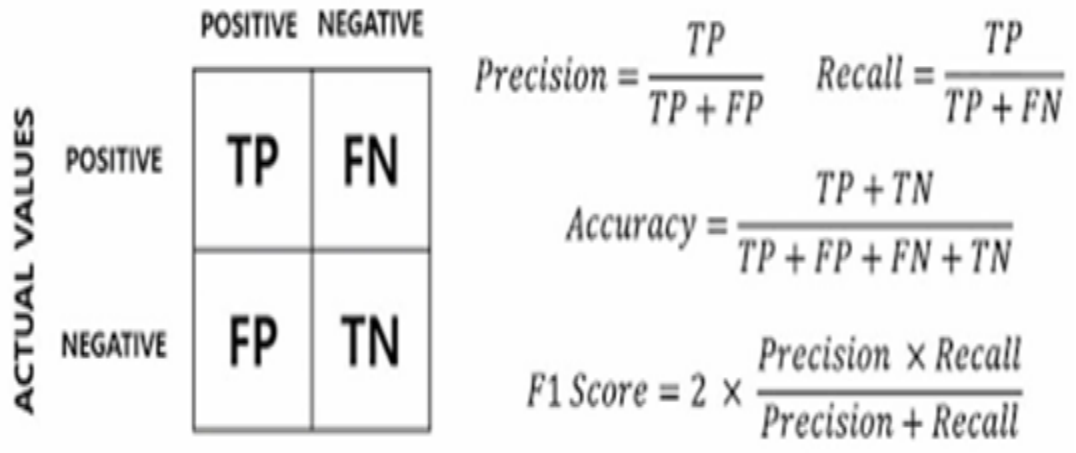
Step seven: Training the SVM Model

- SVC(kernel='linear', random_state=42): initializes an SVM classifier with a linear kernel(linear boundary separating the two categories)
- fit(X_train_smote, y_train_smote): Trains the Support Vector machine using the SMOTE balanced training data

```
#Train the SVM Model
svm_model =
SVC(kernel='linear',random_state = 42)
svm_model.fit(X_train_smote,y_train_smote)
#Making predictions
y_pred = svm_model.predict(X_test_count)
```

DATAPLAY

Step eight: Evaluating the model




```
#Evaluate the model
print("Classification Report:\n",
classification_report(y_test, y_pred,
digits=4))
print("Confusion Matrix:\n",
confusion_matrix(y_test,y_pred))
```

DATAPLAY

Results:

Classification	precision	recall	F1-score	support
ham	0.9901	0.9967	0.9934	1206
spam	0.9777	0.9358	0.9563	187

	precision	Recall	F1-score	support
Accuracy			0.9885	1393
Macro avg	0.9839	0.9663	0.9748	1393
Weighted avg	0.9884	0.9885	0.9884	1393

DATAPLAY

Results:

CONFUSION MATRIX

1202	4
12	175

We can conclude that the weighted F1 score of the model Is 98.84% which is a satisfactory indicator that the model is highly accurate

DATAPLAY

Thank you