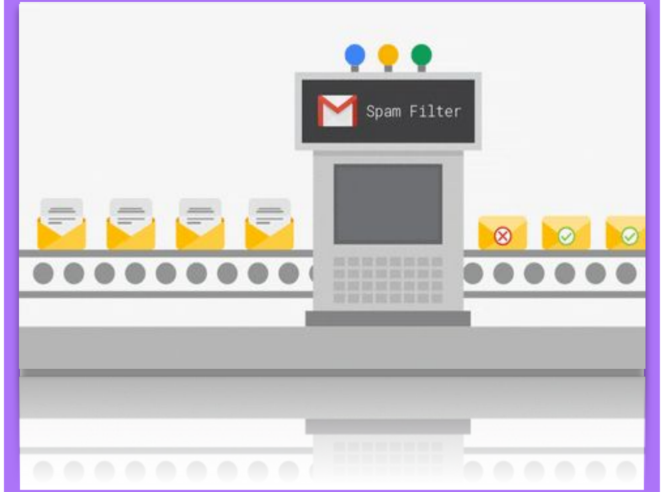
PROJECT 4

Spam or Ham?

Colleen Cobb, Taylor Hill, Gayatri Kotaru,
Stefanie Mendelsohn, Rilee Peebles & Hannah Thelander

Introduction

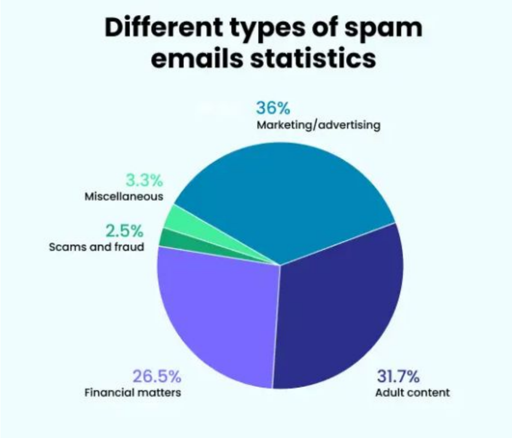


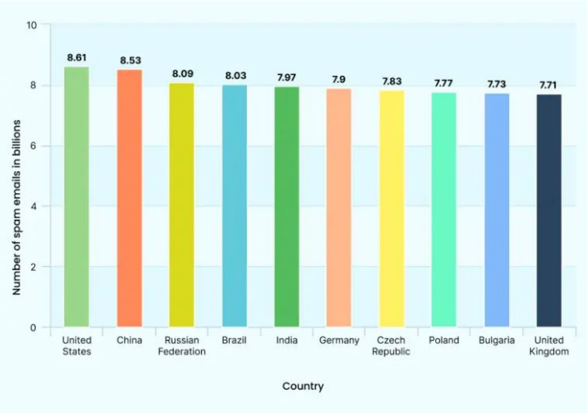
Spam = Unsolicited messages via email or text (SMS)

- Can be malicious
- Can be for tracking purposes

Ham = Directly or indirectly signed up for the email and/or text (SMS)

Why do we care about Spam?





DATA SOURCES

- Email Spam Detection
  - https://www.kaggle.com/code/mfaisalqureshi/email-spam-detection-98-accuracy/data
- Spam Base
  - https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/
- SMS Spam Collection
  - https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset
- Email Spam Classification
  - https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv
- Spam or Not Spam
  - https://www.kaggle.com/datasets/ozlerhakan/spam-or-not-spam-dataset

- ★ Pandas
- ★ Numpy
- ★ Tensor Flow
- ★ Keras
- ★ Pickle

- ★ NLTK
- ★ SKLearn
- ★ Standard Scaler
- ★ Matplotlib

TOOLS:

Used a wealth of software, languages, commands, functions and libraries!

MACHINE LEARNING

LOGISTIC REGRESSION

Supervised machine learning model that is useful in determining shared characteristics among items in a dataset.

NEURAL NETWORK

Machine learning comprised of node layers, one or more hidden layers and an output layer. The nodes connect to each other send data to the other layers.

MACHINE LEARNING

NAIVE BAYES

Supervised learning classifier using Bayes Theorem to predict conditional independence between features and class variables

- ★ Tokenizer
- ★ Hashing
- ★ Stop Words from NLTK

RANDOM FOREST

Supervised machine learning model that is used for classification problems. Similar advantages in relation to logistic regression model.

SEQUENTIAL MODEL

Base model

```
36/36 - 0s - loss: 0.1651 - accuracy: 0.9435 - 246ms/epoch - 7ms/step
Loss: 0.16513986885547638, Accuracy: 0.943527340888977
```

Applied two different activation functions:

★ Relu
★ Tanh

```
36/36 - 0s - loss: 0.1608 - accuracy: 0.9496 - 190ms/epoch - 5ms/step
Loss: 0.16081812977790833, Accuracy: 0.9496090412139893
```

```python
# First hidden layer
nn_model1.add(tf.keras.layers.Dense(units=10, activation='relu', input_dim=input_features)

# Second hidden layer
nn_model1.add(tf.keras.layers.Dense(units=10, activation='relu'))

# Output layer

nn_model1.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
# Check the structure of the model
nn_model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                580

 dense_1 (Dense)             (None, 10)                110

 dense_2 (Dense)             (None, 1)                 11

=================================================================
Total params: 701
Trainable params: 701
Non-trainable params: 0
_________________________________________________________________
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 30)                1740

 dense_4 (Dense)             (None, 20)                620

 dense_5 (Dense)             (None, 1)                 21

=================================================================
Total params: 2,381
Trainable params: 2,381
Non-trainable params: 0
_________________________________________________________________
```

LOGISTIC REGRESSION

Results

Training Data Score: 0.9220289855072463
Testing Data Score: 0.9339704604691572

- Useful in classification
  - Recurrence of words
  - Count of words in a message

- Defines the dependency of dependent variables to independent variables

DATASET ANALYSIS

Email Type	Mean Word Count	Median Word Count	Word Count Variance	Word Count Std. Dev.	Word Count Std. Err.
ham	14.200822	11	130.519444	11.424511	0.164471

Email Type	Mean Word Count	Median Word Count	Word Count Variance	Word Count Std. Dev.	Word Count Std. Err.
spam	23.851406	25	33.778158	5.811898	0.212646

LOGISTIC REGRESSION

```python
# Split the data into X_train, X_test, y_train, y_test

spambase_df.dropna(inplace=True)
X=spambase_df.drop("class",axis=1).values
y=spambase_df["class"].values
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
# Train a Logistic Regression model and print the model score
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier
```

```
8]: LogisticRegression()
```

```python
classifier.fit(X_train, y_train)
```

```
Training Data Score: 0.9220289855072463
Testing Data Score: 0.9339704604691572
```

RANDOM FOREST

RESULTS

★ Precision : 0.992
★ Recall : 0.698
★ Fscore : 0.82
★ Accuracy: 0.959

ADVANTAGES

★ Perform both regression and classification
★ Handle large datasets
★ Higher level accuracy

DISADVANTAGES

★ More resources are required for computation
★ Consumes more time than a decision tree

RANDOM FOREST

```
#Now Random Forest Model
rf = RandomForestClassifier(n_estimators=100,max_depth=None,n_jobs=-1)
rf_model = rf.fit(X_train,y_train)
```

X_test was:
3901x8357 sparse matrix of type '<class 'numpy.int64'>'
with 30356 stored elements in Compressed Sparse Row format>

```
y_pred=rf_model.predict(X_test)
precision,recall,fscore,support =score(y_test,y_pred,pos_label=1, average ='binary')
print('Precision : {} / Recall : {} / fscore : {} / Accuracy: {}'.format(round(precision,3),round(recall,3),round(fscore,3),round((y_pred==y_test).sum()/len(y_test),3)))
```

NAIVE BAYES

RESULTS

```
Accuracy of model at predicting spam was: 0.956618
```

```
alpha              9.570010
Train Accuracy     0.979054
Test Accuracy      0.970008
Test Recall        0.773694
Test Precision     1.000000
Name: 87, dtype: float64
```

ADVANTAGES

★ Requires small amount of training data
★ Simple & fast
★ Suited for categorical data

DISADVANTAGES

★ Assumes all attributes are independent
★ If class is not in dataset, the return is 0

NAIVE BAYES

```
#Now Naive Bayes Model
f = feature_extraction.text.CountVectorizer(stop_words = 'english')
X = f.fit_transform(data["message"])
np.shape(X)
```

```
(5572, 8357)
```

alpha	Train Accuracy	Test Accuracy	Test Recall	Test Precision
0.00001	0.999402	0.961548	0.920696	0.813675
0.11001	0.998205	0.965137	0.955513	0.813839
0.22001	0.998205	0.966163	0.951644	0.821369
0.33001	0.998205	0.968470	0.949710	0.835034
0.44001	0.997008	0.970264	0.945841	0.847487

```
# Classifying spam and not spam msgs as 1 and 0

data["label"]=data["label"].map({'spam':1,'ham':0})
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, data['label'], test_size=0.70, random_state=42)
```

NAIVE BAYES

Preprocessing

```
pos_neg_to_num = StringIndexer(inputCol='Category',outputCol='label')
tokenizer = Tokenizer(inputCol="Message", outputCol="token_text")
stopremove = StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
hashingTF = HashingTF(inputCol="stop_tokens", outputCol='hash_token')
idf = IDF(inputCol='hash_token', outputCol='idf_token')
```

Category	Message
ham	Go until jurong p...
ham	Ok lar... Joking ...
spam	Free entry in 2 a...
ham	U dun say so earl...
ham	Nah I don't think...
spam	FreeMsg Hey there...
ham	Even my brother i...
ham	As per your reque...
spam	WINNER!! As a val...
spam	Had your mobile 1...
ham	I'm gonna be home...
spam	SIX chances to wi...
spam	URGENT! You have ...
ham	I've been searchi...
ham	I HAVE A DATE ON ...
spam	XXXMobileMovieClu...
ham	Oh k...i'm watchi...
ham	Eh u remember how...
ham	Fine if that’s th...
spam	England v Macedon...

Before & After Preprocess

label	features
0.0	(262145,[38555,52...
0.0	(262145,[51783,15...
1.0	(262145,[9443,122...
0.0	(262145,[2306,332...
0.0	(262145,[25964,64...
1.0	(262145,[19835,23...
0.0	(262145,[103497,1...
0.0	(262145,[12650,27...
1.0	(262145,[4314,232...
1.0	(262145,[1546,219...
0.0	(262145,[12716,17...
1.0	(262145,[7415,161...
1.0	(262145,[23209,35...
0.0	(262145,[15585,41...
0.0	(262145,[39504,13...
1.0	(262145,[26364,44...
0.0	(262145,[18184,22...
0.0	(262145,[12524,16...
0.0	(262145,[37132,51...
1.0	(262145,[16168,29...

Category	Message	length	label	token_text	stop_tokens	rawPrediction	probability	prediction
ham	"7 wonders in My ...	155	0.0	["7, wonders, in,...	["7, wonders, wor...	[-1576.1658865825...	[1.0,1.7390252805...	0.0
ham	"7 wonders in My ...	155	0.0	["7, wonders, in,...	["7, wonders, wor...	[-1576.1658865825...	[1.0,1.7390252805...	0.0
ham	"A swt thought: "...	160	0.0	["a, swt, thought...	["a, swt, thought...	[-1582.0029941322...	[1.0,1.8729839925...	0.0
ham	"An Amazing Quote...	144	0.0	["an, amazing, qu...	["an, amazing, qu...	[-1379.4067172087...	[1.0,1.1282744219...	0.0
ham	"And that is the ...	383	0.0	["and, that, is, ...	["and, problem., ...	[-1970.9755119910...	[1.0,5.3057835434...	0.0

hash_token	idf_token	features
(262144,[59381,60...	(262144,[59381,60...	(262145,[59381,60...
(262144,[59381,60...	(262144,[59381,60...	(262145,[59381,60...
(262144,[8804,380...	(262144,[8804,380...	(262145,[8804,380...
(262144,[38640,44...	(262144,[38640,44...	(262145,[38640,44...
(262144,[21823,35...	(262144,[21823,35...	(262145,[21823,35...

NEURAL NETWORK

RESULTS

- ★ Accuracy: 0.9887
- ★ Loss: 0.0679

ADVANTAGES

- ★ Highly Accurate Model
- ★ New data can be easily pre-processed for model prediction

DISADVANTAGES

- ★ Requires a lot of computational resources

NEURAL NETWORK

Preprocessing

```
corpus=[]

for i in range(len(df)):
    # removing all non-alphanumeric characters
    message=re.sub('[^a-zA-Z0-9]',' ',df['Text'][i])
    # converting the message to lowercase
    message=message.lower()
    message = message.split()


    corpus.append(message)
```

```
[ ] cv=CountVectorizer(max_features=2500,ngram_range=(1,3))
    cvfit=cv.fit(corpus)
    X = cvfit.transform(corpus).toarray()
    y=df['Spam']
```

	Text	Spam
0	Subject: naturally irresistible your corporate...	1
1	Subject: the stock trading gunslinger fanny i...	1
2	Subject: unbelievable new homes made easy im ...	1
3	Subject: 4 color printing special request add...	1
4	Subject: do not have money , get software cds ...	1

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [4, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

NEURAL NETWORK

Saving the model

```
[ ]  import pickle
     with open("pickle2.pkl","wb") as f:
         pickle.dump(cvfit, f)
```

```
[ ]  nn.save("success1.h5")
```

```
ham_message = "Subject: meeting with vince set for 1 pm monday , april 17  maureen , thank you very much
```

```
spam_message = "Subject: mail receipt  thank you for your mail regarding our site . we will reply as soon as possible
```

```
[ ]  nn.predict(spam_x)

     1/1 [==============================] - 0s 17ms/step
     array([[3.544541e-05],
            [9.995594e-01]], dtype=float32)
```

NEURAL NETWORK

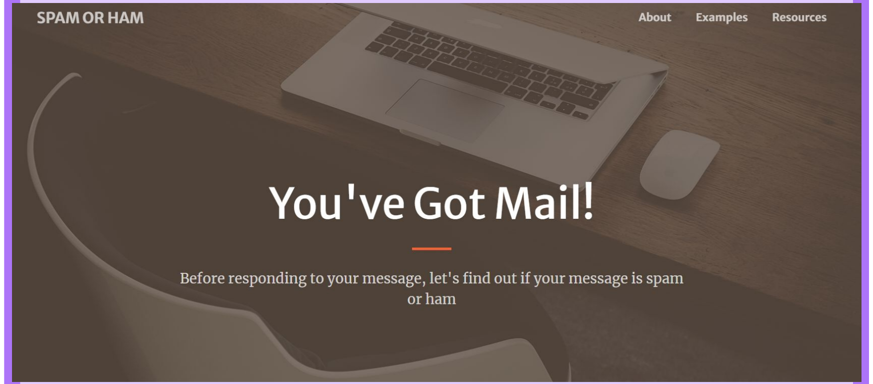
Calling the model

```
def preprocess(text):
    corpus = []
    message=re.sub('[^a-zA-Z0-9]',' ',text)
    # converting the message to lowercase
    message=message.lower()
    # message = np.asarray(message)
    corpus.append(message)
    with open('pickle2.pkl', 'rb') as pickle_file:
        content = pickle.load(pickle_file)
        words = content.transform(corpus)

    return(words)
```

```
model1 = keras.models.load_model("../h5/success1.h5")
model1.compile()

app = Flask(__name__)

@app.route("/api/classify", methods = ["POST"])
def classify():
    if request.method == 'POST':
        text = request.form["text"]
        print(text)

        processed_text = preprocess(text)

        result1= model1.predict(processed_text)
        if result1 > .65:
            classification = 'Spam'
        else:
            classification = 'Ham'
        if classification =='Spam':
            response = random.choice(spam_response)
        if classification == 'Ham':
            response = random.choice(ham_response)

        return render_template("spam.html", result={
            "text": text,
            "classification": classification,
            "response": response
            })
```

01

Spam or Ham Dashboard



Dashboard

HTML

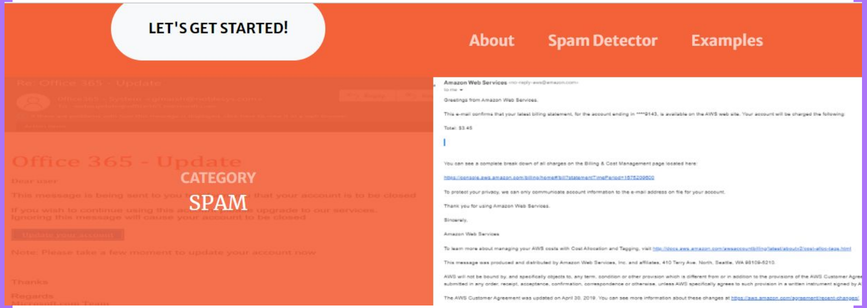
```
v templates
  <> index.html
  <> spam.html
```

CSS

```
    color: #fff;
    background: rgba(244, 98, 58, 0.9);
    transition: opacity 0.25s ease;
    text-align: center;
}
#portfolio .container-fluid .portfolio-box .port
    font-family: "Merriweather Sans", -apple-syste
    font-size: 0.85rem;
    font-weight: 600;
    text-transform: uppercase;
}
#portfolio .container-fluid .portfolio-box .port
    font-size: 1.2rem;
}
#portfolio .container-fluid .portfolio-box:hover
    opacity: 1;
}
```

JavaScript

```
v js
  JS app.js
  JS scripts.js
```



3 out of 6

Were examples of Spam. The others were examples of Ham.

RESOURCES

- https://www.mailmodo.com/guides/email-spam-statistics/
- https://startbootstrap.com/
- https://www.nltk.org/search.html?q=stopwords&check_keywords=yes&area=default
- https://towardsdatascience.com/email-spam-detection-1-2-b0e06a5c0472

THANK YOU
FOR
LISTENING !