PROJECT DEFINITION

Domain Background:

Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) that aims to interact with human language. The field began in the 1940s, shortly after the second world war, when people began to wonder if it was possible to build an automatic translation machine. Noah Chomsky was among the first researchers in the field and his work in the 1957 book “Syntactic Structures” concluded that for computers to understand language, the structure of language itself had to be changed.

The subsequent history of NLP can be divided in three phases, **(I)** symbolic NLP, **(II)** statistical NLP and **(iii)** Neural NLP. Symbolic NLP uses very high-level representations of text and applies logical operators on top of it. The great advantage of symbolic NLP is that it is very human-readable, so when a particular application demands explainability, it might still be an option. Statistical NLP thrived after the 1980s, it applied statistical machine learning models to a vector representation of the text. An example of such an approach would be using Bag of Words vectors and a Naive Bayes classifier. Last but not least, Neural NLP employs the use of deep neural networks to vector representations of text. The vector representation for neural NLP could be Bag of Words, TF-IDF or distributional semantics embeddings. Distributional semantics leverage the context and it has the advantage of similar words having similar vectors, there are several approaches to generate such vectors such as, CBOW, Word2Vec and more recently embeddings generated by transformers as introduced in the “attention is all you need” paper.

A variety of tasks can be performed using NLP, text classification, machine translation, question answering, etc.. NLP models are most valuable when the amount of text to analyze is enormous and the workforce limited or expensive. A well tuned machine learning model can classify millions of documents, taking no breaks and often just as accurately as the human worker. For example, a company might want to classify comments on their products as positive or negative (this particular case of text classification is called sentiment analysis), this might help the company to take action and improve quality where it is needed quicker, a human workforce could not possibly read every single comment in every single marketplace within a reasonable timeframe.

Problem Statement:

Since the dawn of the internet, scammers have been using it for their own advantage. A popular type of scam is the phishing scam, which contacts the user with a tailored message in order to acquire sensitive data. When phishing is performed via SMS, it’s called SMShing.

Task:

Sentence Classification is the task of receiving a sentence as input and predicting a class as output. The goal of this project is to develop a sentence classifier that receives a SMS (sentence) and outputs the probability of it being a SMShing. The dataset originally has three classes, but for the purpose of this project, two of the classes will be merged.

Dataset:
Source: SMS PHISHING DATASET FOR MACHINE LEARNING AND PATTERN RECOGNITION - Mendeley Data

Description: The dataset contains 5971 labeled text messages as Legitimate (Ham) or Spam or Smishing. The dataset contains the textual data as well as features denoting the presence of url, email or phone in text. The columns of the dataset are described below. For the purpose of this project, the ham and spam category will be merged into a new legitimate category. Also the URL, EMAIL and PHONE columns will be kept only for analysis, the idea for the model is to work with text only.

Columns:

- LABEL: categorical with three possible values
  - Ham: Legitimate text
  - Spam: text with advertising or similar content\
  - Smishing: text with the intent to scam the receiver
- TEXT: textual content of the sms
- URL: Categorical with two possible values
  - Yes: the text contains a url
  - No: the text does not contains a url
- EMAIL: Categorical with two possible values
  - Yes: the text contains an email
  - No: the text does not contains an email
- PHONE: Categorical with two possible values
  - Yes: the text contains a phone number
  - No: the text does not contains a phone number

Evaluation Metrics:
Three metrics will be used to evaluate the model, (I) Recall, (II) Precision and (III) F1-Score. The main focus will be high recall, it is important to not let any fraudulent messages reach the end user even if at the cost of blocking a few legitimate ones. Precision will be the secondary goal, as it is also important to not throw out too many legitimate messages. And F1-Score will be used to give an idea of precision and recall together.

1. Recall: The fraction of relevant documents that is retrieved by the model.
2. Precision: The fraction of retrieved documents that are relevant.
3. F1-Score: Harmonic mean between precision and recall.

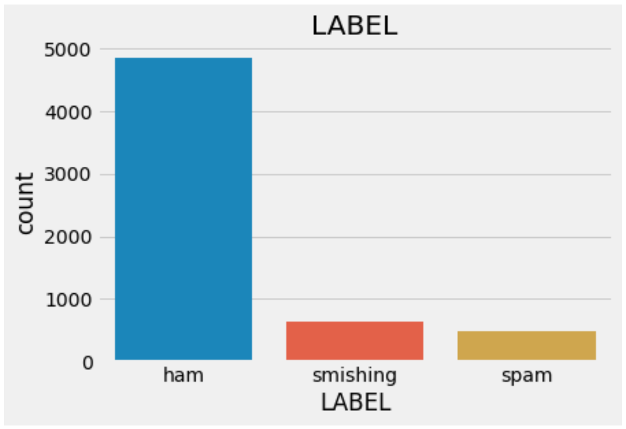
PROJECT ANALYSIS

Basic Exploration:

- Entries: 5971
- Non-null entries: 5971
- Unique values per column:
  - LABEL 3
  - TEXT 5949
  - URL 2
  - EMAIL 2
  - PHONE 2

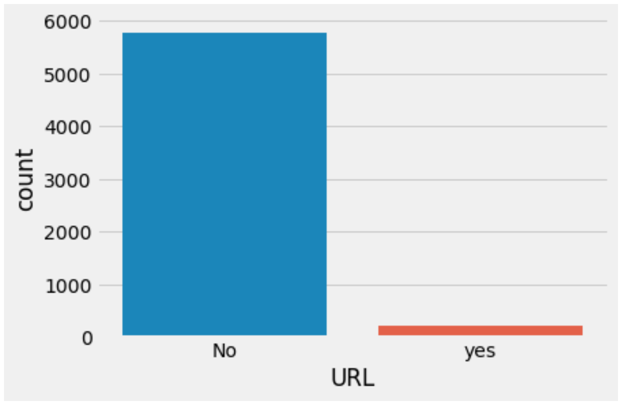
Exploratory data analysis:

Label distribution:

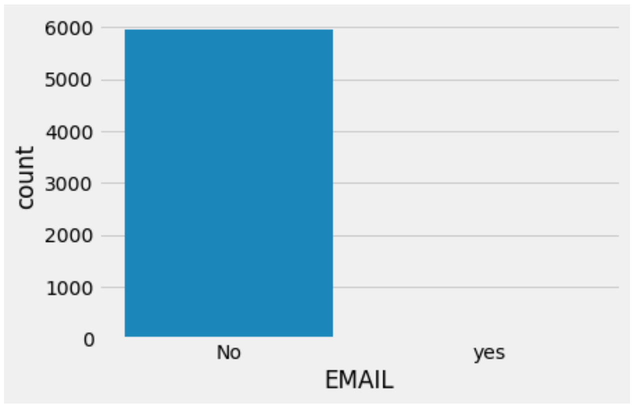


Features distributions:

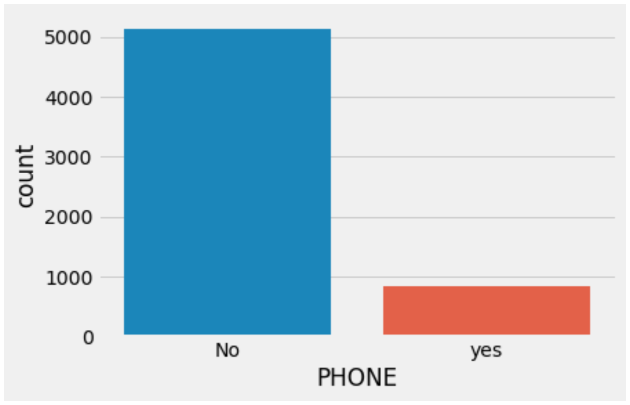
- URL



- EMAIL

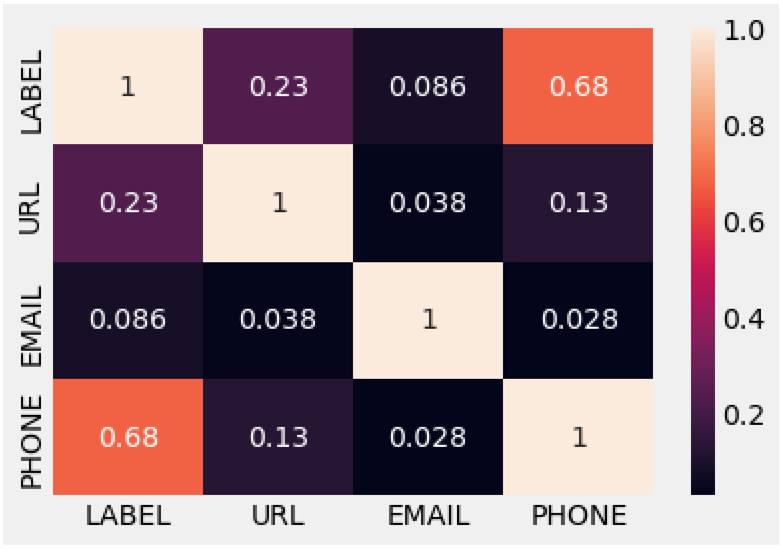


- TELEPHONE



Correlation matrix:

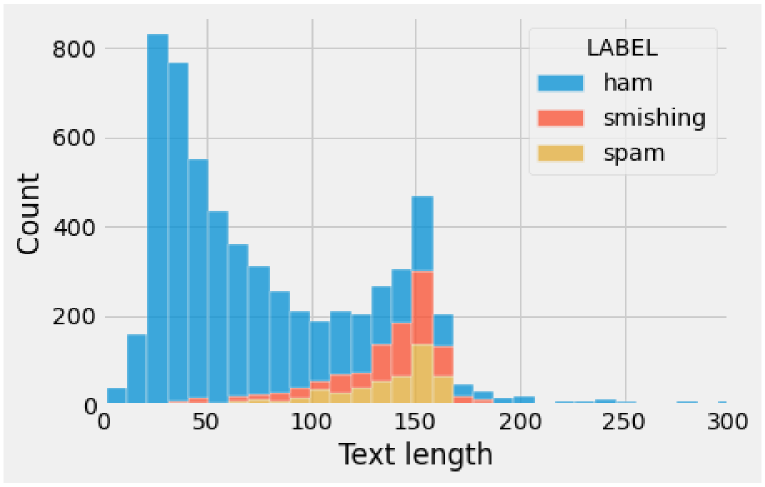
For the purpose of this analysis the categorical features and the label will be binarized. The features will have a value of 1 for yes and 0 for no. The label will have a value of 1 for smishing and 0 otherwise.



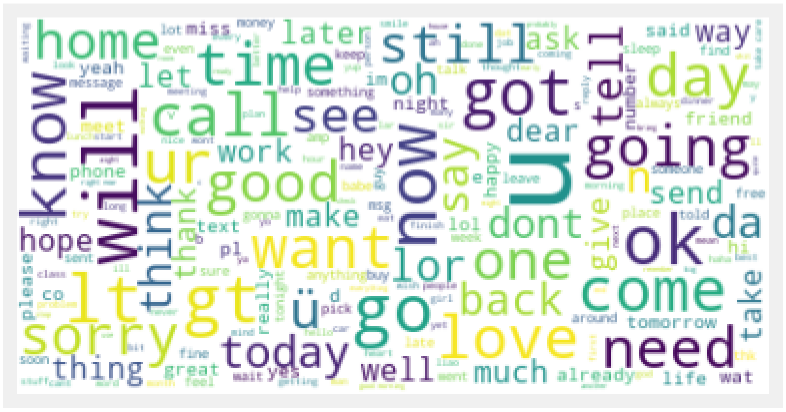
	LABEL	URL	EMAIL	PHONE
LABEL	1	0.23	0.086	0.68
URL	0.23	1	0.038	0.13
EMAIL	0.086	0.038	1	0.028
PHONE	0.68	0.13	0.028	1

Text body analysis:

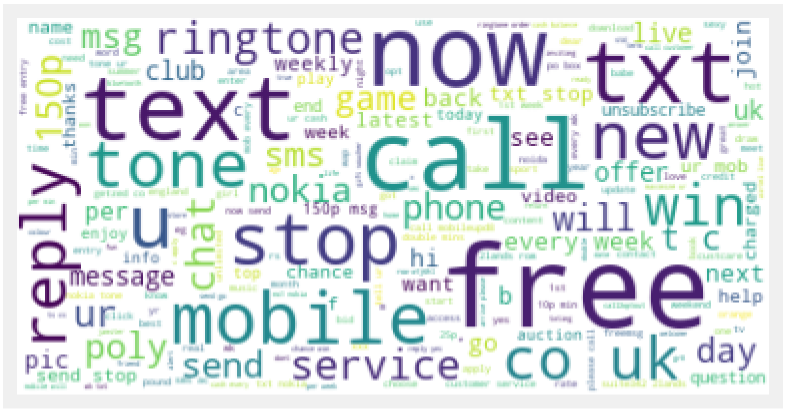
- Text len histogram



- WordClouds
    - Ham



    - Spam



- SMShing

From the word clouds we can see that the Ham class is quite different from the others, however since we will merge Ham and Spam into one class and Spam and SMShing have a relevant intersection, it may result in a challenging problem to the model.

Algorithms and Techniques:

This is a typical supervised classification problem, so we will use classification algorithms to handle it. We will use models from two libraries, SciKit-Learn and HuggingFace Transformers as listed below:

- SciKit-Learn:
    - Vectorizers:
        - Count Vectorizer
        - TFIDF Vectorizer
    - Models:
        - SGD Classifier
        - Naive Bayes
        - Logistic Regression
        - Random Forest Classifier

- HuggingFace Transformers
    - Tokenizers/Models
        - Bert base uncased
        - DistilBert base uncased

Benchmark model:

I could not find benchmarks for SMShing specifically, however Spam detection is a quite similar task and PapersWithCode sets its benchmark at a average F1 of 0.8553 by the 2022 paper “Traditional and context-specific spam detection in low resource settings”.

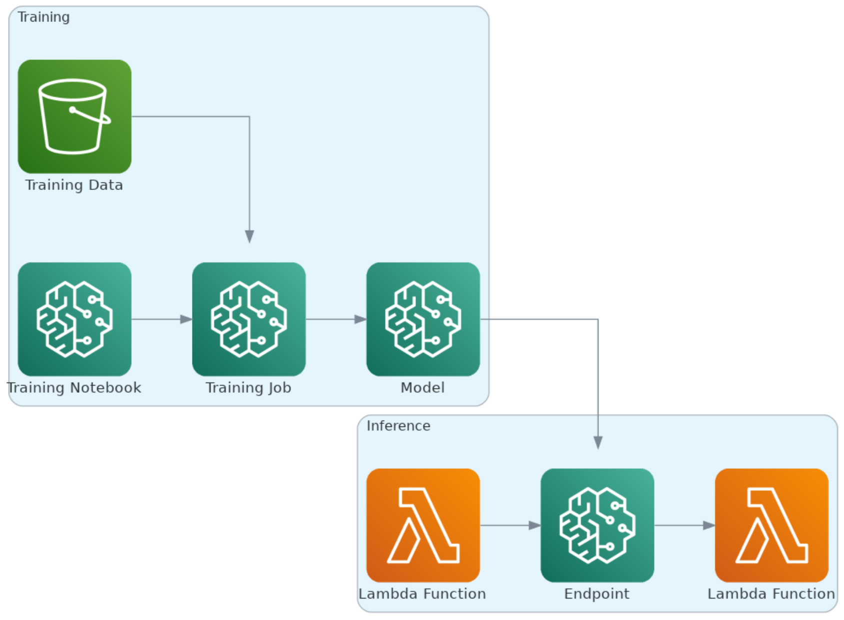
METHODOLOGY

Data Preprocessing:

1. Lower case
2. Substitute phone number for <PHONE>
3. Substitute emails for <EMAIL>
4. Substitute urls for <URL>
5. Substitute numbers for <NUM>

Implementation:

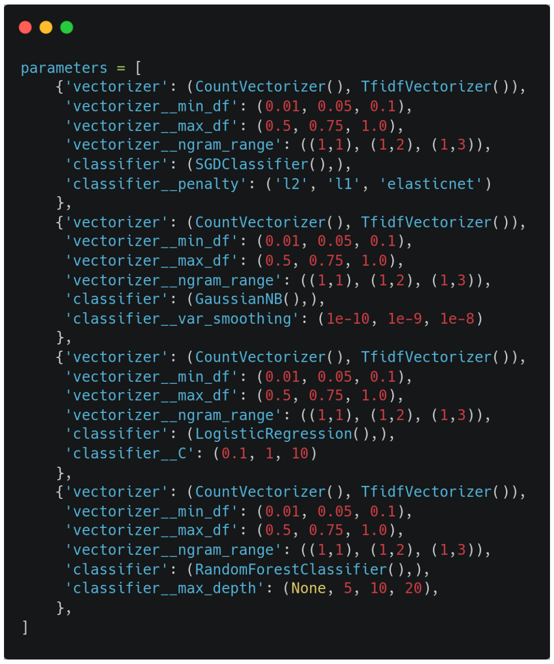
Data exploration, model selection and fine tuning will be done locally on jupyter notebooks. Training of the selected model, endpoint deployment and serving will be done within the AWS ecosystem. During the training phase, a SageMaker notebook will initialize a training job which will consume the training data from S3 and generate a model which is deployed to an endpoint. In the inference phase, a lambda function will access the SageMaker endpoint which will return the result to the lambda function and finally to the user.



Architecture

Refinement:

A grid search will be performed over models and hyperparameter over the scikit-learn models using the following parameter grid:



```python
parameters = [
    {'vectorizer': (CountVectorizer(), TfidfVectorizer()),
     'vectorizer__min_df': (0.01, 0.05, 0.1),
     'vectorizer__max_df': (0.5, 0.75, 1.0),
     'vectorizer__ngram_range': ((1,1), (1,2), (1,3)),
     'classifier': (SGDClassifier(),),
     'classifier__penalty': ('l2', 'l1', 'elasticnet')
    },
    {'vectorizer': (CountVectorizer(), TfidfVectorizer()),
     'vectorizer__min_df': (0.01, 0.05, 0.1),
     'vectorizer__max_df': (0.5, 0.75, 1.0),
     'vectorizer__ngram_range': ((1,1), (1,2), (1,3)),
     'classifier': (GaussianNB(),),
     'classifier__var_smoothing': (1e-10, 1e-9, 1e-8)
    },
    {'vectorizer': (CountVectorizer(), TfidfVectorizer()),
     'vectorizer__min_df': (0.01, 0.05, 0.1),
     'vectorizer__max_df': (0.5, 0.75, 1.0),
     'vectorizer__ngram_range': ((1,1), (1,2), (1,3)),
     'classifier': (LogisticRegression(),),
     'classifier__C': (0.1, 1, 10)
    },
    {'vectorizer': (CountVectorizer(), TfidfVectorizer()),
     'vectorizer__min_df': (0.01, 0.05, 0.1),
     'vectorizer__max_df': (0.5, 0.75, 1.0),
     'vectorizer__ngram_range': ((1,1), (1,2), (1,3)),
     'classifier': (RandomForestClassifier(),),
     'classifier__max_depth': (None, 5, 10, 20),
    },
]
```

The Bert and Distillbert model will also be trained and compared to the best estimator from the grid search. For consistency in both training, a random state parameter will be set to the train/test split. Results are shown bellow:

Best Scikit-Learn Model (CountVectorizer + Random Forest)

- Precision: 0.88
- Recall: 0.77
- F1-score: 0.825

BERT (Epoch 3)

- Precision: 0.85
- Recall: 0.84
- F1-score: 0.85

DistilBERT (Epoch 4)

- Precision: 0.84
- Recall: 0.84
- F1-score: 0.84

The best model was the BERT model, however only by a slight margin over DistilBERT, so for performance we will proceed with DistilBERT.

RESULTS

Model Evaluation and Validation:

The final model was evaluated using holdout validation with a test set size of 20%. The metrics are shown below:

Accuracy: 0.9615062761506277
Precision: 0.7928571428571428
Recall: 0.8671875,
F1 Score: 0.8283582089552239,
Samples per Second: 1147.138

Justification:

While our final score is lower than our benchmark, it is pretty close and also it is important to remember the tasks are not quite the same. A fully automated system might not be completely feasible at this point, however a human in the loop system could work just fine. A human would need to inspect only the messages tagged as fraudulent and with time and the growth of the dataset performance might improve and the human may be excluded.