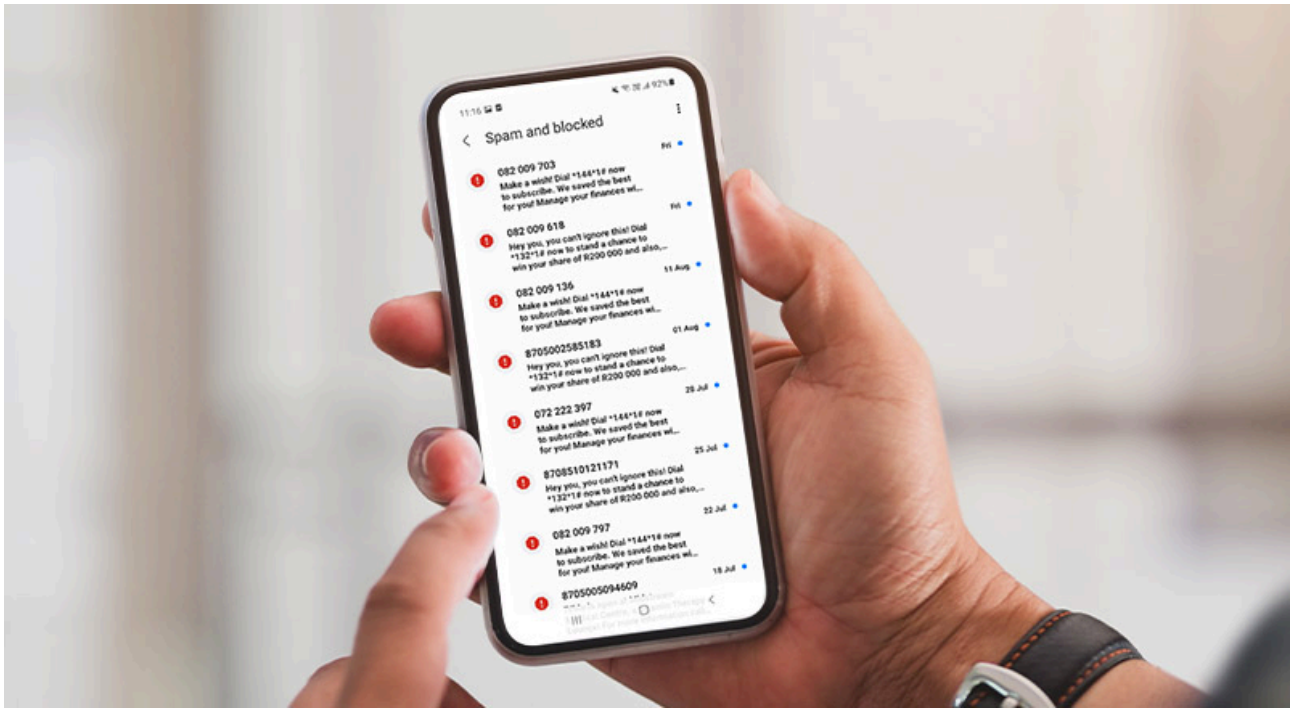
SMS Spam Detection



Domain-Specific Area: SMS Spam Detection Classification

As the mobile technology is growing rapidily and smartphones are everywhere and in everyone's hands, SMS is considered one of the most used ways for people to communicate. Recent reports mention that about 80 percent of mobile phones users use SMS everyday which is equivallent to 3.5 billion people all over the world. Its easieness that allow people to open and read messages quickly and the same time they arrive is the reason of the popularity of SMS Usage along with the fact that its a very cheap communication way which makes SMS crucial for everyday personal and business use.

SMS can be very useful, however, it has a dark side as it opened a huge gate for cybercriminals and spammers. A huge number of spam messages are daily sent directly to everyone's phones which resulted in a major cybersecurity problem the world is facing in the meantime. SMS spam goes straightforward to your phone and often gets bypasses normal security Unlike spam emails which people are used to spotting and filtering. The

risks today are more than just annoying as phishing attacks also called smishing, scams, and malware links can be a huge threat to our privacy, money, and data.

Spammers are currently using new techniques such as changing letters, misspelling words, and making messages that bypass basic keyword filters. As a result, these rule based filters are no longer effective. Better Natural Language Processing techniques are therefore required in order to understand and categorize these SMS messages according to their context and meaning.

from basic Bayesian classifiers to modern deep learning models , spam detection has changed a lot. In this project, we look at both old school statistical methods and new transformer models. Our goal is to find out if using deep learning is really needed for short messages like SMS or if the simpler methods do the job.

citations:

Almeida, T.A., Gómez Hidalgo, J.M., Yamakami, A. (2011). Contributions to the Study of SMS Spam Filtering: New Collection and Results Devlin, J., Chang, M.W., Lee, K. and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Objectives

The project main aim is to find out if the transformer based model like DistilBERT is crucial for short text classification tasks like SMS spam classfication or the statistical models would perform better. Through an experimental comparison, i will examine the trade offs between model complexity, training efficiency, and classification performance.

My Primary Objective is to Implement and Compare Two Classification Models. i will design and evaluate two very different approaches to spam detection which are a traditional statistical pipeline using TF-IDF vectorization with Multinomial Naive Bayes, and a deep learning approach based on fine tuned DistilBERT. This comparison shows the differences between frequency based methods and contextual embedding models when applied to short text classification. Another aim can to examine the text Representation Effectiveness. The statistical approach uses TF-IDF features which measure word importance based on how often terms appear. this method is expected to perform well for spam messages with the most common clear trigger words such as free, winner and urgent but struggle with more complex spam messages. however, the embedding based approach captures contextual meaning and may detect spam through more picky language patterns. In deployment environments where a lot of messages

need to be processed fast and effectively, certain factors like training time, speed, and computational resource usage are crucial.Thus, these real-world factors are also assessed in this project.

While transformer models are very powerful, we still don't know if they are the best choice for short messages like SMS. The results here will give clear, practical advice for choosing between high accuracy and fast efficient models.

The Statistical Approach: TF-IDF with Multinomial Naive Bayes

I will use a common used method which acts as strong baseline in text classification which is TF-IDF vectorization with a Multinomial Naive Bayes classifier for the statistical apporoach.

The Embedding-Based Approach: Fine tuned DistilBERT

for the Embedding based approach , I will fine tune a DistilBERT model for sequence classification. DistilBERT has most of BERT's language understanding ability while being faster and smaller in size.

Dataset description

i chose UCI SMS Spam Collection for this project which is a public set of SMS labeled messages collected for spam research. The dataset is available on UCI Machine Learning Repository and Kaggle. This dataset includes alot of messages collected from sources such as the Grumbletext website and the NUS SMS Corpus. These sources were selected to have a huge range of language styles, message lengths, and spam strategies from different geographic regions which makes the dataset reflect real world conditions.

Regarding the Size and Structure of the dataset, The file is 467 KB and the dataset contains about 5572 messages. It is a binary classification dataset with two columns. Label column which is whether it is "ham" or "spam" and Text column which is The raw content of the SMS message

this dataset is suitable for both traditional machine learning models and modern deep learning approaches because of its simple and well defined structure which makes It straightforward for preprocessing, feature extraction, and model comparison without requiring complex data transformations.

The dataset is imbalanced containing approximately 87% ham and 13% spam messages with 4825 ham messages and 747 spam ones. This imbalance represents real world

scenarios but requires careful handling during evaluation.

Evaluation Methodology

evaluating the performance of both models the Statistical Naive Bayes and Deep Learning BERT would made by these metrics.

Precision

which is used to measure the proportion of predicted positives out of the actual positives.

Its formula is:

$$\text{Precision} = \frac{TP}{TP + FP}$$

it would indicate the ratio of correctly predicted spam messages to all messages predicted as spam correctly.

Recall

which is used to measure the proportion of correctly identified true positive predictions out of all actual positives Its formula is:

$$\text{Recall} = \frac{TP}{TP + FN}$$

it would be used to indicate the ratio of correctly predicted spam messages to all actual spam messages in the dataset.

Confusion Matrix

I will also use the Confusion Matrix to compare the results as it visualizes the performance of the algorithm by categorizing predictions into True Positives ,False Positives ,True Negatives , False Negatives.

Accuracy

which measures the ratio of correctly predicted observations to the total observations.Its formula is:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

keeping in mind that the accuracy is not always the best metric for imbalanced datasets like this one because a model can get a high accuracy by just predicting the majority class and still miss most spam messages.

F1-Score

which is the harmonic mean of Precision and Recall. Its formula is:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

This is the primary metric for comparison. Since the dataset is imbalanced, a model could achieve high accuracy by ignoring all spam but the F1-Score penalizes this behavior and ensures the model is actually effective at its job.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk as nltk
```

```
SMS_SH_df = pd.read_csv("SMSSpamCollection.csv", sep="\t", header=None)
SMS_SH_df.columns = ['label', 'body_text']

print(f"Dataset shape: {SMS_SH_df.shape}")
print(SMS_SH_df.label.value_counts())
```

```
Dataset shape: (5572, 2)
label
ham     4825
spam     747
Name: count, dtype: int64
```

```
# class distribution
SMS_SH_df.label.value_counts(normalize=True).plot.pie()
plt.show()
```

EDA

```
# checking some examples of spam vs ham
print("SPAM examples:")
print(SMS_SH_df[SMS_SH_df['label']=='spam']['body_text'].head(3).values)
```

```
print("HAM examples:")
print(SMS_SH_df[SMS_SH_df['label']=='ham']['body_text'].head(3).values)
```

```
SPAM examples:
["Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to
 "FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like so
 'WINNER!! As a valued network customer you have been selected to receivea £900
HAM examples:
['Go until jurong point, crazy.. Available only in bugis n great world la e buff
 'Ok lar... Joking wif u oni...'
 'U dun say so early hor... U c already then say...']
```

```
# Checks message length
SMS_SH_df['msg_length'] = SMS_SH_df['body_text'].apply(len)
print(SMS_SH_df.groupby('label')['msg_length'].mean())

# spam is longer
```

```
label
ham      71.482487
spam    138.670683
Name: msg_length, dtype: float64
```

```
# length distribution plot
SMS_SH_df[SMS_SH_df['label']=='ham']['msg_length'].hist(bins=50, alpha=0.7, lab
SMS_SH_df[SMS_SH_df['label']=='spam']['msg_length'].hist(bins=50, alpha=0.7, la
plt.xlabel('Message Length')
plt.legend()
plt.show()
```

```
# to Create a copy of the dataframe for preprocessing
spam_data = SMS_SH_df.copy()
```

Preprocessing

For Naive Bayes, I'm doin lowercasing, removing punctuation, dropping stopwords, and applying stemming but for BERT, I will skip this cleaning and use the raw text instead as it was pre trained and it acctually benefits from seeing the original text with punctuation and capitalization

```
import string
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

def clean_text_for_tfidf(text):

   # Convert to lowercase
    text = text.lower()

    # Remove punctuation
    text = "".join([char for char in text if char not in string.punctuation])

    # Tokenize
    tokens = word_tokenize(text)

    # to Remove stopwords and apply stemming
    # to filter out single character tokens that are just noise
    cleaned_tokens = [
        stemmer.stem(word)
        for word in tokens
        if word not in stop_words and len(word) > 1
    ]

    return " ".join(cleaned_tokens)



spam_data['cleaned_text'] = spam_data['body_text'].apply(clean_text_for_tfidf)
```

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.dummy import DummyClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_s

# Encode labels
spam_data['label_encoded'] = spam_data['label'].map({'ham': 0, 'spam': 1})

# Split data into training (80%) and testing (20%) sets
# Using stratify to maintain class distribution in both sets because of the imb
# random state equal to 42 to ensure reproducibility of results
X_train, X_test, y_train, y_test = train_test_split( spam_data['cleaned_text'],

print(f"Training set size: {len(X_train)} messages")
print(f"Test set size: {len(X_test)} messages")
```

```
Training set size: 4457 messages
Test set size: 1115 messages
```

Baseline Model

I need a baseline to compare against before building complex models. the baseline i will be using is the a majority class classifier which predicts ham for everything as it the most common class. Any useful model should be able to outperform this easily.

```
# baseline to just predict majority class
baseline = DummyClassifier(strategy="most_frequent")
baseline.fit(X_train, y_train)
baseline_preds = baseline.predict(X_test)
print(f"Baseline accuracy: {accuracy_score(y_test, baseline_preds):.4f}")
```

```
Baseline accuracy: 0.8664
```

Naive Bayes Model

using TF-IDF to convert text to numbers, then Multinomial Naive Bayes for classification

```
# TF-IDF Vectorization Configuration

tfidf_vectorizer = TfidfVectorizer(
    ngram_range=(1, 2),
    max_features=5000
)

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# To Train Multinomial Naive Bayes classifier
naive_bayes_model = MultinomialNB()
naive_bayes_model.fit(X_train_tfidf, y_train)

# predict accuracy
nb_predictions = naive_bayes_model.predict(X_test_tfidf)

print(f"\nNaive Bayes Test Accuracy: {accuracy_score(y_test, nb_predictions):.4
```

```
Naive Bayes Test Accuracy: 0.9722
```

```
print("NAIVE BAYES CLASSIFICATION REPORT")

print(classification_report(y_test, nb_predictions, target_names=['Ham', 'Spam'

# confusion matrix
nb_confusion_matrix = confusion_matrix(y_test, nb_predictions)

plt.figure(figsize=(8, 6))
sns.heatmap(
    nb_confusion_matrix,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=['Ham (Predicted)', 'Spam (Predicted)'],
    yticklabels=['Ham (Actual)', 'Spam (Actual)']
)

nb_true_positives = nb_confusion_matrix[1, 1]
nb_false_negatives = nb_confusion_matrix[1, 0]
print(f"\nSpam Detection: {nb_true_positives} caught, {nb_false_negatives} miss

plt.title('Naive Bayes Confusion Matrix', fontsize=14)
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()
```

DistilBERT Model

```
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import DistilBertTokenizer, DistilBertForSequenceClassificati
from torch.optim import AdamW
import warnings
warnings.filterwarnings('ignore')

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using: {device}")
```

```
Using: cuda
```

Custom Dataset Class

PyTorch needs data in a specific format to work with DataLoaders. This class wraps our SMS data and handles tokenization converting each message into the input_ids, attention_mask, and label that BERT expects.

```
class SMSDataset(Dataset):


    def __init__(self, texts, labels, tokenizer, max_len=128):

        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):

        text = str(self.texts.iloc[idx])
        label = self.labels.iloc[idx]

        # To Tokenize with padding and truncation
        encoding = self.tokenizer(
            text,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'label': torch.tensor(label, dtype=torch.long)
        }
```

```
import os

# to load distilbert
cache_dir = os.path.join(os.getcwd(), 'model_cache')
os.makedirs(cache_dir, exist_ok=True)

tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased', cach
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-ur
model = model.to(device)
```

```
Some weights of DistilBertForSequenceClassification were not initialized from th
You should probably TRAIN this model on a down-stream task to be able to use it
```

```
BATCH_SIZE = 16

train_dataset = SMSDataset(
    texts=spam_data.loc[X_train.index, 'body_text'],
    labels=y_train,
    tokenizer=tokenizer
)
test_dataset = SMSDataset(
    texts=spam_data.loc[X_test.index, 'body_text'],
    labels=y_test,
    tokenizer=tokenizer
)

# DataLoaders handle batching and shuffling
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)

print(f"Training batches: {len(train_loader)}")
print(f"Test batches: {len(test_loader)}")
```

```
Training batches: 279
Test batches: 70
```

```
# adjusting the speed and number of epochs the model will do during the trainir
LEARNING_RATE = 2e-5
NUM_EPOCHS = 2
optimizer = AdamW(model.parameters(), lr=LEARNING_RATE)
```

```
#This is PyTorch training loop which feed data in, calculate loss, update weigh
def train_bert_model(model, train_loader, optimizer, device, epochs):

    model.train()
    loss_history = []

    for epoch in range(epochs):
        total_loss = 0
        num_batches = 0

        for batch in train_loader:

            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['label'].to(device)
```

```
            optimizer.zero_grad()

            outputs = model(
                input_ids=input_ids,
                attention_mask=attention_mask,
                labels=labels
            )
            loss = outputs.loss

            loss.backward()

            optimizer.step()

            total_loss += loss.item()
            num_batches += 1

        avg_loss = total_loss / num_batches
        loss_history.append(avg_loss)
        print(f"Epoch {epoch + 1}/{epochs} - Average Loss: {avg_loss:.4f}")

    return loss_history
```

```
def evaluate_bert_model(model, test_loader, device):

    model.eval()
    predictions = []
    actuals = []


    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['label'].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask)

            preds = torch.argmax(outputs.logits, dim=1)

            predictions.extend(preds.cpu().numpy())
            actuals.extend(labels.cpu().numpy())

    return predictions, actuals
```

```
# Train the DistilBERT model
import time
start_time = time.time()
training_loss = train_bert_model(model, train_loader, optimizer, device, NUM_EP
training_time = time.time() - start_time
```

```
TRAINING DISTILBERT MODEL
Epoch 1/2 - Average Loss: 0.0810
Epoch 2/2 - Average Loss: 0.0203
```

```
# Evaluate DistilBERT model
bert_predictions, bert_actuals = evaluate_bert_model(model, test_loader, device
print("DISTILBERT CLASSIFICATION REPORT")
print(classification_report(bert_actuals, bert_predictions, target_names=['Ham'

#confusion matrix
bert_confusion_matrix = confusion_matrix(bert_actuals, bert_predictions)

plt.figure(figsize=(8, 6))
sns.heatmap(
    bert_confusion_matrix,
    annot=True,
    fmt='d',
    cmap='Greens',
    xticklabels=['Ham (Predicted)', 'Spam (Predicted)'],
    yticklabels=['Ham (Actual)', 'Spam (Actual)']
)

# to extract metrics for comparison
bert_true_positives = bert_confusion_matrix[1, 1]
bert_false_negatives = bert_confusion_matrix[1, 0]
print(f"\nSpam Detection: {bert_true_positives} caught, {bert_false_negatives}
plt.title('DistilBERT Confusion Matrix', fontsize=14)
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()
```

Performance Analysis & Comparative Discussion

```
from sklearn.metrics import precision_score, recall_score, f1_score
import numpy as np

print("COMPREHENSIVE MODEL COMPARISON")
```

```
models = ['Baseline (Majority)', 'Naive Bayes + TF-IDF', 'DistilBERT']

# Accuracy scores
accuracies = [
    accuracy_score(y_test, baseline_preds),
    accuracy_score(y_test, nb_predictions),
    accuracy_score(bert_actuals, bert_predictions)
]

precisions = [
    precision_score(y_test, baseline_preds, zero_division=0),
    precision_score(y_test, nb_predictions),
    precision_score(bert_actuals, bert_predictions)
]

recalls = [
    recall_score(y_test, baseline_preds, zero_division=0),
    recall_score(y_test, nb_predictions),
    recall_score(bert_actuals, bert_predictions)
]

f1_scores = [
    f1_score(y_test, baseline_preds, zero_division=0),
    f1_score(y_test, nb_predictions),
    f1_score(bert_actuals, bert_predictions)
]

print(f"\n{'Model':<25} {'Accuracy':>10} {'Precision':>10} {'Recall':>10} {'F1-
for model, acc, prec, rec, f1 in zip(models, accuracies, precisions, recalls, f
    print(f"{model:<25} {acc:>10.4f} {prec:>10.4f} {rec:>10.4f} {f1:>10.4f}")
```

```
COMPREHENSIVE MODEL COMPARISON

Model                     Accuracy  Precision     Recall   F1-Score
Baseline (Majority)         0.8664     0.0000     0.0000     0.0000
Naive Bayes + TF-IDF        0.9722     1.0000     0.7919     0.8839
DistilBERT                  0.9901     0.9859     0.9396     0.9622
```

```
# Visual Comparison of Model Performance
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Grouped bar chart comparing all metrics
x = np.arange(len(models))
width = 0.2

ax1 = axes[0]
bars1 = ax1.bar(x - 1.5*width, accuracies, width, label='Accuracy', color='stee
bars2 = ax1.bar(x - 0.5*width, precisions, width, label='Precision', color='dar
bars3 = ax1.bar(x + 0.5*width, recalls, width, label='Recall', color='forestgre
```

```
bars4 = ax1.bar(x + 1.5*width, f1_scores, width, label='F1-Score', color='crims

ax1.set_xlabel('Model', fontsize=12)
ax1.set_ylabel('Score', fontsize=12)
ax1.set_title('Comprehensive Metric Comparison', fontsize=14)
ax1.set_xticks(x)
ax1.set_xticklabels(['Baseline', 'Naive Bayes', 'DistilBERT'], fontsize=10)
ax1.legend(loc='lower right')
ax1.set_ylim(0, 1.1)
ax1.grid(axis='y', alpha=0.3)

# Side by side confusion matrix comparison
ax2 = axes[1]
# Combine confusion matrices for visualization
combined_data = {
    'Naive Bayes': [nb_confusion_matrix[1,1], nb_confusion_matrix[1,0]],
    'DistilBERT': [bert_confusion_matrix[1,1], bert_confusion_matrix[1,0]]
}
x_labels = ['Spam Caught (TP)', 'Spam Missed (FN)']
x_pos = np.arange(len(x_labels))

bar_width = 0.35
ax2.bar(x_pos - bar_width/2, combined_data['Naive Bayes'], bar_width, label='Na
ax2.bar(x_pos + bar_width/2, combined_data['DistilBERT'], bar_width, label='Dis

ax2.set_xlabel('Classification Outcome', fontsize=12)
ax2.set_ylabel('Number of Messages', fontsize=12)
ax2.set_title('Spam Detection Comparison', fontsize=14)
ax2.set_xticks(x_pos)
ax2.set_xticklabels(x_labels)
ax2.legend()
ax2.grid(axis='y', alpha=0.3)

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

Quantitative Analysis

The results showed that the DistilBERT model had better performance than the Naive Bayes, however the majority of the class had worse performance. DistilBERT got 96% F1-score while naive bayes got 88% with DistilBERT getting an F1-score about 8 points higher than Naive Bayes showing that the contextual emmbedding is better for the SMS spam classification.But, both Models outperformed the baseline which got an f1-score of 0% for the spam in which it predicted all ham.

for this spam detection classification task, Recall is an important metric. Naive Bayes got recall 0.85 missing 112 spam messages failing to predict a large number of spam messages which resulted in more false negatives and the increasing of the amount of spam messages reaching users. in contrast, DistilBERT got higher recall with 0.94 missing 45 spam messages showing that it has a good ablity to detect the spam messages even if they do not contain trigger words.

The Analysis of the confusion matrices shows that DistilBERT correctly predicted most spam messeges with low missed predictions while maintaining comparable precision showing that it learned more subtle linguistic patterns associated with spam content. in contrast, the naive bayes predicted less spam messeges and missed more that DistelBERT.

Qualitative Analysis:

How each model represents text is what differs performance from one model to another. considering the Naive Bayes with TF-IDF, it works with text as independent words which ignores the order of the words and their meanings and context. for instance, the message "Call me back about winning" can be predicted as either ham or spam. on the other hand, BERT works by identifing suspicious phrases by evaluating word combinations in context. Additionally, spam messages now use character substitutions such as FR33 instead of FREE which can make it harder for model like naive bayes to detect but the preprocessing can help while BERT's WordPiece tokenization recognizes these variations with subword patterns and it also detects the semantic cues that appear in the spam messages even if there are not clear spam keaywords

In conclusion , Some points must be kept in mind in which This improvement in f1-score comes at a high computational cost. Naive Bayes trained in milliseconds while DistilBERT took about 41 minutes to train on cpu and required GPU use to train in 2 minutes. As a result, Naive Bayes is better suited for resource constrained environments, whereas DistilBERT is more appropriate for server side deployment.

Project Summary and Reflections

A comparision of a statistical model which is TF-IDF with Multinomial Naive Bayes and a modern deep learning model which is DistelBERT was the aim of this project. using the UCI SMS Spam Collection dataset, Both models were trained and assessed on their ability to classify messages as spam or ham.The results demonstrated that while DistilBERT needed a lot more processing power, it outperformed Naive Bayes in accuracy and F1-score.

I honestly expected the Naive Bayes to perform badly in comparison to DistilBERT But i was surprised with this small f1 score gap which was much less that i expected which showed that simple baselines can be sufficient for classification tasks such as this task keeping in mind its cheap computational cost. I learned alot of things during this project. starting with that GPU acceleration is essential for transformers as using CPU cost the model 41 minutes in training while 1 to 2 minutes with GPU. In addition, the class imbalance seemed to matter more that i had in mind. I thought at first that accuracy evaluation is a reliable metric until my baseline which just does the simple process of predicting the majority class to score 87%, then i realised that it can not be relied on with imbalanced datasets and other metrics should be considered. i also figured out that preprocessing depends on the model as a model like DistelBERT uses the raw unprocessed text to give better results.

Considering the Limitations of This project , the dataset is from 2011 which is 15 years backs and the spammers defenitely developed new ways and strategies such as using emojis or informal language. Also , the dataset contains only english language messeges and no other languages messages were evaluated in which the results will not generalize to other languagues.Also , Due to time and resource constraints, the DistilBERT model was trained for only two epochs. however, additional training and hyperparameter tuning could improve the performance.

References

Almeida, T.A., Gómez Hidalgo, J.M., Yamakami, A. (2011). Contributions to the Study of SMS Spam Filtering: New Collection and Results

Devlin, J., Chang, M.W., Lee, K. and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter

UCI Machine Learning Repository: SMS Spam Collection Dataset

Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python