SmartMailGuard: AI-Powered Email Classification

Rupak R. Gupta (RRG)
B. Tech Information Technology (IT)
June 2024

Abstract

In today's age, our inboxes are constantly being flooded with emails from all kinds of sources, which makes finding a legitimate email in a sea of spam challenging. The aim of this project is to develop an Artificial Intelligence (AI) powered tool that can parse a given email and classify whether it is legitimate or spam.

Acknowledgements

I would like to thank the folks behind Project X (our seniors) for providing us with an opportunity to learn these topics. They were also very helpful in resolving doubts and approving tasks. I also appreciate my peers for fostering a competitive atmosphere that motivated me to solve the given tasks independently. I also thank the online resources that helped me understand the concepts required for the tasks such as Support Vector Machines (SVM) and the Bilingual Evaluation Understudy (BLEU) score. Additionally, I am grateful to my family for their understanding and support, allowing me to stay up late at night to work on this project even after the semester ended.

Contents

Section 1

Preface

About me

I am Rupak R. Gupta, though most people know me as RRG. I am a (soon-to-be) second year student at VJTI in IT. I have a deep interest in statistics, probability, and mathematics in general, and I am very curious about their applications in the field of computer science for problem-solving.

I have had a strong grasp on the programming languages Java, C++ and Python; I have practised some standard Data Structures and Algorithms using each of these languages.

I have also acquired skills in writing LaTeX and creating digital diagrams, some of which are included in this proposal. I wish to utilize these skills as an educator/explainer myself to contribute to the education of others.

Tasks repository: `https://github.com/aitwehrrg/Project-X/`

Motivation for this topic

As mentioned earlier, I have a lot of interest in topics such as probability and statistics, which encouraged me to select a project based on Machine Learning (ML). Another reason is that I am very much a patron of the best practices in cybersecurity, such as password management and email aliasing. Avoiding spam and phishing emails is yet another important cybersecurity practice, so this topic felt coherent to me.

Section 2

Theory and Approach

2.1 Preprocessing

We will convert the text data from the emails into a format that can be fed into the neural network. This includes:

1. Cleaning up unwanted words and characters (like punctuation).
2. Converting all text to lowercase.
3. Extracting individual *tokens* from the data.

2.2 Neural Networks

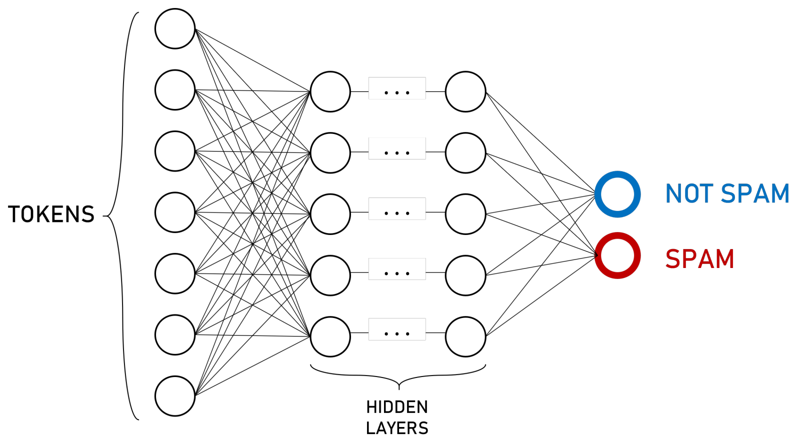


Figure 2.1: Neural Network

A prototype Recurrent Neural Network (RNN) is depicted in fig. 2.1.

- **Input or embedding layer:** This layer has as many nodes as there are tokens. Each token input is vectorized using techniques such as `Word2Vec` and fed into the network.
- **LSTM layer:** Long Short-Term Memory (LSTM) [1] is a type of RNN architecture which is designed to remember information for long periods of time, allowing it to learn semantics of a some text using the context of words (tokens) present long back in the email.

- **Hidden layers:** These layers perform the computation to generate the required probabilities. The algorithm used is discussed in § 2.4.
- **Output layer:** This layer has two neurons that hold their respective final probability of the given email being not spam and the given email being spam. An alternative idea would be to use a *sigmoid function* [2] to output just one binary output; the email being spam or not.

2.3 Transformers

The architecture of a transformer is depicted in fig. 2.2.

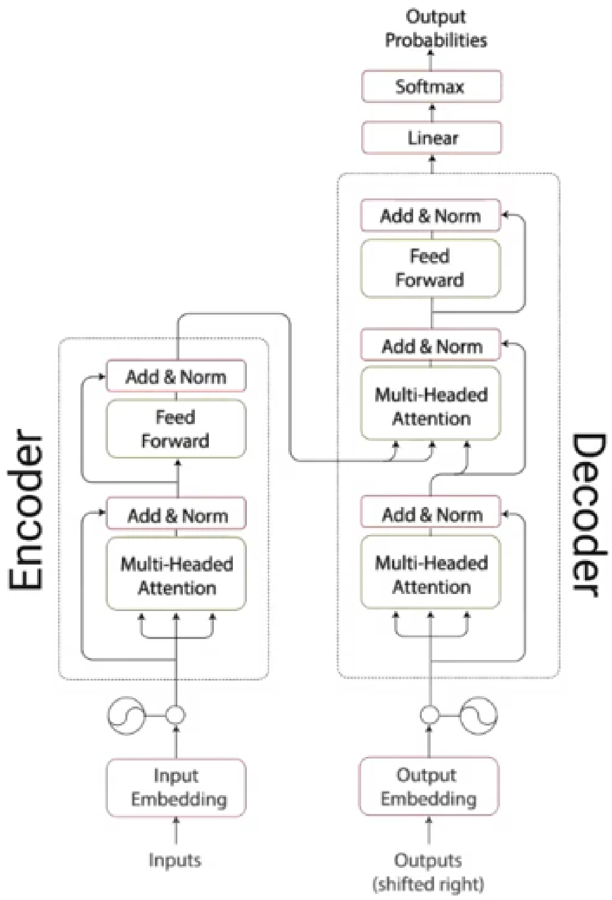


Figure 2.2: Transformer Architecture [1]

- We shall select a pretrained transformer model to use such as Bidirectional Encoder Representations from Transformers (BERT), or Generative Pretrained Transformer (GPT) like GPT-2 [1].
- Using attention mechanism [3], understand semantics of text in the email to properly encode the tokens into vectors.
- The transformer is integrated with the LSTM described in § 2.2 using methods such as Concatenation, Stacking *etc.*

2.4 Naïve Bayes Algorithm

2.4.1 Bayes' Theorem

We will obtain a sample of emails that we can use for training. Our training data would be used to determine the *prior probability* [4] of an email being spam $P(\text{Spam})$.

After preprocessing, we can determine the *posterior probability* [5] for each token by using Bayes' theorem.

$$P(\text{Token} \mid \text{Spam}) := \frac{P(\text{Spam} \mid \text{Token})P(\text{Token})}{P(\text{Spam})} \tag{2.1}$$

$P(\text{Spam} \mid \text{Token})$ and $P(\text{Token})$ are probabilities that can be estimated from the training data.

2.4.2 Naïve Bayes

Using Naïve Bayes, we can determine the total probability that an email is spam by multiplying the posterior probabilities of each token [4].

$$\begin{aligned} P(\text{Spam} \mid \text{Tokens}) &= \frac{P(\text{Spam}) \times \prod P(\text{Token} \mid \text{Spam})}{\prod P(\text{Token})} \\ &\propto \boxed{P(\text{Spam}) \times \prod_{\text{all Tokens}} P(\text{Token} \mid \text{Spam})} \end{aligned} \tag{2.2}$$

We can ignore the prior probability $\prod P(\text{Token})$ since it is independent of the legitimacy of the email. Therefore, we only care about the relative value of the expression in eq. (2.2).

2.4.3 Smoothing

To account for exactly zero posterior probabilities of certain tokens in one of the categories, we will use *Laplace smoothing* [6]: $\alpha = 1$ to add one point of frequency to every token (figs. 2.3 and 2.4). Another option is using *additive smoothing* for other values of α.

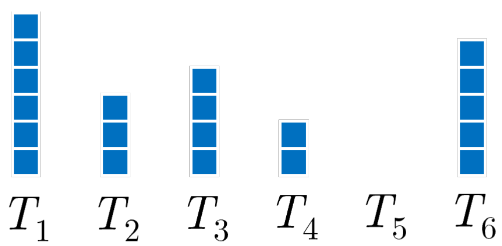


Figure 2.3: Before smoothing

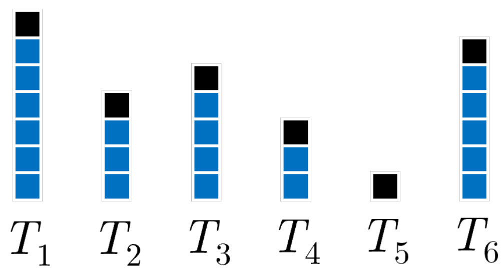


Figure 2.4: After smoothing ($\alpha = 1$)

Section 3

Workflow

3.1 Week 1

Gather a large enough sample for training and testing the model. Study the sample and determine the techniques to preprocess the data in order for it to be fed to the neural network.

3.2 Week 2

Preprocess the data: clean the text to remove unwanted information, split into individual tokens. Learn the implementations of RNN (LSTM) and the Naïve Bayes Algorithm.

3.3 Weeks 3 and 4

Implement the neural network and and train it on the training data. Study the transformer model to use such as BERT, GPT or GPT-2, as well as the technique to integrate it to the network such as Concatenation or Stacking.

3.4 Week 5 onwards

Integrate the transformer model with the neural network and evaluate its performance on the testing sample. Grade its accuracy, fix bugs and tweak parameters to maximise accuracy and efficiency.

Appendix A

Task 1: Difficult dating

Link: `https://github.com/aitwehrrg/Project-X`

A.1 Code

This task is completed as a Jupyter notebook.

Importing packages

[1]:
```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

Defining constants

[2]:
```python
TEST_SIZE: float = 0.2  # Train size would be 1 - TEST_SIZE
TEST_CASE: str = 'dating.csv'
```

Importing the `csv` as a `pandas DataFrame`

[3]:
```python
df: pd.DataFrame = pd.read_csv(TEST_CASE)
df.drop(df.columns[0], axis=1, inplace=True)  # Dropping the first
    column (serial number)
```

Preprocessing

Preprocessing `career`

[4]:
```
df['career'] = df['career'].fillna('undecided')  # Filling empty␣
↪cells with 'undecided'
df['career'] = df['career'].str.lower()  # Turining everything␣
↪lowercase to prepare for encoding
```

Preprocessing using One Hot Encoder

[5]:
```
encoder = OneHotEncoder()

# Encoding the string column
career_encoded = encoder.fit_transform(df[['career']])

# Convert back to DataFrame
career_encoded_df = pd.DataFrame(career_encoded.toarray(),␣
↪columns=encoder.get_feature_names_out(['career']))

# Combining with original DataFrame
df = df.reset_index(drop=True)
df = df.join(career_encoded_df)
df.drop('career', axis=1, inplace=True)  # No more need for the␣
↪original string column
```

Preprocessing the numeric columns (`age` and `income`)

[6]:
```
df = df.fillna(df.mean())  # Replacing empty cells with mean instead
```

Standard Scaling all numeric columns (Mean = 0, Standard Deviation = 1)

[7]:
```
scaler = StandardScaler()
# Attractiveness and fun are not scaled because we will need them␣
↪for plotting
df[['age', 'income', 'sinc', 'intel', 'amb', 'like']] = scaler.
↪fit_transform(df[['age', 'income', 'sinc', 'intel', 'amb',␣
↪'like']])
```

Splitting into training and testing data

[8]:
```
X = df.drop('dec', axis=1)  # The SVM must not train on the output
y = df['dec']  # We will test the output
X_train, X_test, y_train, y_test = train_test_split(X, y,␣
↪test_size=TEST_SIZE, random_state=69)
```

Training an SVM classifier

[9]:
```
svm = SVC()
svm.fit(X_train, y_train)
```

[9]:
```
SVC()
```

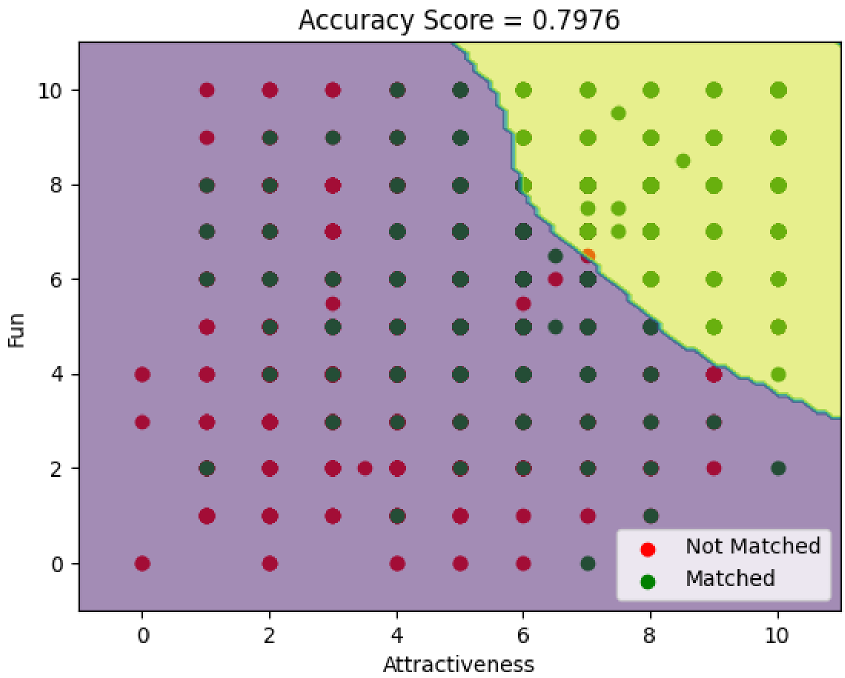
Predicting the output of the test data and calculate the accuracy

[10]:
```
y_pred = svm.predict(X_test)
accuracy: float = accuracy_score(y_test, y_pred)
accuracy
```

[10]:
```
0.7975852272727273
```

A.2 Output

[11]:
```
# Setting up the plot
svm.fit(X_train[['attr', 'fun']], y_train)  # Fit the plot based on attractiveness and fun
plt.scatter(X_train[y_train == 0]['attr'],
        X_train[y_train == 0]['fun'],
        c='r',
        label='Not Matched')
plt.scatter(X_train[y_train == 1]['attr'],
        X_train[y_train == 1]['fun'],
        c='g',
        label='Matched')
# Adding labels and title
plt.xlabel('Attractiveness')
plt.ylabel('Fun')
plt.title('Accuracy Score = ' + str(round(accuracy, 4)))
plt.legend()
# Plotting the decision boundary
DecisionBoundaryDisplay.from_estimator(
svm,
X_train[['attr', 'fun']],
response_method='predict',
alpha=0.5,
ax=plt.gca()
)
# Final output
plt.show()
```



References

- *SVM* by StatQuest with Josh Starmer [7]

Appendix B

Task 2: Where my diamonds at?

Link: `https://github.com/aitwehrrg/Project-X`

B.1 Code

This task is completed had a really long codebase, so it has been split into four Python files.

Main

```python
import cv2
import numpy as np
import process_image as pi
import start_finder as sf
import vertices_finder as vf

BLACK: int = 0
WHITE: int = 255

TEST_CASES_1: tuple[str, str, str] = ('pics/tc1-1.png', 'pics/tc1-2.png',
    'pics/tc1-3.png')
TEST_CASES_2: tuple[str, str, str] = ('pics/tc2-1.png', 'pics/tc2-2.png',
    'pics/tc2-3.png')

SUIT_DICT: dict[int, str] = {
    0: 'Hearts',
    1: 'Clubs',
    2: 'Spades',
    3: 'Diamonds'
}

VALUE_DICT: dict[int, str] = {
    0: 'Ace',
    1: '2',
    2: '3',
    3: '4',
    4: '5',
    5: '6',
}


```

```python
# Traverse the top row
def horizontal_ratios(image: np.ndarray, target: int) -> list[float]:
    start_x, start_y = sf.StartFinder(image, target).left()
    ratios: list[float] = []
    _, width = image.shape
    for x in range(start_x, width):
        if image[start_y][x] == target and image[start_y][x - 1] != target:
            ratio: float = vf.VerticesFinder(image, target, (x, start_y), 'left').aspect_ratio()
            if ratio != 0:
                ratios.append(ratio)
    return ratios


# Traverse the left column
def vertical_ratios(image: np.ndarray, target: int, start_x: int = 0, start_y: int = 0) -> list[float]:
    ratios: list[float] = []
    height, _ = image.shape
    for y in range(start_y, height):
        if image[y][start_x] == target and image[y - 1][start_x] != target:
            ratio: float = vf.VerticesFinder(image, target, (start_x, y), 'top').aspect_ratio()
            if ratio != 0:
                ratios.append(ratio)
    return ratios


# Required function
def output(image: np.ndarray, target: int, suit: str | None = None, value: str | None = None) -> str:
    if suit is None:
        suit_ratios: list[float] = horizontal_ratios(image, BLACK)
        suit_min: float = min(suit_ratios)
        suit_max: float = max(suit_ratios)

        # Check the minimum and maximum ratios of the rop row to determine the nature of the card
        if suit_min == 1 and suit_max == 1:  # the card is upside down and not a 6
            image = cv2.flip(image, -1)  # hold the card upright
            return output(image, target, suit, value)  # try again

        if suit_min < 1 and suit_max == 1:  # the card is upside down and a 6
            value = VALUE_DICT[5]  # the card is a 6 (since it is upside down)
            image = cv2.flip(image, -1)  # hold the card upright
            return output(image, target, suit, value)  # try again

        if suit_min < 1 < suit_max:  # the card is upright and an ace
            suit = SUIT_DICT[suit_ratios.index(suit_max) - 1]  # minus 1 because there are 5 diamonds but only 4 suits
            value = VALUE_DICT[0]  # the card is an ace
            return output(image, target, suit, value)

        # the card is upright and not an ace
        suit = SUIT_DICT[suit_ratios.index(suit_max) - 1]  # minus 1 because there are 5 diamonds but only 4 suits

    # if the upright card is not an ace and the upside down card is not a 6
    if value is None:
```

```python
        top_x, top_y = sf.StartFinder(image, target).top()
        value_ratios = vertical_ratios(image, target, top_x, top_y)
        value = VALUE_DICT[value_ratios.index(min(value_ratios))]

    output_: str = f'{value} of {suit}'
    print(output_)
    cv2.imshow(output_, image)
    cv2.waitKey(0)
    return output_


def main() -> None:
    # Select the test case from here
    # test_cases: tuple[str, str, str] = TEST_CASES_1
    # test_cases: tuple[str, str, str] = TEST_CASES_2

    test_cases: tuple = TEST_CASES_1 + TEST_CASES_2
    result: str = ''
    for test_case in test_cases:
        image = cv2.imread(test_case, cv2.IMREAD_GRAYSCALE)
        image = pi.Process(image, BLACK, WHITE).process()
        result += f'{output(image, BLACK)}, '
    result += '\b\b'  # remove trailing comma and space; on some terminals it may not
        work
    print(f'Final result: {result}')


if __name__ == '__main__':
    main()
```

Listing B.1: main.py

Process Image

```python
import cv2
import numpy as np

BLACK: int = 0
GRAY: int = 28
WHITE: int = 255


class Process:
    def __init__(self, image: np.ndarray, border_value: int, bg_value: int,
        tolerance: int = 2) -> None:
        self._image = image
        self._border_value = border_value
        self._bg_value = bg_value
        self._TOLERANCE = tolerance  # 2 is the tightest tolerance

    def _remove_border(self) -> None:
        height, width = self._image.shape
        for y in range(height):
            for x in range(width):
                if self._image[y][x] == self._border_value:
                    self._image[y][x] = self._bg_value
        # cv2.imshow('Border removed', self._image)
        # cv2.waitKey(0)
```

```python

    # Crops the image to remove the remaining pixels that are not the background
    def _crop(self) -> None:
        height, width = self._image.shape
        # Left crop
        for x in range(width):
            if self._image[0][x] != self._bg_value:
                self._image = self._image[:, x + 1:]
                _, width = self._image.shape
                break
        # Right crop
        for x in range(width - 1, -1, -1):
            if self._image[height - 1][x] != self._bg_value:
                self._image = self._image[:, :x - 1]
                _, width = self._image.shape
                break
        # Top crop
        for y in range(height):
            if self._image[y][0] != self._bg_value:
                self._image = self._image[y + 1:, :]
                height, _ = self._image.shape
                break
        # Bottom crop
        for y in range(height - 1, -1, -1):
            if self._image[y][width - 1] == self._bg_value:
                self._image = self._image[:y - 1, :]
                break
        # cv2.imshow('Cropped', self._image)
        # cv2.waitKey(0)

    def process(self) -> np.ndarray:
        self._remove_border()

        # Remove remnants of the border (salt and pepper noise)
        self._image = cv2.medianBlur(self._image, 3)
        # cv2.imshow('Median blur', self._image)
        # cv2.waitKey(0)

        self._crop()

        # Remove any pixels that are not the required color
        self._image[self._image > GRAY + self._TOLERANCE] = WHITE
        self._image[self._image < GRAY - self._TOLERANCE] = WHITE
        # cv2.imshow('Pixel color filter', self._image)
        # cv2.waitKey(0)

        # Convert the image to binary
        _, self._image = cv2.threshold(self._image, 254, 255, cv2.THRESH_BINARY)
        return self._image
```

Listing B.2: process_image.py

Start Finder

```python
import numpy as np


class StartFinder:
```

```python
    def __init__(self, image: np.ndarray, target: int) -> None:
        self._image: np.ndarray = image
        self._height, self._width = image.shape
        self._target: int = target

    def top(self) -> tuple[int, int]:
        for y in range(self._height):
            for x in range(self._width):
                if self._image[y][x] == self._target:
                    return x, y
        raise ValueError('Target not found')

    def left(self) -> tuple[int, int]:
        for x in range(self._width):
            for y in range(self._height):
                if self._image[y][x] == self._target:
                    return x, y
        raise ValueError('Target not found')
```

Listing B.3: start_finder.py

Vertices Finder

```python
import numpy as np

# To account for pixel inaccuracies
MIN_RATIO: float = 0.96
MAX_RATIO: float = 1.04


class VerticesFinder:
    def __init__(self, image: np.ndarray, target: int, initial_vertex: tuple[int,
        int], direction: str) -> None:
        self._image: np.ndarray = image
        self._height, self._width = image.shape
        self._target: int = target
        self._initial_vertex = initial_vertex
        self._start_x, self._start_y = initial_vertex
        self._direction: str = direction

    def _top(self) -> tuple[tuple[int, int], tuple[int, int], tuple[int, int]]:
        # Find bottom vertex
        y: int = self._start_y
        for y in range(self._start_y, self._height):
            if self._image[y, self._start_x] != self._target:
                break
        bottom_vertex: tuple[int, int] = self._start_x, y - 1

        mid_y: int = (self._start_y + bottom_vertex[1]) // 2

        # Find right vertex
        x = self._start_x
        for x in range(self._start_x, self._width):
            if self._image[mid_y, x] != self._target:
                break
        right_vertex: tuple[int, int] = x - 1, mid_y

        # Find left vertex
```

```python
        left_vertex: tuple[int, int] = 2 * self._start_x - right_vertex[0], mid_y

        return bottom_vertex, left_vertex, right_vertex

    def _left(self) -> tuple[tuple[int, int], tuple[int, int], tuple[int, int]]:
        # Find right vertex
        x: int = self._start_x
        for x in range(self._start_x, self._width):
            if self._image[self._start_y, x] != self._target:
                break
        right_vertex: tuple[int, int] = x - 1, self._start_y

        mid_x: int = (self._start_x + right_vertex[0]) // 2

        # Find bottom vertex
        y = self._start_y
        for y in range(self._start_y, self._height):
            if self._image[y, mid_x] != self._target:
                break
        bottom_vertex: tuple[int, int] = mid_x, y - 1

        # Find top vertex
        top_vertex: tuple[int, int] = mid_x, 2 * self._start_y - bottom_vertex[1]

        return top_vertex, bottom_vertex, right_vertex

    def vertices(self) -> tuple[tuple[int, int], tuple[int, int], tuple[int, int], tuple[int, int]]:
        # top, bottom, left, right
        match self._direction:
            case 'top':
                bottom_vertex, left_vertex, right_vertex = self._top()
                return self._initial_vertex, bottom_vertex, left_vertex, right_vertex

            case 'left':
                top_vertex, bottom_vertex, right_vertex = self._left()
                return top_vertex, bottom_vertex, self._initial_vertex, right_vertex

            case _:
                raise ValueError(f'Invalid direction: {self._direction}')

    def aspect_ratio(self) -> float:
        top_vertex, bottom_vertex, left_vertex, right_vertex = self.vertices()
        width = right_vertex[0] - left_vertex[0]
        height = bottom_vertex[1] - top_vertex[1]
        try:
            ratio: float = round(width / height, 2)
        except ZeroDivisionError:
            return 0

        # To account for pixel inaccuracies
        if MIN_RATIO < ratio < MAX_RATIO:
            return 1

        return ratio
```

Listing B.4: vertices_finder.py

B.2 Output

```
2 of Spades
3 of Clubs
4 of Hearts
3 of Spades
6 of Hearts
6 of Clubs
Final result: 2 of Spades, 3 of Clubs, 4 of Hearts, 3 of Spades, 6 of Hearts, 6 of Clubs

Process finished with exit code 0
```

References

- Computer Vision (CV) Xplore Workshop materials [8]

Appendix C

Task 3: This BLEU me away

Link: https://github.com/aitwehrrg/Project-X

C.1 Code

This task is completed using one Python file.

```python
from collections import Counter
import math

TEST_CASE_1: tuple[list[str], str] = (['It is a guide to action that ensures that the military will forever heed Party '
                                       'commands',
                                       'It is the guiding principle which guarantees the military forces always being '
                                       'under the command of the Party',
                                       'It is the practical guide for the army always to heed the directions of the '
                                       'party'],
                                      'It is a guide to action which ensures that the military always obeys the '
                                      'commands of the party')

TEST_CASE_2: tuple[list[str], str] = (['It is a guide to action that ensures that the military will forever heed Party '
                                       'commands',
                                       'It is the guiding principle which guarantees the military forces always being '
                                       'under the command of the Party',
                                       'It is the practical guide for the army always to heed the directions of the '
                                       'party'],
                                      'It is the to action the troops forever hearing the activity guidebook that '
                                      'party direct')


# Maximum number of n-grams
N: int = 4


# Compute n-grams
```

```python
def n_grams(sentence: str, n: int = 1) -> list[str]:
    words: list[str] = sentence.split()
    if n == 1:
        return words
    n_grams_: list[str] = []
    for i in range(len(words) - n + 1):
        n_grams_.append(' '.join(words[i:i + n]))
    return n_grams_


# Count frequency of n-grams
def count(n_gram: list[str]) -> int:
    return sum(Counter(n_gram).values())


# Count frequency of clipped n-grams
def count_clip(candidate: str, references: list[str], n: int) -> int:
    n_gram_candidate: list[str] = n_grams(candidate, n)
    n_gram_references: list[list[str]] = [n_grams(reference, n) for reference in
        references]
    counts: Counter = Counter(n_gram_candidate)
    for key in counts:
        max_reference_count: int = 0
        for n_gram_reference in n_gram_references:
            # n_gram_reference_count: Counter = Counter(n_gram_reference)
            max_reference_count = max(max_reference_count,
                n_gram_reference.count(key))
        counts[key] = min(counts[key], max_reference_count)
    return sum(counts.values())


# Calculate p_n
def precision(candidate: str, references: list[str], n: int) -> float:
    count_: int = count(n_grams(candidate, n))
    count_clip_: int = count_clip(candidate, references, n)
    # print(n, f'{count_clip_}/{count_}')
    return count_clip_ / count_ if count_ != 0 else 0


# Calculate BP
def brevity_penalty(candidate: str, references: list[str]) -> float:
    c: int = len(n_grams(candidate))  # total length of candidate
    word_count_references: list[int] = [len(n_grams(reference)) for reference in
        references]
    deviations: list[int] = [abs(c - reference) for reference in
        word_count_references]
    r: int = word_count_references[deviations.index(min(deviations))]  # best match
        reference length
    if c > r:
        return 1
    return math.exp(1 - r / c)


# Required function
def bleu_score(references: list[str], candidate: str) -> float:
    bleu: float = 1
    bp: float = brevity_penalty(candidate, references)
    print(f'BP: {bp}')
    for n in range(1, N + 1):
```

```python
        p_n: float = precision(candidate, references, n)
        bleu *= p_n
    bleu **= 1 / N
    bleu *= bp
    return bleu


def main():
    test_cases: tuple = (TEST_CASE_1, TEST_CASE_2)
    for test_case in test_cases:
        bleu: float = bleu_score(test_case[0], test_case[1])
        print(f'BLEU score: {bleu}\n')


if __name__ == '__main__':
    main()
```

Listing C.1: main.py

C.2 Output

```
BP: 1.0
BLEU score: 0.5045666840058485

BP: 0.9355069850316178
BLEU score: 0.0


Process finished with exit code 0
```

References

- "*BLEU: a Method for Automatic Evaluation of Machine Translation*" [9]
- *C5W3L06 BLEU Score* by DeepLearningAI [10]

Acronyms

Bleu Bilingual Evaluation Understudy 1, 19, 21

AI Artificial Intelligence 1

BERT Bidirectional Encoder Representations from Transformers 5, 7

CV Computer Vision 18

GPT Generative Pretrained Transformer 5, 7

IT Information Technology 1, 3

LSTM Long Short-Term Memory 4, 5, 7

ML Machine Learning 3

RNN Recurrent Neural Network 4, 7

RRG Rupak R. Gupta 1, 3

SVM Support Vector Machines 1, 10, 11

References

[1] Michael Phi. Illustrated guide to transformers- step by step explanation. https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0.

[2] Neural networks. http://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. http://arxiv.org/abs/1706.03762.

[4] StatQuest with Josh Starmer. Naive bayes, clearly explained!!! https://www.youtube.com/watch?v=O2L2Uv9pdDA.

[5] Petuum Inc. Intro to modern bayesian learning and probabilistic programming. https://petuum.medium.com/intro-to-modern-bayesian-learning-and-probabilistic-programming-c61830df5c50.

[6] Scikit-learn. 1.9. Naive Bayes. https://scikit-learn/stable/modules/naive_bayes.html.

[7] StatQuest with Josh Starmer. Support vector machines part 1 (of 3): Main ideas!!! https://www.youtube.com/watch?v=efR1C6CvhmE.

[8] Computer vision Xplore workshop - google drive. https://drive.google.com/drive/folders/1R2WOCz8mDvmbE0qSEnq51uNh8EZigTU9.

[9] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. https://aclanthology.org/P02-1040.

[10] DeepLearningAI. C5w3l06 bleu score (optional). https://www.youtube.com/watch?v=DejHQYAGb7Q.

$$\boxed{\begin{bmatrix} \mathbb{R} & \mathfrak{R} \\ g & \infty \end{bmatrix}}$$

The End.