Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Inteligencia Artificial Avanzada
Curso 2023-2024

Sistema de detección automática de Phishing en correos electrónicos

Álvaro Fontenla León

La Laguna, 1 de mayo de 2024

Índice general

Índice de Figuras

Capítulo 1

Introducción

1.1. Contexto

En este informe se describirá el proceso realizado para el desarrollo de un modelo de clasificación para detectar correos electrónicos con amenzas de Phishing, trabajando sobre la rama de clasificación de textos en lenguaje natural.

Capítulo 2

Desarrollo e implementación

2.1. Entorno de trabajo

El modelo de clasificación será desarrollado utilizando el lenguaje de programación Python. Esto se debe al amplio catálogo de librerías que ofrece para el procesamiento del lenguaje natural, lo que facilitará en gran medida el trabajo a realizar.

2.2. Preprocesamiento

La tarea de preprocesar el texto es un punto de inflexión en el resultado del modelo. Esto se debe a las decisiones que se han de tomar para obtener un vocabulario de buena calidad, cuya información sea relevante y sin incluir palabras que no sean significativas.

Para ello, se han realizado las siguientes tareas de preprocesamiento:

- Conversión del texto a minúsculas.
- Eliminación de los signos de puntuación.
- Eliminación de las “stopwords“.
- Eliminación de palabras vacías.
- Eliminación de palabras no pertenecientes al diccionario.
- Sustitución de palabras que contengan “http“ por el token “URL“.

2.2.1. Implementación

Se ha hecho uso de la librería NLTK (Natural Language Toolkit), la cual incluye distintos métodos diseñados para el tratamiento del lenguaje natural.

```python
def preprocess_text(text):
    # Convertir a minúsculas
    text = text.lower()

    # Eliminar signos de puntuación
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Tokenizar el texto
    tokens = word_tokenize(text)

    # Eliminar palabras reservadas (stopwords)
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    for i in range(len(tokens)):
        if "http" not in tokens[i]:
            # Eliminar palabras que contienen números y no significan nada
            if any(char.isdigit() for char in tokens[i]):
                tokens[i] = ""

            # Filtrar palabras por pertenencia al diccionario de palabras comunes
            if tokens[i] not in word_set:
                tokens[i] = ""
        else:
            tokens[i] = "URL"

    # Eliminar palabras vacías
    tokens = [word for word in tokens if word ≠ ""]

    return tokens
```

Figura 2.1: Código del preprocesamiento

2.3. Vocabulario

En cuanto a la creación del vocabulario, para todo el corpus de entrenamiento, se ha realizado el preprocesamiento y se han incluido las palabras en un set, evitando que se repitan.

```python
def vocabulary(file_name, output_file):
  # Leer el archivo CSV y procesar el texto
  file_path = os.path.join(os.path.dirname(__file__), '..', 'data', file_name)

  # Lista para almacenar los textos completos
  textos_completos = []

  with open(file_path, 'r', encoding='utf-8') as train_file:
      data = csv.reader(train_file)
      for row in data:
          for text in row:
              textos_completos.append(text)

  # Preprocesar todos los textos completos
  textos_procesados = [preprocess_text(texto) for texto in textos_completos]

  # Crear el vocabulario
  vocabulario = set()
  for texto in textos_procesados:
    for palabra in texto:
      vocabulario.add(palabra)

  # Guardar el vocabulario en un archivo
  nombre_archivo = os.path.join(os.path.dirname(__file__), '..', 'data', output_file)
  guardar_vocabulario(vocabulario, nombre_archivo)
  print(f"Vocabulario guardado en '{nombre_archivo}' con éxito.")

  return vocabulario
```

Figura 2.2: Código para la generación del vocabulario

2.4. Modelo del lenguaje

Para obtener el modelo del lenguaje, es necesario dividir el conjunto de entrenamiento en las dos clases, safe y phishing, y obtener los corpus de cada clase. A estos corpus se le aplicará el mismo preprocesamiento al que hemos sometido para obtener el vocabulario.

Una vez tenemos los corpus, obtenemos las probabilidades de cada palabra para ese corpus, aplicando el logaritmo neperiano para evitar un error de underflow. Para hacer uso de las operaciones matemáticas se ha hecho uso de la librería “math“, la cual nos permite utilizar el logaritmo neperiano.

```python
def calculate_probabilities(vocabulary, corpus):
    word_probabilities = {}
    corpus_size = len(corpus)
    vocabulary_size = len(vocabulary)
    unknown_probability = math.log(1 / (corpus_size + vocabulary_size))
    for word in vocabulary:
        # Count the times the word appears in the corpus.
        word_count = corpus.count(word)
        probability = math.log((word_count + 1) / (corpus_size + vocabulary_size))
        word_probabilities[word] = {'count': word_count, 'log_probability': probability}
    # Calculate the probability of the unknown words.
    word_probabilities['UNK'] = {'count': 0, 'log_probability': unknown_probability}
    return word_probabilities
```

Figura 2.3: Código para el cálculo de probabilidades

2.5. Clasificación

Por último, para clasificar los correos, es necesario preprocesar los correos de entrada, al igual que hemos realizado anteriormente. Una vez preprocesados, para cada correo, calculamos sus probabilidades, sumando las probabilidades de cada palabra, discriminando según la mayor probabilidad entre safe y phishing.

```python
# Función para clasificar un correo
def classify(mail):
    phishing_score = 0
    safe_score = 0
    for word in mail:
        if word in phishing_probabilities:
            phishing_score += phishing_probabilities[word]['logProb']
        else:
            phishing_score += phishing_probabilities['UNK']['logProb']
        if word in safe_probabilities:
            safe_score += safe_probabilities[word]['logProb']
        else:
            safe_score += safe_probabilities['UNK']['logProb']
    safe_score += math.log(safe_documents / (safe_documents + phishing_documents))
    phishing_score += math.log(phishing_documents / (safe_documents + phishing_documents))
    return safe_score, phishing_score, 'S' if safe_score > phishing_score else 'P'
```

Figura 2.4: Código para la clasificación

Capítulo 3

Estimación de errores

3.1. Estimación de error sin dividir conjuntos

La primera estimación calculada es utilizando el mismo conjunto de entrenamiento como el de test. Para este modelo, arroja un resultado de exactitud del 95 %.

```
Introduce el nombre del fichero de correos a clasificar (se asume que el fichero está en el directorio ../data): PH_train-no-classified.csv
Accuracy: 0.95
```

Figura 3.1: Exactitud con el modelo entrenado y testeado con el mismo conjunto

3.2. Estimación de error con conjuntos divididos

Para la segunda estimación, se ha dividido el conjunto original de correos electrónicos en dos subconjuntos, los primeros 10.000 correos como conjunto de entrenamiento y los restantes como conjunto de test. Habiendo realizado los cálculos nuevamente, el resultado de exactitud es un 93 %.

```
Introduce el nombre del fichero de correos a clasificar (se asume que el fichero está en el directorio ../data): PH_train_2.csv
Accuracy: 0.93
```

Figura 3.2: Exactitud del conjunto entrenado y testeado con distintos conjuntos