PUCE Virtual

CLASE

3

PROCESAMIENTO DEL LENGUAJE NATURAL

Fundamentos de la clasificación de amenazas

QS WORLD UNIVERSITY RANKINGS

Educación de calidad

INTRODUCCIÓN DE LA CLASE

GRADO / POSGRADO / TECNOLOGÍAS

Estudia a tu tiempo
son interrupciones

EDUCACIÓN EN LÍNEA DE CALIDAD

En la Semana 3 la asignatura de PLN realiza la transición desde la preparación lingüística del texto hacia el diseño de sistemas inteligentes capaces de tomar decisiones. Después de limpiar, normalizar y representar matemáticamente el lenguaje en la Semana 2, el estudiante dispone ahora de un corpus estructurado listo para el aprendizaje automático. El objetivo central de esta etapa es comprender cómo los modelos de procesamiento de lenguaje natural aprenden a clasificar textos según el tipo de amenaza, utilizando ejemplos etiquetados que permiten identificar patrones recurrentes en comunicaciones maliciosas y legítimas dentro de contextos reales de ciberseguridad.

Esta semana introduce los principios del aprendizaje supervisado aplicado al PLN, comenzando por la definición de etiquetas y clases de amenazas, continuando con la relación entre representación vectorial y desempeño del modelo, y finalizando con uno de los problemas más críticos en seguridad informática: el desbalance de clases. A diferencia de otros dominios, en ciberseguridad los eventos peligrosos suelen ser minoritarios pero de alto impacto, lo que obliga a interpretar el rendimiento del sistema desde una perspectiva de riesgo. Así, la clasificación textual deja de ser únicamente una tarea técnica y pasa a constituir un mecanismo de apoyo a la toma de decisiones, sentando las bases para el desarrollo del sistema inteligente de alerta temprana que culminará en el proyecto final de la asignatura.

Fundamentos de la clasificación de amenazas.

Aprendizaje supervisado en PLN.

Hasta la Semana 2 se preparó el lenguaje para poderlo procesar automáticamente. Ahora comienza el verdadero objetivo del proyecto:
enseñar a la máquina a reconocer amenazas textuales. Esto se logra mediante aprendizaje supervisado.

El Aprendizaje Supervisado es un tipo de aprendizaje automático (Machine Learning en inglés) donde el modelo aprende a partir de ejemplos previamente etiquetados. Cada ejemplo del dataset contiene dos elementos fundamentales:

- **Entrada:** texto procesado (correo, ticket, log, mensaje).
- **Salida esperada:** etiqueta o clase correcta
- "phishing" si el texto corresponde a esta etiqueta.
- "legítimo" si no es amenaza.

El modelo analiza múltiples ejemplos y aprende relaciones estadísticas entre patrones lingüísticos y categorías de amenaza, identificando correlaciones matemáticas entre representaciones vectoriales y etiquetas.

El aprendizaje supervisado requiere dividir el dataset en subconjuntos con funciones distintas. El conjunto de entrenamiento es para enseñar al modelo mediante el ajuste de sus parámetros internos, observando: patrones lingüísticos, combinaciones de palabras, y distribuciones de clases.

El conjunto de prueba se utiliza para evaluar si el modelo realmente aprendió o simplemente memorizó ejemplos. El modelo recibe textos nuevos (que no estuvieron en el conjunto de entrenamiento) y debe clasificarlos correctamente. Esto mide la capacidad de generalización.

Ilustración 1 muestra un ejemplo de implementación en Python de aprendizaje automático:

- **Importa la función de scikit-learn que permite dividir en dos partes:** datos para entrenar (conjunto de entrenamiento) y datos para evaluar (conjunto de prueba). En este caso: train_test_split crea cuatro conjuntos de datos:
    - **X_train:** datos para entrenamiento.
    - **X_test:** datos para evaluar.
    - **Y_train:** etiquetas correctas.
    - **Y_test:** etiquetas reales.

- X_tfidf contiene las características del texto, es la matriz creada durante la vectorización: Texto→TF-IDF→números.
- Las respuestas correctas están en df[“etiqueta”]: phishing y legítimo.
- Random_state: controla la aleatoriedad del proceso, es decir, el dataset se mezcla antes de dividirse para obtener mismos resultados y sean reproducibles.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_tfidf,
    df["etiqueta"],
    test_size=0.2,
    random_state=42
)
```

Ilustración 1. Implementación inicial Python para entrenamiento (creación de autor Rafael Melgarejo)

El flujo del aprendizaje supervisado se resume:

Dataset etiquetado→Vectorización→Entrenamiento→Modelo aprendido→Predicción de nuevas amenazas

Si el dataset está mal dividido puede producir modelos aparentemente excelentes, pero inútiles en la práctica. Se pueden identificar los siguientes tipos de problemas:

- Data Leakage (fuga de información): ocurre cuando el modelo recibe información del conjunto de prueba durante el entrenamiento, es decir recuerda, no entrena. Ejemplo:
  - **Dato de entrenamiento:** “confirme su cuenta urgente”
  - **Dato de prueba:** “confirme su cuenta urgente!!”
- **División aleatoria peligrosa en ciberseguridad:** en otras dominios la separación aleatoria del conjunto de datos funciona bien, pero en ciberseguridad hay riesgos adicionales, como por ejemplo el envío masivo de emails de un mismo atacante: el modelo aprende una campaña específica. Por lo que es aconsejable separar por campaña, fecha y origen del incidente.
- **Desbalance oculto:** si hay un alto porcentaje sea de legítimos en un dataset (y pocos phishing), el modelo obtiene alta precisión sin detectar amenazas. La división estratificada de Ilustración 1 (test_size=0.2, y random_state=42) mantiene proporciones de clases.

- **Preprocesar antes del split:** por ejemplo si se ejecuta: "vectorizer.fit_transform(dataset_completo)", el vocabulario del test ya fue aprendido. Aquí hay fuga de información silenciosa. El procedimiento correcto es: (i) dividir el texto, (ii) ajustar vectorizador solo con datos de entrenamiento, (iii) transformar test después. Ilustración 2 muesta la manera correcta:

```
X_train_raw, X_test_raw, y_train, y_test = train_test_split(...)

vectorizer.fit(X_train_raw)
X_train = vectorizer.transform(X_train_raw)
X_test = vectorizer.transform(X_test_raw)
```

Ilustración 2. Vectorización conjunto de entrenamiento (creación de autor Rafael Melgarejo)

Etiquetas y clases de amenazas.

Un modelo de PLN aprende cuando se le indica qué tipo de evento representa cada mensaje. Las etiquetas (labels) son categorías asignadas por expertos humanos que indican la naturaleza del texto. Las etiquetas convierten datos en conocimiento entrenable.

Taxonomía básica de amenazas textuales.

Actualmente existe una amplia gama de clases de amenazas textuales. La Tabla 1 muestra una taxonomía mínima general basada en el framework de Veris (Verizon,2024).

Tabla 1. Taxonomía de amenazas (basado en Verizon,2024)

Clase	Descripción
phishing	intento de robo de credenciales
malware	descarga o ejecución maliciosa
ingeniería_social	manipulación psicológica
spam_legitimo	mensajes no críticos
incidente_tecnico	problema operativo real

Creación de buenas etiquetas

A menudo se crean etiquetas ambiguas (peligroso, sospechoso, raro). Esto no es operativo en el modelo. Las clases deben ser: claras, mutuamente excluyentes, consistentes y útiles. El aprendizaje supervisado depende directamente del proceso de etiquetado humano que interpreta el mensaje, reconoce la intención y asigna clase.

Ambigüedad semántica

Un mensaje puede pertenecer a más de una categoría, por ejemplo “su cuenta presenta actividad irregular”, puede ser alerta legítima o phishing. Es necesario guías de etiquetado, criterios claros y consistencia entre evaluadores (sí, más de uno).

Buenas prácticas de etiquetado (basado en Eisenstein,2019)

Tome en cuenta las siguientes recomendaciones de etiquetado (Eisenstein,2019):

- Definir guía de anotación: documento que explica cuándo usar cada etiqueta.
- Etiquetas con contexto: no solamente palabras clave.
- Revisar consistencia: evitar decisiones contradictorias.
- Balancear clases: evitar datasets dominados por una sola categoría.

Representación computacional de etiquetas.

Las etiquetas se convierten en valores numéricos, por ejemplo: phishing 0, legítimo 1, malware 2. Ilustración 3 muestra la manera de implementar el etiquetado en Python.

```
from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

y = le.fit_transform(df["etiqueta"])
```

Ilustración 3. Etiquetado a valores numéricos (creación autor Rafael Melgarejo)

Representación vectorial: BoW, TF-IDF, embeddings.

En la Semana 2 se mostró que el texto debe convertirse en números.

En la Semana 3 ocurre algo nuevo: la representación vectorial determina cómo aprende el clasificador. No todas las representaciones producen el mismo desempeño. En ciberseguridad esto es crítico porque: pequeños cambios lingüísticos pueden ocultar ataques, y el modelo depende completamente de cómo representamos el lenguaje. La vectorización es el puente entre el lenguaje y decisión automática:

Texto→Preprocesamiento→Vectorización→Algoritmo de clasificación→Predicción de amenaza

Bag of Words (BoW)

BoW representa cada documento mediante el conteo de palabras, ignorando orden, gramática y contexto. Por ejemplo:

- **Doc1:** confirme su cuenta

- **Doc2:** error de acceso vpn
- **Vocabulario:** [confirmar, cuenta, error, acceso, vpn]
- Representación

Documento	confirmar	cuenta	error	acceso	vpn
Doc1	1	1	0	0	0
Doc2	0	0	1	1	1

La implementación Python en Ilustración 4.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
X_bow = vectorizer.fit_transform(df["texto_limpio"])
```

Ilustración 4. Implementación Conteo Palabras (contenido autor Rafael Melgarejo)

TF-IDF

TF-IDF pondera palabras según su importancia

$$\mathbf{TF - IDF = frecuencia\ local\ x\ rareza\ global}$$

Palabras raras pero informativas reciben mayor peso, por ejemplo: alta importancia (urgente, credenciales, verifique), baja importancia (usuario, mensaje, sistema). Ilustración 5 muestra implementación Python.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
X_tfidf = tfidf.fit_transform(df["texto_limpio"])
```

Ilustración 5. Implementación Python TD-IDF (creación autor Rafael Melgarejo)

Embeddings

BoW y TF-IDF tratan palabras como símbolos independientes. Los embeddings representan palabras según su contexto semántico. El modelo aprende relaciones entre palabras, por ejemplo:

phishing ≈ fraude ≈ suplantación

Algunos tipos comunes de embeddings son: Word2Vec, GloVe, FastText

BERT embeddings. El video “Word2Vec, GloVe, FastText: ¡EXPLICADO!” explica la manera general de funcionamiento de estos tipos: https://www.youtube.com/watch?v=9S0-OC4LFNo

La calidad del modelo depende más de la representación del lenguaje que del algoritmo utilizado. En conclusión: BoW detecta palabras explícitas, TF-IDF detecta mejor vocabulario discriminativo, y Embeddings se lo utiliza para identificar ataques evasivos.

Desbalance de clases y riesgo en ciberseguridad.

En la mayoría de dominios de Machine Learning se asume que las clases están relativamente equilibradas. Esto asumción genera un problema, pues
en ciberseguridad ocurre lo contrario: Los ataques son raros, pero críticos. En otras palabras, se confirma la importancia de analizar si un conjunto de datos está desbalanceado. En seguridad, la mayoría de datos (o casi todos, o todos en el peor de los casos) son tráfico normal, y una clase minoritaria o por aparecer son las amenazas.

El engaño de la precisión (Accuracy Trap)

Si un modelo entrenado con datos desbalanceados (sin advertirlo formalmente por parte del experto humano), de 100 casos, acierta que el 95% son legítimos, y falla en predecir 5 ataques, es posible concluir que el modelo es exitoso porque predice el 95% de casos. Pero, esto es un peligro, pues no detecta ningún ataque.

Perspectiva de riesgo en ciberseguridad

En seguridad, los errores no tienen el mismo costo. Un error no detectado entre miles de aciertos, puede comprometer toda la infraestructura. La consecuencia de error de un falso positivo es una alerta innecesaria, pero de un falso negativo es un ataque exitoso. El falso negativo es muy peligroso. Consecuentemente, en ciberseguridad la métrica “accuracy” no es suficiente.

Estrategias para manejar el desbalance

Los algoritmos intentan minimizar errores totales. Como la clase mayoritaria domina, el modelo aprende a ignorar ataques. Esto se denomina: bias hacia la clase mayoritaria. A continuación 4 estrategias para manejar el desbalance:

- **División estratificada:** mantener proporciones entre los datos de entrenamiento (train) y los de prueba (test). A continuación el código Python:
  train_test_split(…, stratify=y)
- **Rebalanceo de datos:** aquí pueden ocurrir dos cosas:
  - **Oversampling:** duplicar ejemplos minoritarios.
  - **Undersampling:** reducir clase mayoritaria.
- **Pesos de clase:** indicar al modelo que los ataques son más importantes. La implementación en Python es de la siguiente manera:
  class_weight=”balanced”

- **Métricas orientadas a riesgo:** evaluar con Recall, F1 y Matriz de Confusión.

Métricas más importantes en ciberseguridad

Las métricas más importantes en ciberseguridad son Recall y F1-score. Tabla 2 muestra las métricas, su objetivo, importancia y fórmula. Se añade Precision o Accuracy como contraste.

Tabla 2. Métricas de Evaluación en Clasificación de Amenazas (Jurafsky,2023)

Métrica	Pregunta que responde	Importancia en Ciberseguridad	Fórmula
Precision	¿Cuántas alertas eran realmente ataques?	Reduce falsas alarmas	TP / (TP + FP)
Recall	¿Cuántos ataques reales detectamos?	Evita ataques no detectados	TP / (TP + FN)
F1-score	Balance entre precision y recall	Evaluación integral	2·(P·R)/(P+R)

Las fórmulas se basan en:

- **TP:** Verdaderos positivos (ataques detectados correctamente)
- **FP:** Falsos positivos (alertas falsas)
- **FN:** Falsos negativos (ataques no detectados)

El sistema ideal puede generar más alertas, pero evita compromisos críticos.

Ilustración 6 muestra el código Python de una implementación directa de métricas, importándolas desde “sklearn”. El resultado de Precision, Recall y F1-score es sobre la base delos datos supuestos de y_test y y_pred, indica que todas las alertas fueron correctas (1.0), pero algunos ataques no fueron detectados.

```python
from sklearn.metrics import precision_score, recall_score, f1_score

#Salida supuesta
# Etiquetas reales
y_test = ["phishing", "legitimo", "phishing", "legitimo", "phishing"]

# Predicciones del modelo
y_pred = ["phishing", "legitimo", "legitimo", "legitimo", "phishing"]

#Cálculo de métricas
precision = precision_score(y_test, y_pred, pos_label="phishing")
recall = recall_score(y_test, y_pred, pos_label="phishing")
f1 = f1_score(y_test, y_pred, pos_label="phishing")

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Precision: 1.0
Recall: 0.6666666666666666
F1-score: 0.8
```

Ilustración 6. Implementación de Métricas (creación de autor Rafael Melgarejo)

Las métricas Precision, Recall y F1-score utilizadas en la asignatura se fundamentan en estándares internacionales de evaluación de sistemas de clasificación descritos en Jurafsky & Martin (2023), Manning et al. (2008).

Una versión profesional de calcular métricas presenta Ilustración 7. En este caso, toma los mismos datos del código que está en Ilustración 6, pero utiliza el reporte de clasificación.

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

     legitimo       0.67      1.00      0.80         2
     phishing       1.00      0.67      0.80         3

     accuracy                           0.80         5
    macro avg       0.83      0.83      0.80         5
 weighted avg       0.87      0.80      0.80         5
```

Ilustración 7. Presentación de métricas con reporteador (creación de autor Rafael Melgarejo)

La interpretación de la columna "support" es la siguiente:

- "support" para la clase 'legitimo' es 2, lo que significa que había 2 instancias legítimas en los datos de prueba (y_test).
- support para la clase 'phishing' es 3, lo que significa que había 3 instancias de phishing en los datos de prueba (y_test).

Es útil para entender cuántos ejemplos de cada clase contribuyen a las métricas de precisión, recall y f1-score.

Matriz de confusión

Matriz de confusión es una herramienta para evaluar modelos de aprendizaje automático supervisado que muestra, en forma de tabla, el rendimiento de un algoritmo de clasificación comparando los valores reales con los predichos. Datademia explica en el video ¿Qué es una matriz de confusión?, en el siguiente enlace: https://www.youtube.com/watch?v=jcBL6jxec9E.

A continuación, se muestra visualmente los componentes y su distribución en la Matriz de Confusión: verdaderos positivos y negativos, así como los falsos positivos y negativos. La primera columna es la Predicción Positiva, y la primera fila el dato actual.

		Predicción	
		Positivo	Negativo
Actual	Positivo	Verdadero Positivo	Falso Negativo
	Negativo	Falso Positivo	Verdadero Negativo

La Ilustración 8 muestra el código Python que implementa la matriz de confusión (Confusion Matrix), importando la librería indicada desde metrics de sklearn. Utiliza el mismo conjunto de datos del código indicado en Ilustración 6.

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[2 0]
 [1 2]]
```

Ilustración 8. Implementación Matriz de Confusión (creación de autor Rafael Melgarejo)

La matriz de confusión [[2, 0], [1, 2]] se interpreta de la siguiente manera, asumiendo que el orden de las clases es 'legitimo' y luego 'phishing':

- **Verdaderos Negativos (TN):** 2 casos que eran realmente 'legítimo' y fueron correctamente clasificados como 'legítimo'.
- **Falsos Positivos (FP):** 0 casos que eran realmente 'legítimo' pero fueron incorrectamente clasificados como 'phishing'. (No hubo errores de este tipo)
- **Falsos Negativos (FN):** 1 caso que era realmente 'phishing' pero fue incorrectamente clasificado como 'legítimo'.
- **Verdaderos Positivos (TP):** 2 casos que eran realmente 'phishing' y fueron correctamente clasificados como 'phishing'.

  En otras palabras: El modelo clasificó correctamente 2 emails legítimos y 2 emails de phishing. El modelo falló al clasificar 1 email de phishing, etiquetándolo como legítimo. Como resumen completo de la semana: en ciberseguridad la precisión del modelo no significa seguridad, y un recall alto si puede interpretarse como un sistema seguro. La clasificación automática no es solo un problema técnico, es un problema de gestión del riesgo.

En esta semana se analizó la distribución de clases, se justifica las métricas usadas, y su explica el impacto del error en ciberseguridad.

Educación de calidad

REFERENCIAS

Eisenstein, J. (2019). Introduction to natural language processing. MIT Press. https://cdn.jsdelivr.net/gh/it-ebooks-0/it-ebooks-2018-04to07/Natural%20Language%20Processing%20%28Jacob%20Eisenstein%29.pdf

Jurafsky, D., & Martin, J. H. (2023). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition (3rd ed., draft). Stanford University. https://web.stanford.edu/~jurafsky/slp3/.

Manning, C. D., Raghavan, P., & Schütze, H. (2008).
Introduction to information retrieval. Cambridge University Press.
https://nlp.stanford.edu/IR-book/

Verizon. (2024). Data breach investigations report. https://www.verizon.com/business/resources/reports/dbir/

PUCE Virtual

síguenos en:

www.pucevirtual.puce.edu.ec