PUCE Virtual

CLASE

4

PROCESAMIENTO DEL LENGUAJE NATURAL

Construcción del Clasificador

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 4 la asignatura avanza desde la comprensión conceptual del aprendizaje supervisado hacia la implementación práctica de clasificadores de texto capaces de identificar amenazas en comunicaciones digitales. Después de preparar el corpus (Semana 2) y comprender los fundamentos de la clasificación y las etiquetas de amenaza (Semana 3), el estudiante comienza ahora a construir modelos completos de procesamiento de lenguaje natural que integran representación vectorial, algoritmos de aprendizaje automático y métricas de evaluación. Esta etapa marca el inicio del desarrollo técnico del sistema de detección automática que constituye el núcleo del proyecto de la asignatura.

Durante esta semana se introducen los componentes esenciales de un clasificador funcional: la vectorización del texto mediante TF-IDF y embeddings, la correcta división de los datos en conjuntos de entrenamiento, validación y prueba, y el uso de algoritmos de clasificación comunes en PLN como Naive Bayes, máquinas de soporte vectorial (SVM) y redes neuronales simples. Asimismo, se analizan métricas de evaluación orientadas al riesgo en ciberseguridad y el problema del sobreajuste, junto con técnicas como la validación cruzada que permiten evaluar de manera confiable el desempeño de los modelos. En conjunto, estos elementos permiten que el estudiante comprenda cómo diseñar, entrenar y evaluar un sistema de clasificación de amenazas basado en texto.

Construcción del Clasificador

Vectorización TF-IDF y embeddings

En la construcción de un clasificador funcional, la vectorización deja de ser un paso conceptual y se convierte en la entrada directa del modelo de aprendizaje automático.
Un clasificador recibe vectores numéricos que representan el contenido lingüístico del texto. Por ello, la calidad de la representación vectorial influye directamente en la capacidad del sistema para identificar patrones asociados a amenazas de ciberseguridad.
En esta etapa del curso se utilizan dos tipos de representaciones ampliamente empleadas en sistemas de procesamiento de lenguaje natural:

- TF-IDF, que pondera la importancia estadística de las palabras.
- Embeddings, que representan relaciones semánticas entre términos y documentos.

Vectorización mediante TF-IDF

TF-IDF (Term Frequency – Inverse Document Frequency) es una técnica que representa documentos mediante el peso de cada término según:

- su frecuencia en el documento,
- su rareza en el conjunto de documentos.

La idea central es que las palabras más informativas para distinguir textos reciben mayor peso. La siguiente es la fórmula conceptual de la técnica TF-IDF:

$$\mathbf{TF - IDF = TF * IDF}$$

donde:

- TF (Term Frequency) mide cuántas veces aparece un término en un documento.
- IDF (Inverse Document Frequency) reduce el peso de palabras muy comunes en el corpus.

Considerando los dos siguientes textos: “verifique su cuenta inmediatamente”, y “error de autenticación de vpn”. Las palabras “verifique”, “credenciales” y “suspensión”, tienden a a tener mayor peso en mensajes de phishing, mientras que términos como usuario o sistema aparecen en muchos contextos y reciben menor ponderación. Ilustración 1 con la implementación Python básica:

```python
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

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

Ilustración 1. Implementación Python TF-IDF (contenido de autor Rafael Melgarejo)

Al ejecutar el código Python se espera una matriz documento-término, en la cual:

- Cada fila representa un documento,
- Cada columna representa una palabra del vocabulario

Ventajas de TF-IDF en la clasificación de amenazas

TF-IDF se utiliza ampliamente en clasificación de textos porque:

- reduce el impacto de palabras muy frecuentes,
- resalta términos discriminativos,
- funciona bien con modelos clásicos de aprendizaje automático.

En muchos sistemas industriales de detección de phishing o spam, TF-IDF sigue siendo una representación extremadamente efectiva. Se recomienda ver el video “TF-IDF: Vectorización de Documentos para NLP con Python” https://www.youtube.com/watch?v=4QsG4oNcGpM con ejemplos prácticos.

Embeddings: representación semántica del lenguaje

Los embeddings representan palabras según su contexto semántico. Esto significa que palabras con significados similares se ubican cerca en el espacio vectorial. En Ilustración 2 un ejemplo conceptual

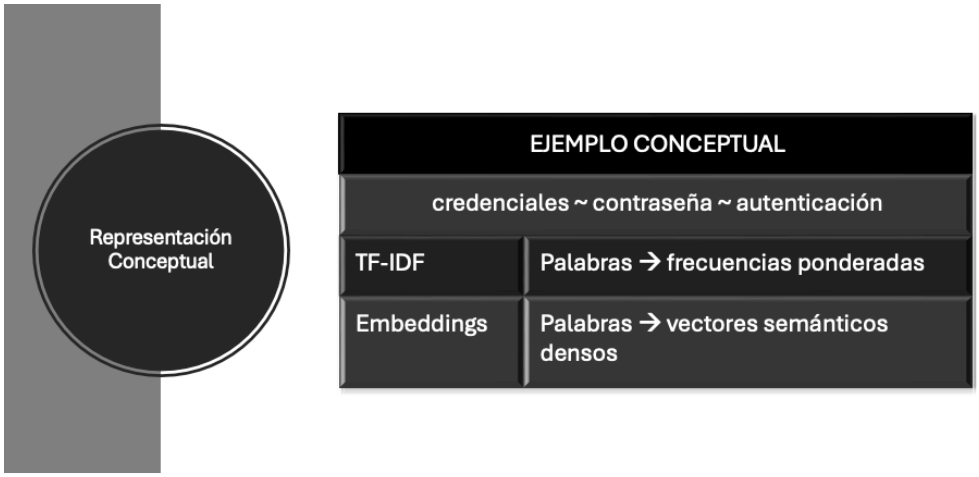

Ilustración 2. Representación Semántica del Lenguaje (creación de autor Rafael Melgarejo)

Embeddings a nivel de documento

En clasificación de texto también se construyen embeddings de documentos completos que representan el significado global de un mensaje.

Esto permite detectar patrones como:

- intención de manipulación,
- coerción lingüística,
- suplantación de autoridad.

Este tipo de representación resulta especialmente útil para detectar ataques evasivos que no usan palabras clave explícitas.

Comparación entre TF-IDF y embeddings

Existen varias características a tomar en cuenta al decidir entre TF-IDF y Embeddings. La Tabla 1 muestra las diferencias (Jurafsky et al 2023).

Tabla 1. TF-IDF vs Embeddings (adaptado de (Jurafsky et al 2023))

Característica	TF-IDF	Embeddings
Tipo de representación	Estadística	Semántica
Dimensión del vector	Alta (vocabulario)	Baja o media
Interpretabilidad	Alta	Media
Capacidad semántica	Limitada	Alta

Conjuntos train/validation/test.

En aprendizaje automático, un modelo debe evaluarse en condiciones que simulen su uso real. Si un modelo se entrena y se evalúa utilizando los mismos datos, puede parecer muy preciso, pero en realidad puede estar memorizando ejemplos específicos en lugar de aprender patrones generales del lenguaje. Para evitar este problema, el conjunto de datos se divide en tres subconjuntos con funciones distintas:

- Training set (entrenamiento)
- Validation set (validación)
- Test set (prueba)

Esta separación permite evaluar si el modelo realmente generaliza a datos nuevos, lo cual es fundamental en sistemas de detección de amenazas donde los ataques cambian continuamente. En sistemas de detección de amenazas, la correcta separación de datos es crítica. Los ataques evolucionan constantemente, por lo que el modelo debe evaluarse en datos que representen situaciones nuevas y desconocidas. Un sistema evaluado

incorrectamente puede generar una falsa sensación de seguridad y permitir que ataques reales pasen desapercibidos.

El conjunto de entrenamiento (training set)

El training set contiene los datos que el modelo utiliza para aprender. Durante el entrenamiento, el algoritmo analiza:

- las representaciones vectoriales del texto,
- las etiquetas asociadas a cada ejemplo,
- las relaciones estadísticas entre patrones lingüísticos y clases de amenaza.

El objetivo del entrenamiento es ajustar los parámetros del modelo para minimizar errores en la clasificación. El modelo aprende patrones a partir de muchos ejemplos de etiquetado **(e.g.: "verifique su cuenta inmediatamente"→phishing; "error de autenticación"→incidente técnico).** Una vez aprendido (identificado el patrón) lo aplicará a textos nuevos.

El conjunto de validación (validation set)

El validation set se utiliza para ajustar el modelo durante el proceso de desarrollo. Mientras el modelo se entrena, se toman decisiones como:

- elegir el algoritmo más adecuado,
- ajustar hiperparámetros,
- seleccionar la representación vectorial más efectiva.

Estas decisiones deben evaluarse utilizando datos diferentes del entrenamiento para evitar que el modelo se adapte excesivamente a los ejemplos usados para aprender.

Por esta razón, el conjunto de validación actúa como un simulador de nuevos datos durante el proceso de construcción del modelo.

El conjunto de prueba (test set)

El test set se utiliza únicamente al final del proceso para evaluar el desempeño real del modelo. Los datos del conjunto de prueba:

- no deben utilizarse durante el entrenamiento,
- no deben utilizarse durante el ajuste de hiperparámetros,
- deben permanecer completamente separados hasta la evaluación final.

Esto permite estimar con mayor precisión cómo se comportará el sistema en producción.

Distribución típica de los datos

Una división común del dataset es:

- **Training:** 70-80%, para aprendizaje del modelo
- **Validation:** 10-15%, para ajuste de hiperparámetros
- **Test:** 10-15%, para evaluación final

La proporción exacta depende del tamaño del dataset disponible.

Problema crítico: fuga de información (data leakage)

Uno de los errores más graves en aprendizaje automático ocurre cuando el modelo recibe indirectamente información del conjunto de prueba durante el entrenamiento. Esto puede suceder si:

- se vectoriza todo el dataset antes de dividirlo,
- se usan estadísticas calculadas con datos de prueba,
- se reutilizan ejemplos similares en entrenamiento y prueba.

Este fenómeno se conoce como fuga de información (data leakage). Cuando ocurre, el modelo parece tener un desempeño excelente, pero falla cuando se enfrenta a datos reales.

Implementación básica en Python

Una forma común de dividir los datos es utilizar la función train_test_split de Scikit-learn (ver Ilustración 3).

```
from sklearn.model_selection import train_test_split

X_train, X_temp, y_train, y_temp = train_test_split(
    X_tfidf,
    df["etiqueta"],
    test_size=0.3,
    stratify=df["etiqueta"],
    random_state=42
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp,
    y_temp,
    test_size=0.5,
    stratify=y_temp,
    random_state=42
)
```

Ilustración 3. Implementación Python de división de datos (creación de autor Rafael Melgarejo)

En este ejemplo: 70 % de los datos se destinan al entrenamiento, 15 % a validación,

15 % a prueba. El parámetro stratify mantiene la misma proporción de clases en cada subconjunto.

Algoritmos: Naive Bayes, SVM, redes simples (adaptado de (Eisenstein,2019)).

Una vez que el texto ha sido transformado en vectores numéricos y el dataset dividido correctamente en conjuntos de entrenamiento, validación y prueba, el siguiente paso en la construcción del sistema es seleccionar el algoritmo de clasificación.

El algoritmo es el componente del sistema que aprende la relación entre las representaciones vectoriales del texto y las etiquetas de amenaza. Es el mecanismo que permite que el sistema tome decisiones automáticas sobre nuevos mensajes.

En procesamiento de lenguaje natural, varios algoritmos han demostrado ser efectivos para la clasificación de textos. Se introducen tres enfoques ampliamente utilizados:

- Naive Bayes
- Máquinas de soporte vectorial (SVM)
- Redes neuronales simples

Cada uno presenta diferentes ventajas en términos de interpretabilidad, capacidad de modelado y complejidad computacional.

Clasificador Naive Bayes

El clasificador Naive Bayes es uno de los algoritmos más utilizados en clasificación de textos debido a su simplicidad y eficiencia. Este modelo se basa en el teorema de Bayes, que calcula la probabilidad de una clase dada la presencia de ciertas características del texto.

La idea central es estimar:

$$P(clase|palabras)$$

Es decir, la probabilidad de que un texto pertenezca a una categoría determinada considerando las palabras que contiene.

Supuesto de independencia: El término “naive” (ingenuo) proviene del supuesto de que las palabras del documento son estadísticamente independientes entre sí. Aunque esta suposición no es completamente realista, en la práctica el modelo funciona sorprendentemente bien en tareas de clasificación textual. Por ejemplo, si el texto recibido es **“verifique su cuenta inmediatamente”,** el modelo calcula que la probabilidad de que sea phishing es 0.91, es decir lo clasifica como phishing.

En Python, en sklearn, hay una librearía naive_bayes. En el ejemplo siguiente (Ilustración 4), se usa MultinomialNB para dos conjuntos de entrenamiento y predicción de uno. Esta implementación funciona tanto con TF-IDF o BoW, requiriendo pocos datos para entrenar.

```
from sklearn.naive_bayes import MultinomialNB

modelo = MultinomialNB()

modelo.fit(X_train, y_train)

predicciones = modelo.predict(X_test)
```

Ilustración 4. Implementación Python NB (creación de autor Rafael Melgarejo)

Explicación clara y con ejemplos puede verse en “El Teorema de Bayes ¡Explicado!”, https://www.youtube.com/watch?v=3vtS2qc4t6o

Máquinas de soporte vectorial (SVM)

Support Vector Machines son algoritmos que buscan encontrar una frontera de decisión que separe diferentes clases de documentos. En el espacio vectorial del lenguaje, cada documento es un punto en un espacio multidimensional.

El objetivo del SVM es encontrar un hiperplano que divida ese espacio de forma óptima. En el hiperplano (espacial de datos de entrenamiento), el algoritmo busca maximizar la distancia entre el hiperplano y los puntos más cercanos de cada clase (phishing o legítimo), quedando en la frontera los que no están o no puede clasificar.

La implementación en Python toma de sklearn la librería svm (Ilustración 5).

```
from sklearn.svm import LinearSVC

modelo = LinearSVC()

modelo.fit(X_train, y_train)

predicciones = modelo.predict(X_test)
```

Ilustración 5. Implementación Pyton SVM (creación de autor Rafael Melgarejo)

Redes neuronales simples

Son modelos inspirados en el funcionamiento del cerebro humano. En su forma más simple, consisten en capas de neuronas artificiales que transforman las entradas mediante funciones matemáticas. La estructura básica de una red neuronal es de tres capas. La

primera es la entrada de datos de prueba, luego la capa oculta o donde se establecen relaciones entre neuronas, y la tercera o capa de salida que predice.

En clasificación de textos, una red neuronal simple puede aprender relaciones más complejas entre características del texto y clases de amenaza.

Ilustración 6 muestra la implementación Python.

```
from sklearn.svm import LinearSVC

modelo = LinearSVC()

modelo.fit(X_train, y_train)

predicciones = modelo.predict(X_test)
```

Ilustración 6. Implementación básica de RN (creación de autor Rafael Melgarejo)

Comparación y selección de algoritmos

En sistemas de clasificación de amenazas textuales, la elección del algoritmo depende de varios factores:

- tamaño del dataset
- tipo de representación vectorial
- recursos computacionales
- interpretabilidad requerida

En muchos casos, Naive Bayes y SVM ofrecen resultados muy competitivos con menor complejidad que redes neuronales profundas. La comparación de algoritmos basada en (Jurafsky et al, 2023).

Algoritmo	Ventaja Principal	Limitación
Naive Bayes	Rápido y simple	Suposición de independencia
SVM	Alta precisión en texto	Mayor costo computacional
Redes Neuronales	Modela relaciones complejas	Requiere más datos

Métricas orientadas a riesgo.

Una vez entrenado un clasificador de amenazas textuales, es necesario evaluar su desempeño para determinar si el sistema es confiable. En ciberseguridad no basta con medir únicamente la precisión global del modelo (accuracy). Medir el porcentaje total de predicciones correctas, puede resultar engañosa cuando el conjunto de datos está desbalanceado, situación común en sistemas de seguridad informática donde los eventos maliciosos representan una fracción pequeña del total de mensajes analizados.

Matriz de confusión

La evaluación de un modelo de clasificación comienza con la matriz de confusión, que muestra cómo se distribuyen las predicciones correctas e incorrectas (Jurafsky et al, 2023).

	Predicción: Ataque	Predicción: NO Ataque.
Ataque real	TP: Verdadero Positivo	FN: Falso Negativo
NO ataque	FP: Falso Positivo	TN: Verdadero Negativo

Cada uno de estos resultados tiene implicaciones distintas en un sistema de ciberseguridad.

- **TP:** detecta correctamente un ataque.
- **FP:** genera una alerta sobre un evento que no es una amenaza.
- **FN:** no detecta un ataque real.
- **TN:** identifica correctamente un evento benigno.

Entre estos errores, los falsos negativos suelen representar el mayor riesgo, ya que implican ataques que pasan desapercibidos.

Precision

La precision mide la proporción de alertas generadas por el sistema que corresponden realmente a amenazas.

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

Esta métrica evita una gran cantidad de alertas falsas que saturen a los analistas de seguridad. Un sistema con baja precision produce muchas alertas incorrectas, lo que puede provocar fatiga de alertas en los equipos SOC.

Recall (sensibilidad)

El recall mide la proporción de ataques reales que el sistema logra detectar.

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

En ciberseguridad, el recall suele ser una métrica prioritaria, ya que indica la capacidad del sistema para identificar amenazas reales. Un recall bajo significa que muchos ataques están pasando inadvertidos. En un entorno de ciberseguridad, las métricas deben interpretarse

considerando el impacto de cada tipo de error. Un Falso positivo genera trabajo adicional para los analistas, mientras un Falso negativo permite que un ataque ocurra sin detección. Por esta razón, muchos sistemas de detección priorizan recall alto, incluso si ello implica un número mayor de falsas alarmas (Jurafsky et al, 2023).

F1-score

El F1-score combina precision y recall en una única métrica que refleja el equilibrio entre ambas.

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

Esta métrica es particularmente útil cuando el dataset está desbalanceado y se necesita evaluar simultáneamente la capacidad del sistema para detectar ataques y evitar falsas alarmas.

Métricas en clasificación multiclase

Cuando el sistema clasifica múltiples tipos de amenazas (por ejemplo phishing, malware o ingeniería social), las métricas se calculan para cada clase. Adicionalmente, se utilizan promedios para evaluar el desempeño global del modelo, por ejemplo:

- Macro average: promedio simple de todas las clases
- Weighted average: promedio ponderado según el número de ejemplos por clase

El promedio macro es útil para analizar el desempeño cuando las clases están desbalanceadas.

```
from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)

print(cm)
```

Ilustración 7. Implentación de métricas en Python (creación de autor Rafael Melgarejo)

Implementación básica en Python

La biblioteca Scikit-learn permite calcular estas métricas de forma sencilla, usando “classification report, incluye: precision, recall, F1-score, y soporte (número de ejemplos por clase); y, “confusion_matrix” permite visualizar los errores del modelo (Ilustración 7).

Sobreajuste y validación cruzada.

Overfitting

A veces el modelo puede aprender demasiado bien los ejemplos específicos del conjunto de entrenamiento y perder la capacidad de generalizar a datos nuevos. Esto es sobreajuste. Un modelo sobreajustado memoriza los datos de entrenamiento en lugar de aprender relaciones que se mantengan en situaciones diferentes.

El sobreajuste suele identificarse comparando el desempeño del modelo en diferentes conjuntos de datos. En el training_set, el desempeño del modelo es alto, mientras en el test_set es considerablemente menor. Esto indica que el modelo ha aprendido patrones específicos del conjunto de entrenamiento que no se repiten en datos nuevos.
Las técnicas usuales para evitar que el modelo memorice datos son:

- aumentar el tamaño del dataset,
- reducir el número de características,
- regularizar el modelo,
- utilizar validación cruzada.

Cross-validation

La validación cruzada es una técnica que permite evaluar el desempeño del modelo utilizando múltiples divisiones del dataset. En lugar de dividir los datos una sola vez en entrenamiento y prueba, el dataset se divide en k subconjuntos.
El modelo se entrena y evalúa varias veces utilizando combinaciones distintas de estos subconjuntos. En la validación cruzada de tipo k-fold, el procedimiento es el siguiente:

- El dataset se divide en k partes iguales.
- En cada iteración, una de las partes se utiliza como conjunto de validación.
- Las partes restantes se utilizan para entrenamiento.
- El proceso se repite k veces.

Ilustración 8 muestra la implementación en Python.

```python
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB

modelo = MultinomialNB()

scores = cross_val_score(modelo, X_tfidf, y, cv=5)

print("Scores:", scores)
print("Promedio:", scores.mean())
```

Ilustración 8. Cross-Validation (creación de autor Rafael Melgarejo)

Educación de calidad

REFERENCIAS

GRADO / POSGRADO / TECNOLOGÍAS

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/.

PUCE Virtual

síguenos en:

www.pucevirtual.puce.edu.ec