Entrega final

Responsables:

Daniel Quintero Álvarez

Profesor:

Raul Ramos Pollan

Universidad de Antioquia

Introducción a la inteligencia artificial

2023/2

1. Planteamiento del problema

El enfoque se centra en la identificación y clasificación de páginas dedicadas al phishing, una forma de estafa en línea. El concepto de anti-phishing abarca las estrategias para prevenir estos ataques. El phishing constituye un delito cibernético en el cual los estafadores simulan ser entidades confiables, contactando a individuos a través de diversos medios como correos electrónicos, mensajes de texto o llamadas telefónicas, con el propósito de obtener información confidencial. En general, los correos electrónicos fraudulentos suelen alertar sobre problemas en facturas, actividades sospechosas en cuentas, o solicitar iniciar sesión para verificar información. Asimismo, los estafadores pueden requerir detalles de tarjetas de crédito, información bancaria u otros datos personales sensibles. Una vez obtenida esta información, los estafadores pueden acceder a cuentas, robar datos e identidades, o incluso introducir malware en el dispositivo del usuario. Por consiguiente, resulta crucial implementar medidas de seguridad sólidas para prevenir los ataques de phishing y resguardar la privacidad y seguridad en línea de los usuarios.

1.2.Dataset

El dataset seleccionado es Phishing website dataset (https://www.kaggle.com/datasets/akashkr/phishing-website-dataset/data) Este conjunto de datos contiene 30 atributos extraídos de 11000 páginas.

Algunos de los atributos más significativos son:

- Iframe contains: [1 -1]
- age_of_domain contains: [-1 1]
- DNSRecord contains: [-1 1]
- web_traffic contains: [-1 0 1]
- Page_Rank contains: [-1 1]
- Google_Index contains: [1 -1]
- Links_pointing_to_page contains: [1 0 -1]
- Statistical_report contains: [-1 1]
- Result contains [-1 1]

Según la descripción de los datos, estos son el significado de los valores en los datos.

1 significa legítimo

0 es sospechoso

-1 es phishing

1.3. Métricas

Se emplearán dos métricas principales para evaluar el rendimiento del sistema: la precisión (accuracy) y el puntaje F1, ya que ambas se enfocan en la exactitud. Junto a estas métricas técnicas, se incorpora la evaluación desde una perspectiva empresarial: la fiabilidad de las predicciones para determinar si una página presenta o no actividades de phishing. Resulta fundamental que estas predicciones sean confiables, dado que permitirán al navegador web evitar que sus usuarios accedan a sitios web perjudiciales.

1.4. Desempeño

En un modelo de esta naturaleza, se aspira a lograr una precisión de predicción superior al 80%, mientras se busca minimizar considerablemente la ocurrencia de falsos positivos. En un entorno operativo, este modelo se emplearía como un filtro para prevenir el acceso de los usuarios a sitios web sospechosos, garantizando así su seguridad. Por consiguiente, resulta fundamental que el modelo sea capaz de generar predicciones precisas y confiables para cumplir eficazmente con este propósito.

2. Exploración descriptiva del dataset

El proyecto utiliza la base de datos conocida como "Phishing Website Dataset", la cual comprende 11,055 registros y 32 columnas. De estas columnas, 31 se enfocan en detallar las características específicas de las URL de los sitios web, mientras que la restante, llamada "Resultado", señala si el sitio web en cuestión es o no un sitio de phishing.

La información alojada en estas 30 variables descriptivas es fundamental para identificar patrones y atributos característicos en los sitios web que practican el phishing. Al analizar detenidamente estas variables, se pueden desarrollar modelos de aprendizaje automático capaces de detectar de manera precisa y efectiva la presencia de sitios web de phishing, brindando así una capa de protección a los usuarios contra posibles ataques.

Las columnas presentes en el conjunto de datos se denominan así:

0	index
1	having_IPhaving_IP_Address
2	URLURL_Length
3	Shortining_Service
4	having_At_Symbol
5	double_slash_redirecting
6	Prefix_Suffix
7	having_Sub_Domain
8	SSLfinal_State
9	Domain_registeration_length
10	Favicon
11	port
12	HTTPS_token
13	Request_URL
14	URL_of_Anchor
15	Links_in_tags
16	SFH
17	Submitting_to_email
18	Abnormal_URL
19	Redirect
20	on_mouseover
21	RightClick
22	popUpWidnow
23	Iframe
24	age_of_domain
25	DNSRecord
26	web_traffic
27	Page_Rank
28	Google_Index
29	Links_pointing_to_page
30	Statistical_report
31	Result

Además, se ha optado por excluir la columna "INDEX" en el transcurso del análisis, dado que esta variable no aporta relevancia al propósito del estudio.

La columna "INDEX" comúnmente representa un identificador único asignado a cada muestra en un conjunto de datos. En la mayoría de los casos, esta variable no ejerce influencia alguna en la predicción ni en el análisis de los datos; simplemente es un número de referencia utilizado para identificar cada muestra en la base de datos.

Por consiguiente, la supresión de la columna "INDEX" no incidirá en la calidad de los resultados y puede simplificar el análisis al reducir el número de variables en el conjunto de datos. Este procedimiento también podría contribuir a mejorar el rendimiento y la eficacia de los modelos de aprendizaje automático, al tener menos variables para analizar. En general, eliminar variables irrelevantes es una práctica común en la etapa de preprocesamiento de datos, con el objetivo de optimizar la precisión y eficiencia en el análisis de la información.

Elimación de la columna Index

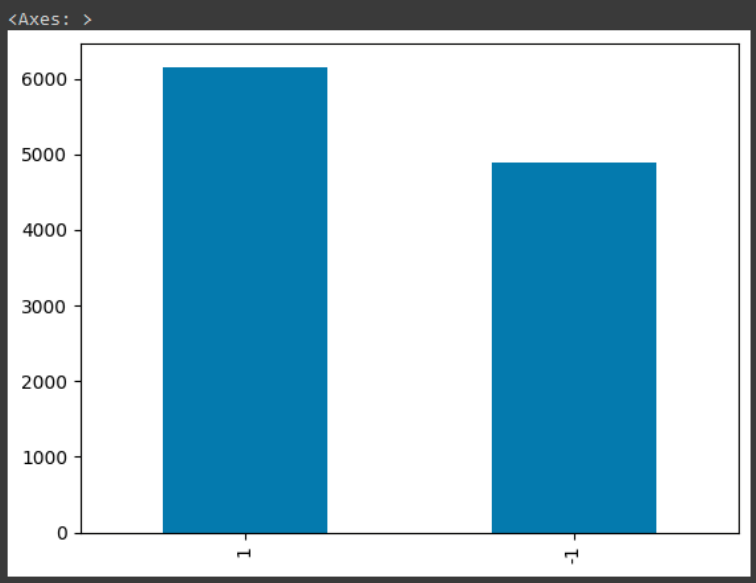
```
[ ]  del data["Index"]
```

2.1 Balanceo de los datos

Tras examinar detenidamente el conjunto de datos, se constató la ausencia de desequilibrio en los mismos. La base de datos contiene un total de 10,000 muestras, distribuidas en 6,157 muestras para la clase 1 (sin phishing) y 4,898 muestras para la clase -1 (phishing).

Balanceo de Result

```
[14] data['Result'].value_counts().plot(kind='bar')
```



Tras el análisis de balanceo, se determina que el conjunto de datos exhibe una distribución bastante equilibrada entre sus dos clases. La clase 1 (sin phishing) representa el 61.57% de las muestras, mientras que la clase -1 (phishing) abarca el 48.98%. Esto indica la ausencia de una clase dominante o minoritaria que pudiera sesgar los resultados del modelo. Un equilibrio en el conjunto de datos es favorable para que el modelo pueda aprender a distinguir adecuadamente entre ambas clases sin ser influenciado por un desequilibrio.

El dataset se dividió en dos bloques destinados al entrenamiento y validación del modelo. Durante la observación, se identificó la mejor precisión (accuracy) con una profundidad de árbol igual a 10.

```
[ ] resultados_dt = experimentar_dt([3,10,50,100],MinMaxScaler().fit_transform(X), Y)
    resultados_dt
```

	profundidad del arbol	eficiencia de entrenamiento	desviacion estandar entrenamiento	eficiencia de prueba	desviacion estandar prueba	accuracy
0	3.0	0.908227	0.001163	0.904029	0.011213	0.904029
1	10.0	0.958571	0.001633	0.944277	0.006502	0.944277
2	50.0	0.989889	0.000318	0.961458	0.020049	0.961458
3	100.0	0.989889	0.000318	0.962091	0.019774	0.962091

Se están llevando a cabo pruebas eliminando datos de manera aleatoria del conjunto de datos.

```
remove_n = 3000
drop_indices = np.random.choice(data.index, remove_n, replace=False)
df_subset = data.drop(drop_indices)

X = df_subset.drop('Result', axis=1).values
Y = df_subset['Result'].values
print (X.shape , Y.shape)
```

3. Algoritmos de predicción

Se han escogido los algoritmos Decision Tree y Random Forest para entrenar los datos respectivos y crear modelos predictivos.

3.1 hiper parámetros para dos algoritmos predictivos

Con el propósito de encontrar los mejores hiperparámetros para los algoritmos predictivos elegidos, como Decision Tree y Random Forest, se realizó un proceso iterativo en cada pliegue del modelo. Esto posibilitó examinar y evaluar la eficacia tanto en el entrenamiento como en las pruebas para cada configuración. Además, se empleó la técnica de validación cruzada K-Fold para garantizar una distribución equitativa de los datos entre el conjunto de entrenamiento y el de pruebas.

3.1.1 Decision tree

Para este algoritmo en particular, se investigó el hiperparámetro de la profundidad del árbol, probando con distintos valores (3, 10, 20 y 50).

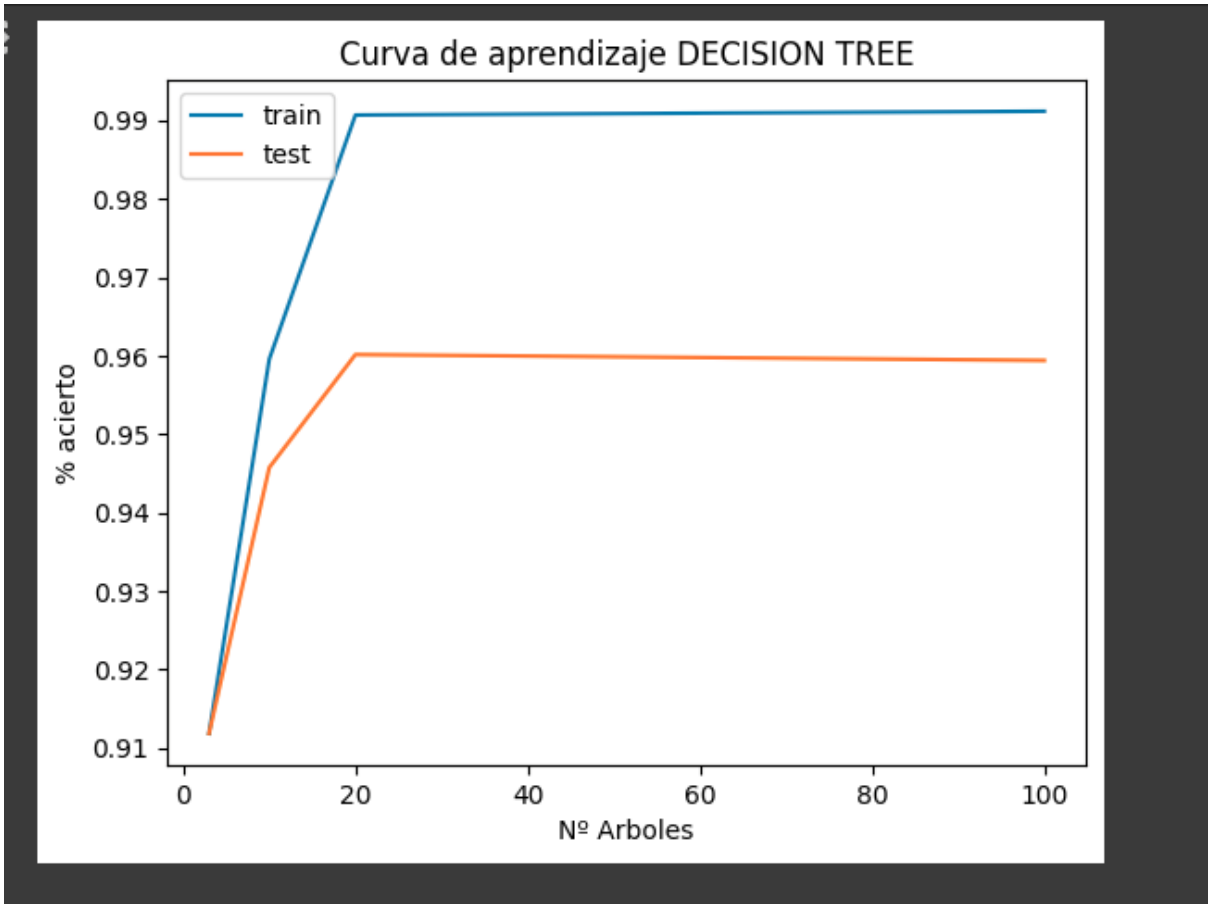
A continuación, se adjunta una imagen que representa el código empleado para identificar los mejores hiperparámetros en el algoritmo Decision Tree:

```python
def experimentar_dt(depths,X, Y):
    """funcion que realiza experimentos de arboles de decision
    depths: list[int] lista con la profundidad de arboles a experimentar
    normalize bool: indica si se aplica normalización a los datos
    X: matriz con las caractersiticas
    Y: matriz de numpy con etiquetas
    retorna: dataframe con:
        - profunidad de los arboles
        - eficiencia de entrenamiento
        - desviacion de estandar eficiencia de entrenamiento
        - eficiencia de prueba
        - desviacion estandar eficiencia de prueba
    """
    folds = 10
    skf = KFold(n_splits=folds)
    resultados = pd.DataFrame()
    idx = 0
    for depth in depths:
        EficienciaTrain = []
        EficienciaVal = []
        Macc = []
        Mf1 = []
        for train, test in skf.split(X, Y):
            Xtrain = X[train,:]
            Ytrain = Y[train]
            Xtest = X[test,:]
            Ytest = Y[test]

            #Haga el llamado a la función para crear y entrenar el modelo usando los datos de entrenamiento
            modelo = DecisionTreeClassifier(max_depth=depth)
            modelo = modelo.fit(Xtrain, Ytrain)
            #predecir muestras de entrenamiento
            Ytrain_pred = modelo.predict(Xtrain)
            #predecir muestras de pruebas
            Yest = modelo.predict(Xtest)
            #Evaluamos las predicciones del modelo con los datos de test
            EficienciaTrain.append(np.mean(Ytrain_pred.ravel() == Ytrain.ravel()))
            EficienciaVal.append(np.mean(Yest.ravel() == Ytest.ravel()))
            Macc.append(accuracy_score(Ytest, Yest))
            #Mf1.append(f1_score(Ytest, Yest))

        resultados.loc[idx,'profundidad del arbol'] = depth
        resultados.loc[idx,'eficiencia de entrenamiento'] = np.mean(EficienciaTrain)
        resultados.loc[idx,'desviacion estandar entrenamiento'] = np.std(EficienciaTrain)
        resultados.loc[idx,'eficiencia de prueba'] = np.mean(EficienciaVal)
        resultados.loc[idx,'desviacion estandar prueba'] = np.std(EficienciaVal)
        resultados.loc[idx,'accuracy'] = np.mean(Macc)
        #resultados.loc[idx,'f1_score'] = np.mean(Mf1)
        idx= idx +1

    return (resultados)
```

Es evidente que el algoritmo de árbol de decisión produce los mejores resultados al emplear una profundidad de árbol de 20.

	profundidad del arbol	eficiencia de entrenamiento	desviacion estandar entrenamiento	eficiencia de prueba	desviacion estandar prueba	accuracy
0	3.0	0.911856	0.001276	0.911860	0.011484	0.911860
1	10.0	0.959570	0.001111	0.945746	0.008759	0.945746
2	20.0	0.990661	0.000386	0.960139	0.018340	0.960139
3	100.0	0.991144	0.000342	0.959394	0.018999	0.959394

Seguidamente, se procede con la creación de la curva de aprendizaje, y se presenta el siguiente resultado:



Al examinar el gráfico, se nota que la línea de entrenamiento está notablemente por encima de la línea de prueba. Esto puede indicar una alta variabilidad en el modelo o un probable sobreajuste. Para afrontar esta situación, se recomienda contemplar la inclusión de más datos o la exclusión de variables menos relevantes.

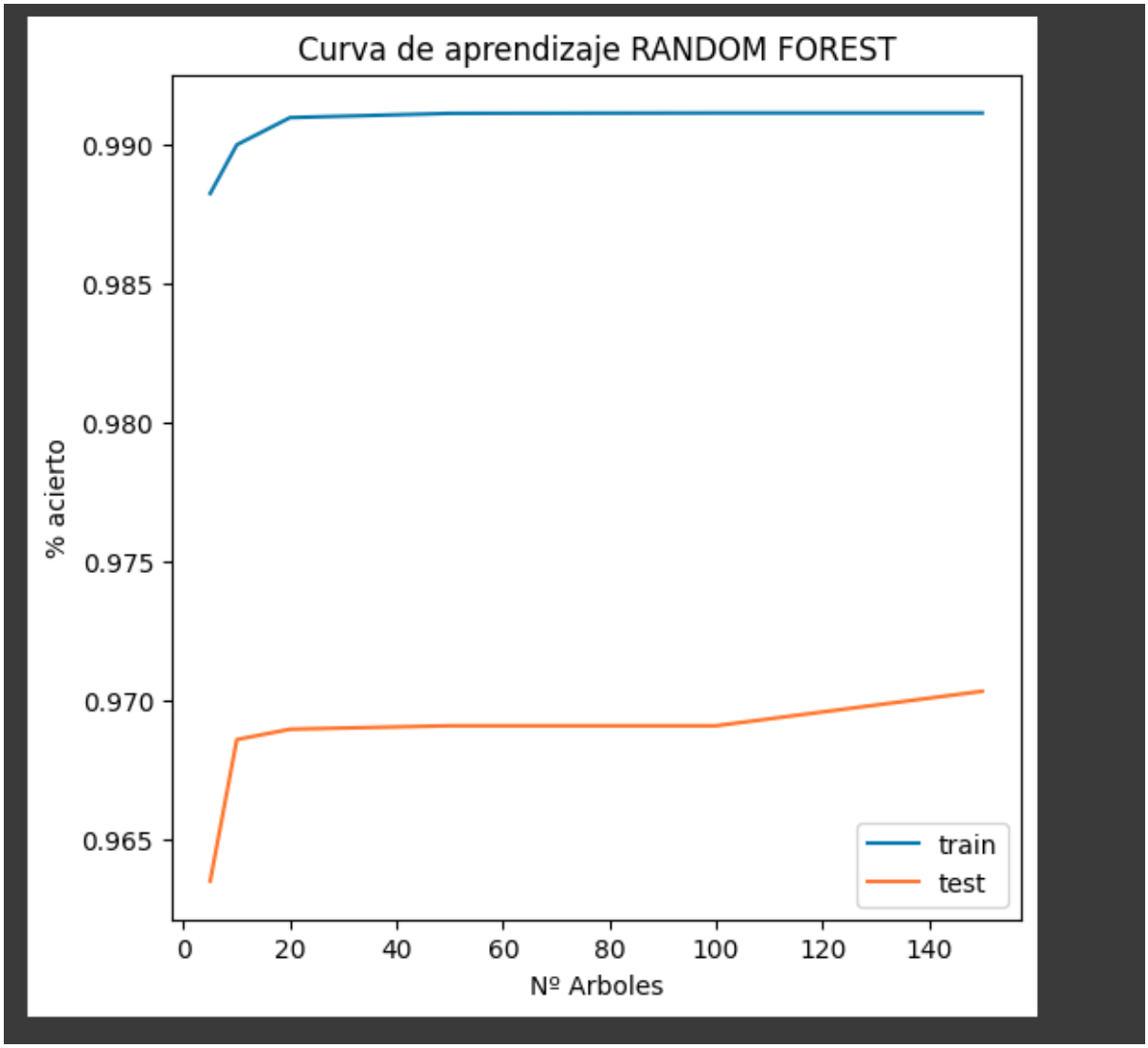
3.1.2 Random forest

Para el algoritmo de Random Forest, se probaron diversas cantidades de árboles (5, 10, 20, 100 y 150) como hiperparámetros. Se determinó que los resultados óptimos se logran con 150 árboles.

```
[ ]  def experimentar_rf(num_trees,numero_de_variables, X, Y):
         folds = 10
         skf = KFold(n_splits=folds)
         resultados = pd.DataFrame()
         idx = 0
         for trees in num_trees:
             for num_variables in numero_de_variables:
                 EficienciaTrain = []
                 EficienciaVal = []
                 Macc = []
                 Mpre = []
                 Mrec = []
                 Mf1 = []
                 for train, test in skf.split(X, Y):
                     Xtrain = X[train,:]
                     Ytrain = Y[train]
                     Xtest = X[test,:]
                     Ytest = Y[test]
                     modelo = RandomForestClassifier(n_estimators=trees, max_features=num_variables, criterion="gini")
                     modelo.fit(Xtrain,Ytrain)
                     Ytrain_pred = modelo.predict(Xtrain)
                     Yest = modelo.predict(Xtest)
                     EficienciaTrain.append(np.mean(Ytrain_pred.ravel() == Ytrain.ravel()))
                     EficienciaVal.append(np.mean(Yest.ravel() == Ytest.ravel()))
                     Macc.append(accuracy_score(Ytest, Yest))
                     Mf1.append(f1_score(Ytest, Yest))



                 resultados.loc[idx,'número de arboles'] = trees
                 resultados.loc[idx,'variables para la selección del mejor umbral'] = num_variables
                 resultados.loc[idx,'eficiencia de entrenamiento'] = np.mean(EficienciaTrain)
                 resultados.loc[idx,'desviacion estandar entrenamiento'] = np.std(EficienciaTrain)
                 resultados.loc[idx,'eficiencia de prueba'] = np.mean(EficienciaVal)
                 resultados.loc[idx,'Intervalo de confianza (prueba)'] = np.std(EficienciaVal)
                 resultados.loc[idx,'accuracy real'] = np.mean(Macc)
                 resultados.loc[idx,'f1_score'] = np.mean(Mf1)
                 idx= idx +1
             print(f"termina para {trees} arboles")

         return (resultados)
```

	número de arboles	variables para la selección del mejor umbral	eficiencia de entrenamiento	desviacion estandar entrenamiento	eficiencia de prueba	Intervalo de confianza (prueba)	accuracy real	f1_score
0	5.0	5.0	0.988247	0.000780	0.963495	0.014816	0.963495	0.967202
1	10.0	5.0	0.989999	0.000573	0.968585	0.010391	0.968585	0.971854
2	20.0	5.0	0.990979	0.000366	0.968958	0.012105	0.968958	0.972148
3	50.0	5.0	0.991130	0.000321	0.969082	0.010903	0.969082	0.972308
4	100.0	5.0	0.991144	0.000342	0.969082	0.010674	0.969082	0.972291
5	150.0	5.0	0.991144	0.000342	0.970324	0.010528	0.970324	0.973428

Después, se genera la curva de aprendizaje, obteniendo el siguiente resultado:



Al examinar el gráfico, se aprecia que la línea correspondiente al conjunto de entrenamiento está considerablemente por encima de la línea de prueba, sugiriendo la posibilidad de alta variabilidad o sobreajuste. Para abordar esta situación, se sugiere contemplar la inclusión de más datos o la eliminación de variables menos relevantes.

3.2 Hiper parámetros para dos algoritmos predictivos + no supervisado PCA

En el marco de ajuste de hiperparámetros para los algoritmos predictivos previamente mencionados, incluyendo el algoritmo no supervisado PCA, se realizó un proceso iterativo en cada fold del modelo con el fin de identificar los hiperparámetros óptimos. Paralelamente, se implementó una transformación PCA. Esta metodología permitió obtener la eficacia en el entrenamiento y en las pruebas para cada combinación evaluada.

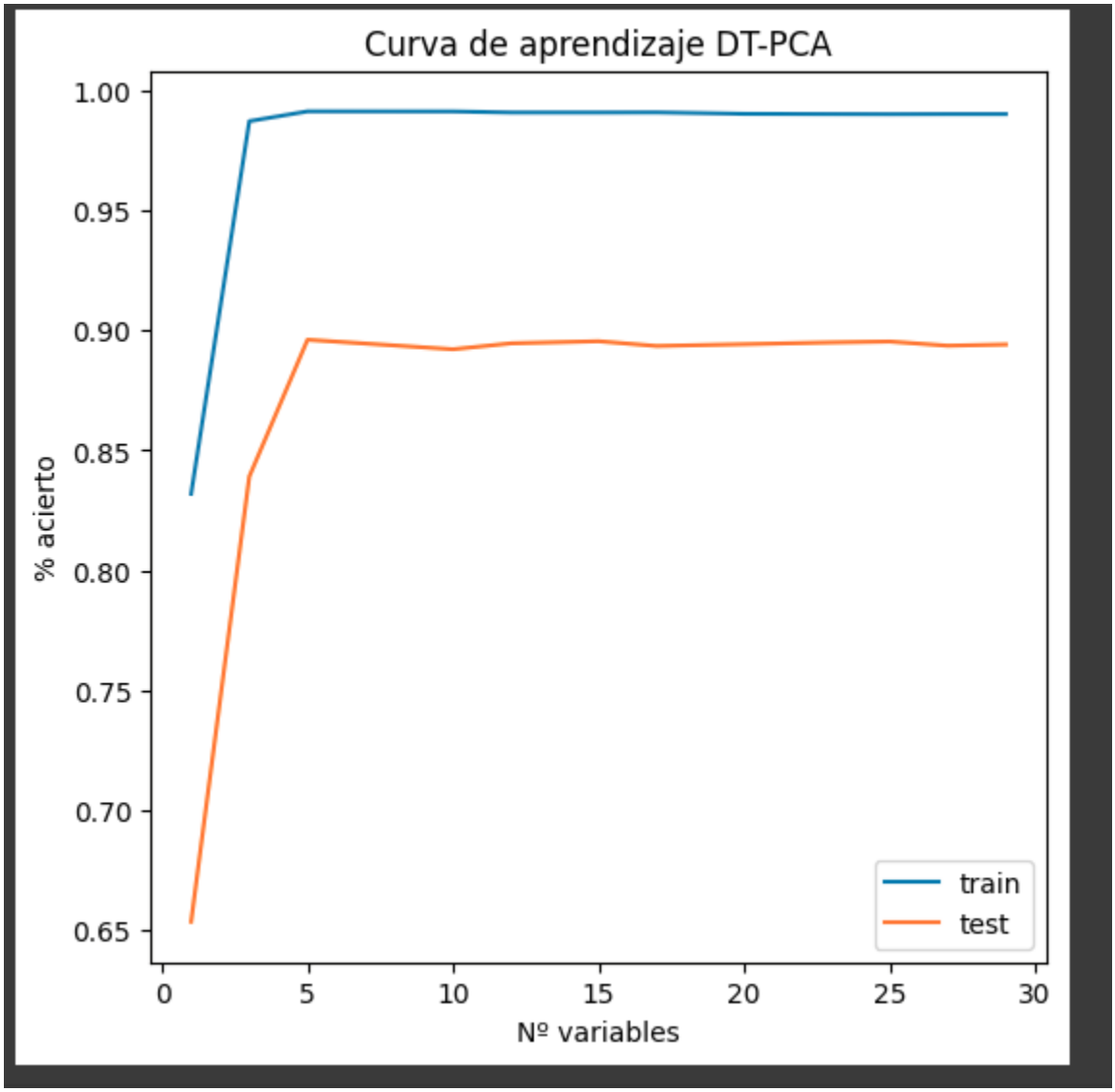
3.2.1 Decision tree con PCA

Para el algoritmo Decision Tree junto con PCA, se determinó que se alcanzan los resultados más óptimos utilizando 29 componentes.

```python
def experimentar_dt_PCA(num_componentes,depths,X, Y):
    folds = 4
    skf = KFold(n_splits=folds)
    resultados = pd.DataFrame()
    idx = 0
    for num_comp in num_componentes:
      for depth in depths:
          Mf1 = []
          EficienciaTrain = []
          EficienciaVal = []
          for train, test in skf.split(X, Y):
              Xtrain = X[train,:]
              Ytrain = Y[train]
              Xtest = X[test,:]
              Ytest = Y[test]
              pca = PCA(n_components=num_comp)
              pca.fit(Xtrain)
              Xtrain = pca.transform(Xtrain)
              Xtest = pca.transform(Xtest)
              modelo = DecisionTreeClassifier(max_depth=depth)
              modelo = modelo.fit(Xtrain, Ytrain)
              Ytrain_pred = modelo.predict(Xtrain)
              Yest = modelo.predict(Xtest)
              EficienciaTrain.append(np.mean(Ytrain_pred.ravel() == Ytrain.ravel()))
              EficienciaVal.append(np.mean(Yest.ravel() == Ytest.ravel()))
              Mf1.append(f1_score(Ytest, Yest))

          resultados.loc[idx,'PCA componentes'] = num_comp
          resultados.loc[idx,'eficiencia de entrenamiento'] = np.mean(EficienciaTrain)
          resultados.loc[idx,'desviacion estandar entrenamiento'] = np.std(EficienciaTrain)
          resultados.loc[idx,'eficiencia de prueba'] = np.mean(EficienciaVal)
          resultados.loc[idx,'desviacion estandar prueba'] = np.std(EficienciaVal)
          resultados.loc[idx,'f1_score'] = np.mean(Mf1)
          idx= idx +1

    return (resultados)
```

	PCA componentes	eficiencia de entrenamiento	desviacion estandar entrenamiento	eficiencia de prueba	desviacion estandar prueba	f1_score
0	1.0	0.831905	0.022016	0.653244	0.097545	0.711525
1	3.0	0.987461	0.003138	0.839096	0.085986	0.854928
2	5.0	0.991558	0.000702	0.896202	0.061655	0.908862
3	10.0	0.991517	0.000744	0.892229	0.066036	0.904742
4	12.0	0.991103	0.001202	0.894712	0.063872	0.906205
5	15.0	0.991103	0.001149	0.895582	0.065685	0.907267
6	17.0	0.991144	0.000603	0.893596	0.063774	0.905346
7	20.0	0.990523	0.002253	0.894342	0.060014	0.905201
8	23.0	0.990441	0.002392	0.895087	0.060197	0.905327
9	25.0	0.990399	0.002461	0.895459	0.058760	0.905687
10	27.0	0.990441	0.002392	0.893721	0.061186	0.904255
11	29.0	0.990441	0.002392	0.894217	0.062031	0.905163

Luego, se llevó a cabo el trazado de la curva de aprendizaje, obteniendo el resultado siguiente:



Al examinar el gráfico, se evidencia que la línea correspondiente al conjunto de entrenamiento está significativamente por encima de la línea de prueba. Asimismo, el porcentaje de precisión es similar al del algoritmo sin aplicar PCA, lo que sugiere que la implementación de PCA no impacta la precisión del modelo. También se observa una posible alta variabilidad o sobreajuste. Para resolver esta situación, se recomienda contemplar la inclusión de más datos o la eliminación de variables menos relevantes.

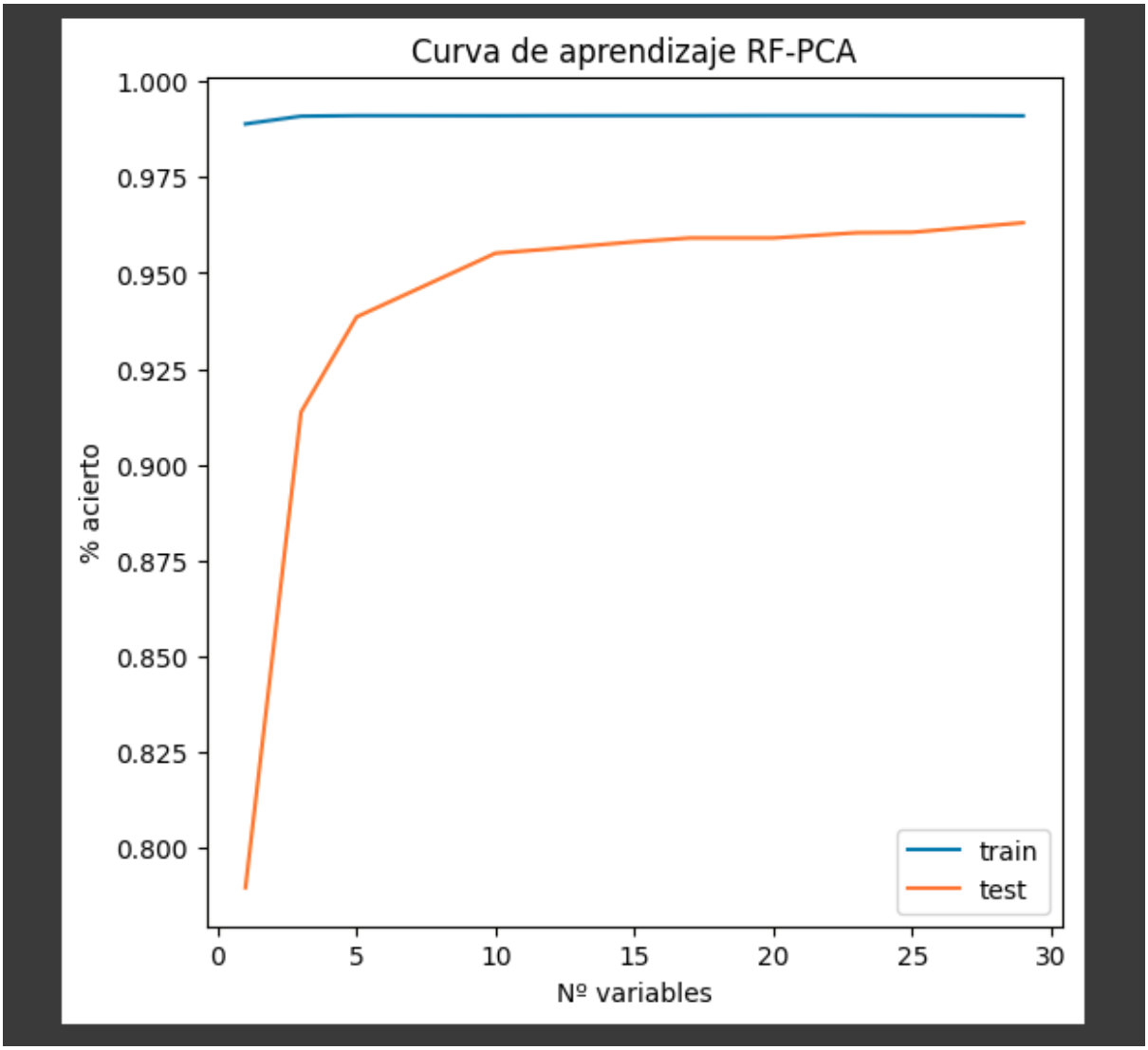
3.2.2 Random forest con PCA

Para el algoritmo Random Forest combinado con PCA, se determinó que se logran los resultados más óptimos utilizando 29 componentes.

```python
def experimentar_rf_PCA(num_componentes, X, Y):
    folds = 10
    skf = StratifiedKFold(n_splits=folds)
    resultados = pd.DataFrame()
    idx = 0
    for num_comp in num_componentes:

          EficienciaTrain = []
          EficienciaVal = []
          Mf1 = []
          for train, test in skf.split(X, Y):
              Xtrain = X[train,:]
              Ytrain = Y[train]
              Xtest = X[test,:]
              Ytest = Y[test]

              pca = PCA(n_components=num_comp)
              pca.fit(Xtrain)
              Xtrain = pca.transform(Xtrain)
              Xtest = pca.transform(Xtest)
              modelo = RandomForestClassifier(n_estimators=50, criterion="gini")
              modelo.fit(Xtrain, Ytrain)
              Ytrain_pred = modelo.predict(Xtrain)
              Yest = modelo.predict(Xtest)
              EficienciaTrain.append(np.mean(Ytrain_pred.ravel() == Ytrain.ravel()))
              EficienciaVal.append(np.mean(Yest.ravel() == Ytest.ravel()))
              Mf1.append(f1_score(Ytest, Yest))

          resultados.loc[idx,'PCA componentes'] = num_comp
          resultados.loc[idx,'eficiencia de entrenamiento'] = np.mean(EficienciaTrain)
          resultados.loc[idx,'desviacion estandar entrenamiento'] = np.std(EficienciaTrain)
          resultados.loc[idx,'eficiencia de prueba'] = np.mean(EficienciaVal)
          resultados.loc[idx,'desviacion estandar prueba'] = np.std(EficienciaVal)
          resultados.loc[idx,'f1_score'] = np.mean(Mf1)
          idx= idx +1

    return (resultados)
```

	PCA componentes	eficiencia de entrenamiento	desviacion estandar entrenamiento	eficiencia de prueba	desviacion estandar prueba	f1_score
0	1.0	0.988910	0.000773	0.789638	0.100304	0.814001
1	3.0	0.990937	0.000370	0.913696	0.039437	0.922612
2	5.0	0.991075	0.000355	0.938531	0.029262	0.944940
3	10.0	0.991048	0.000420	0.955173	0.019342	0.959681
4	12.0	0.991075	0.000344	0.956290	0.018900	0.960768
5	15.0	0.991089	0.000410	0.958153	0.017643	0.962419
6	17.0	0.991089	0.000381	0.959148	0.015151	0.963365
7	20.0	0.991130	0.000365	0.959148	0.014800	0.963477
8	23.0	0.991130	0.000365	0.960513	0.014509	0.964646
9	25.0	0.991089	0.000333	0.960638	0.014490	0.964776
10	27.0	0.991089	0.000366	0.961879	0.013991	0.965903
11	29.0	0.991034	0.000349	0.963121	0.013667	0.967076

Después, se llevó a cabo el trazado de la curva de aprendizaje, dando como resultado lo siguiente:



Al examinar el gráfico, se aprecia que la línea correspondiente al entrenamiento está por encima de la línea de prueba. No obstante, se percibe una reducción en la separación entre ambas líneas en comparación con el modelo sin utilizar PCA, lo que sugiere que el PCA ha disminuido la variabilidad. No obstante, aún persiste cierto grado de sobreajuste.

Recomendaciones para mejorar el desempeño

Para mejorar el desempeño obtenido, se sugieren las siguientes recomendaciones:

- Explorar otros tipos de algoritmos de aprendizaje automático, como redes neuronales o máquinas de soporte vectorial, con el fin de evaluar si generan resultados superiores.
- Realizar pruebas con diversas configuraciones de hiperparámetros en los algoritmos de Decision Tree y Random Forest para identificar combinaciones óptimas que mejoren el rendimiento.

- Llevar a cabo un análisis detallado para determinar las variables más relevantes para el modelo, por ejemplo, mediante el uso de técnicas de selección secuencial de características.

4. Retos y condiciones para desplegar en producción el modelo

En cuanto a la implementación del modelo en un entorno de producción, uno de los desafíos centrales reside en la extracción de datos desde bases de datos externas. Este proceso conlleva la búsqueda de información relevante para cada URL escaneada por la aplicación, como la antigüedad de la página, la verificación de presencia en listas negras o la validación de los permisos correspondientes.

Una opción viable para implementar el modelo en producción es utilizar un servicio basado en una API que pueda manejar solicitudes a través del método POST, proporcionando los parámetros de entrada necesarios. La respuesta ofrecería la predicción junto con la probabilidad de que la página sea de phishing, con el objetivo de alertar al usuario o cliente pertinente.

Por último, para supervisar el rendimiento del modelo a lo largo del tiempo, se recomienda mantener registros que contengan las predicciones obtenidas y los valores reales subsiguientes. Esto permitirá realizar un seguimiento del desempeño del modelo y evaluar cómo se comporta a medida que se enfrenta a datos más recientes.

5. Conclusión

En resumen, este análisis ha proporcionado detalles acerca de los hiperparámetros más efectivos, las variaciones al aplicar PCA y ha ofrecido recomendaciones para optimizar el rendimiento de los modelos predictivos empleados. Además, se han señalado los desafíos y consideraciones necesarias para su implementación en un entorno de producción.

Referencias

1. Kaggle - Phishing website dataset: link
2. scikit-learn - F1 Score: link
3. Minitab - ¿Qué son variables categóricas, discretas y continuas?: link
4. scikit-learn - Accuracy Score: link
5. scikit-learn - Decision Trees: link
6. scikit-learn - Principal Component Analysis (PCA): link
7. scikit-learn - Random Forest Classifier: link