Entrega final

Responsables:

DAVID LONDONO LÓPEZ

ISAAC JIMÉNEZ FERNADEZ

Profesor:

Raul Ramos Pollan

Universidad de Antioquia

Introducción a la inteligencia artificial

2023/1

1. Planteamiento del problema

El problema que se está planteando es un problema de clasificación, que tiene como objetivo detectar y clasificar páginas con phishing. El término anti-phishing se refiere a las medidas preventivas para bloquear los ataques de phishing. El phishing es un delito cibernético en el que los atacantes se hacen pasar por entidades confiables o conocidas y contactan a las personas a través de diferentes medios, como correo electrónico, mensajes de texto o teléfono, con el fin de obtener información confidencial.

Por lo general, en un ataque de phishing por correo electrónico, el mensaje engañoso sugerirá que hay un problema con una factura, que ha habido actividad sospechosa en una cuenta o que el usuario debe iniciar sesión para verificar una cuenta o contraseña. Además, los atacantes pueden solicitar a los usuarios que ingresen información de la tarjeta de crédito, detalles bancarios y otros datos personales confidenciales.

Una vez que los atacantes recopilan esta información, pueden utilizarla para acceder a cuentas, robar datos e identidades, así como descargar malware en la computadora del usuario. Por lo tanto, la implementación de medidas de seguridad efectivas para prevenir los ataques de phishing es esencial para proteger la privacidad y seguridad en línea de los usuarios.

1.2.Dataset

El dataset seleccionado es Phishing Dataset for Machine Learning (https://www.kaggle.com/datasets/shashwatwork/phishing-dataset-for-machine-learning) Este conjunto de datos contiene 48 atributos extraídos de 5000 páginas web de phishing y 5000 páginas web legítimas, que se descargaron de enero a mayo de 2015 y de mayo a junio de 2017. Se emplea una técnica mejorada de extracción de funciones aprovechando el marco de automatización del navegador (es decir, Selenium WebDriver), que es más preciso y robusto en comparación con el enfoque de análisis basado en expresiones regulares.

Algunos de los atributos más significativos son:

- NumDots, variable continua numérica
- UrlLength, variable continua numérica
- NumDash, variable continua numérica
- NoHttps, variable categórica 0 y 1
- IpAddress, variable categórica 0 y 1
- RandomString, variable categórica 0 y 1
- HostnameLength, variable continua numérica
- PopUpWindow, variable categórica 0 y 1

1.3. Métricas

Para la evaluación del sistema se emplearán dos métricas de evaluación principales: el accuracy y el f1 score, ya que ambos se enfocan en la precisión. Además de estas métricas técnicas, se tiene en cuenta la métrica de negocio, la fiabilidad de las predicciones para determinar si una página tiene phishing o no. Es crucial que estas predicciones sean confiables para que el navegador web pueda evitar que sus usuarios accedan a páginas maliciosas.

1.4. Desempeño

En un modelo de este tipo, se espera que la precisión de las predicciones sea alta, superando el 80%, además será importante evitar un gran número de falsos positivos.

En un ambiente productivo, el modelo sería utilizado como un filtro para prevenir que los usuarios accedan a páginas sospechosas y, de esta manera, garantizar la seguridad de los usuarios. Por lo tanto, es fundamental que el modelo pueda proporcionar predicciones confiables y precisas para lograr este objetivo.

2. Exploración descriptiva del dataset

La base de datos utilizada en este proyecto es el "Phishing Dataset for Machine Learning" y consta de 10,000 muestras y 48 variables. De estas variables, 47 se utilizan para describir las características de una URL de un sitio web, mientras que la otra variable es la salida "CLASS_LABEL", que indica si el sitio web es o no un sitio de phishing.

La información contenida en las 47 variables descriptivas se utiliza para identificar patrones y características comunes entre los sitios web de phishing. Al analizar estas variables, se pueden crear modelos de aprendizaje automático que puedan detectar de manera efectiva la presencia de sitios web de phishing y proteger a los usuarios contra posibles ataques.

El nombre de las columnas presentes en el dataset son las siguientes:

INFORMACION DE LAS COLUMNAS

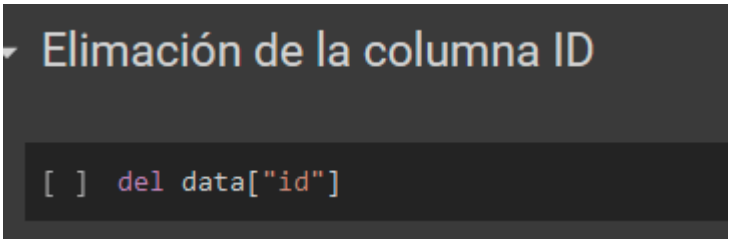
```
[ ] data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 49 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   NumDots                               10000 non-null  int64
 1   SubdomainLevel                        10000 non-null  int64
 2   PathLevel                             10000 non-null  int64
 3   UrlLength                             10000 non-null  int64
 4   NumDash                               10000 non-null  int64
 5   NumDashInHostname                     10000 non-null  int64
 6   AtSymbol                              10000 non-null  int64
 7   TildeSymbol                           10000 non-null  int64
 8   NumUnderscore                         10000 non-null  int64
 9   NumPercent                            10000 non-null  int64
 10  NumQueryComponents                    10000 non-null  int64
 11  NumAmpersand                          10000 non-null  int64
 12  NumHash                               10000 non-null  int64
 13  NumNumericChars                       10000 non-null  int64
 14  NoHttps                               10000 non-null  int64
 15  RandomString                          10000 non-null  int64
 16  IpAddress                             10000 non-null  int64
 17  DomainInSubdomains                    10000 non-null  int64
 18  DomainInPaths                         10000 non-null  int64
 19  HttpsInHostname                       10000 non-null  int64
 20  HostnameLength                        10000 non-null  int64
 21  PathLength                            10000 non-null  int64
 22  QueryLength                           10000 non-null  int64
 23  DoubleSlashInPath                     10000 non-null  int64
 24  NumSensitiveWords                     10000 non-null  int64
 25  EmbeddedBrandName                     10000 non-null  int64
 26  PctExtHyperlinks                      10000 non-null  float64
 27  PctExtResourceUrls                    10000 non-null  float64
 28  ExtFavicon                            10000 non-null  int64
 29  InsecureForms                         10000 non-null  int64
 30  RelativeFormAction                    10000 non-null  int64
 31  ExtFormAction                         10000 non-null  int64
 32  AbnormalFormAction                    10000 non-null  int64
 33  PctNullSelfRedirectHyperlinks         10000 non-null  float64
 34  FrequentDomainNameMismatch            10000 non-null  int64
 35  FakeLinkInStatusBar                   10000 non-null  int64
 36  RightClickDisabled                    10000 non-null  int64
 37  PopUpWindow                           10000 non-null  int64
 38  SubmitInfoToEmail                     10000 non-null  int64
 39  IframeOrFrame                         10000 non-null  int64
 40  MissingTitle                          10000 non-null  int64
 41  ImagesOnlyInForm                      10000 non-null  int64
 42  SubdomainLevelRT                      10000 non-null  int64
 43  UrlLengthRT                           10000 non-null  int64
 44  PctExtResourceUrlsRT                  10000 non-null  int64
 45  AbnormalExtFormActionR                10000 non-null  int64
 46  ExtMetaScriptLinkRT                   10000 non-null  int64
 47  PctExtNullSelfRedirectHyperlinksRT    10000 non-null  int64
 48  CLASS_LABEL                           10000 non-null  int64
dtypes: float64(3), int64(46)
memory usage: 3.7 MB
```

Además, se optó por eliminar la columna "ID" durante el análisis, ya que esta variable no es relevante para el propósito del estudio.

La columna "ID" suele ser un identificador único asignado a cada muestra en un conjunto de datos. En la mayoría de los casos, esta variable no tiene ningún impacto en la predicción o en el análisis de los datos. Es solo un número de referencia que

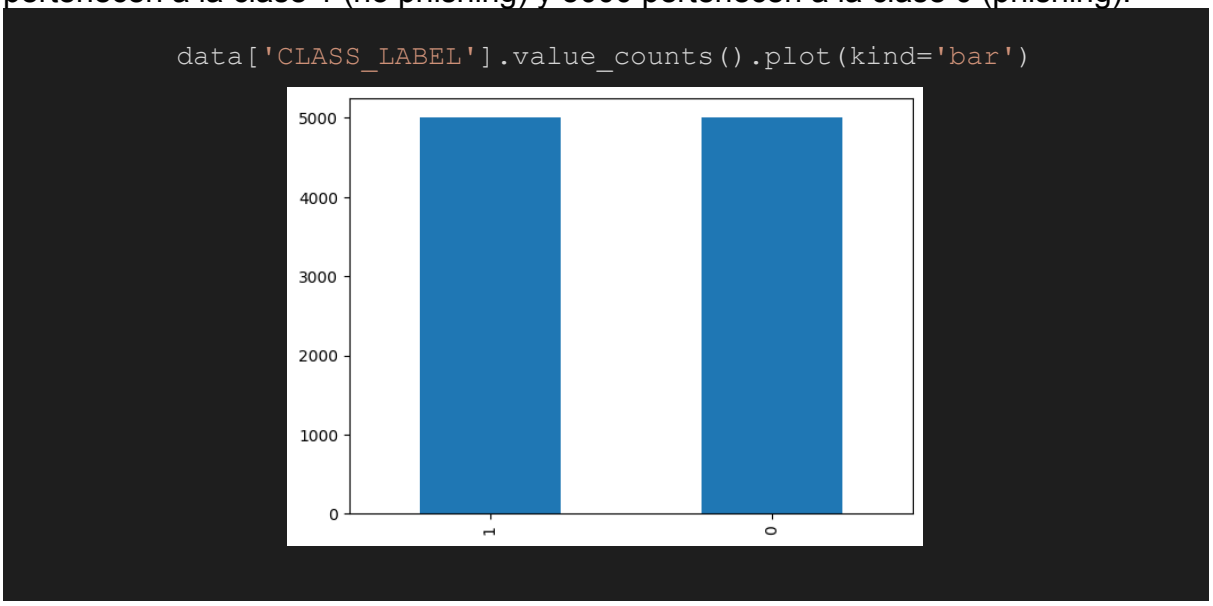
se utiliza para identificar la muestra en la base de datos.

Por lo tanto, eliminar la columna "ID" no afectará la calidad de los resultados y puede simplificar el análisis, ya que se reducirá la cantidad de variables en el conjunto de datos. Esto también puede ayudar a mejorar el rendimiento y la eficiencia de los modelos de aprendizaje automático, ya que tendrán menos variables que analizar. En general, la eliminación de variables irrelevantes puede ser una práctica común en el preprocesamiento de datos para mejorar la precisión y la eficiencia en el análisis de los datos.



2.1 Balanceo de los datos

Después de explorar el conjunto de datos, se observó que no hay desequilibrio en los datos. El dataset consta de un total de 10000 muestras, de las cuales 5000 pertenecen a la clase 1 (no phishing) y 5000 pertenecen a la clase 0 (phishing).



Este es un resultado muy importante en el análisis de datos, ya que el desequilibrio en la distribución de las muestras puede llevar a problemas en la construcción de modelos de aprendizaje automático. El desbalance en los datos puede hacer que el modelo esté sesgado hacia la clase dominante, lo que puede llevar a una baja precisión en la clasificación de la clase minoritaria.

Sin embargo, en este caso, el hecho de que no exista un desequilibrio en los datos facilita la construcción de modelos precisos para detectar sitios web de phishing. Al tener un conjunto de datos equilibrado, se puede entrenar el modelo de manera

justa y precisa, lo que se traduce en mejores resultados y una mayor seguridad en línea para los usuarios.

Se dividió él dataset en dos bloques destinados al entrenamiento de los datos y validación del modelo.

Observamos la mejor accuracy con profundidad del árbol 10.

Resultados del entrenamiento

```
[ ] 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	0.942922	0.002025	0.9367	0.017607	0.9367
1	10.0	0.987400	0.000788	0.9549	0.007543	0.9549
2	50.0	1.000000	0.000000	0.9480	0.014471	0.9480
3	100.0	1.000000	0.000000	0.9491	0.013671	0.9491

Se están realizando pruebas eliminando datos random del dataset

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

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

    (7000, 48) (7000,)

[ ] resultados_dt = experimentar_dt([3,10,20,100],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	0.943016	0.002393	0.929000	0.029166	0.929000
1	10.0	0.987587	0.000954	0.944000	0.027978	0.944000
2	20.0	0.999730	0.000355	0.938429	0.027334	0.938429
3	100.0	1.000000	0.000000	0.939143	0.029062	0.939143

3. Algoritmos de predicción

Se seleccionaron los algoritmos: Decision Tree y Random Forest para realizar el respectivo entrenamiento de los datos y consecuentemente generar modelos predictivos.

3.1 hiper parámetros para dos algoritmos predictivos

Con el objetivo de descubrir los hiperparámetros óptimos para los algoritmos predictivos seleccionados, es decir, Decision Tree y Random Forest, se llevó a cabo un proceso iterativo en cada pliegue del modelo. Esto permitió analizar y evaluar la eficiencia de entrenamiento y prueba para cada configuración. Además, se utilizó el método de validación cruzada K-Fold para asegurar una distribución equitativa de los datos de entrenamiento y prueba.

3.1.1 Decision tree

En el caso de este algoritmo, se exploró el hiperparámetro de la profundidad del árbol, utilizando diferentes valores (3, 10, 20 y 50).

A continuación, se muestra una imagen que ilustra el código utilizado para encontrar los mejores hiperparámetros en el algoritmo Decision Tree:

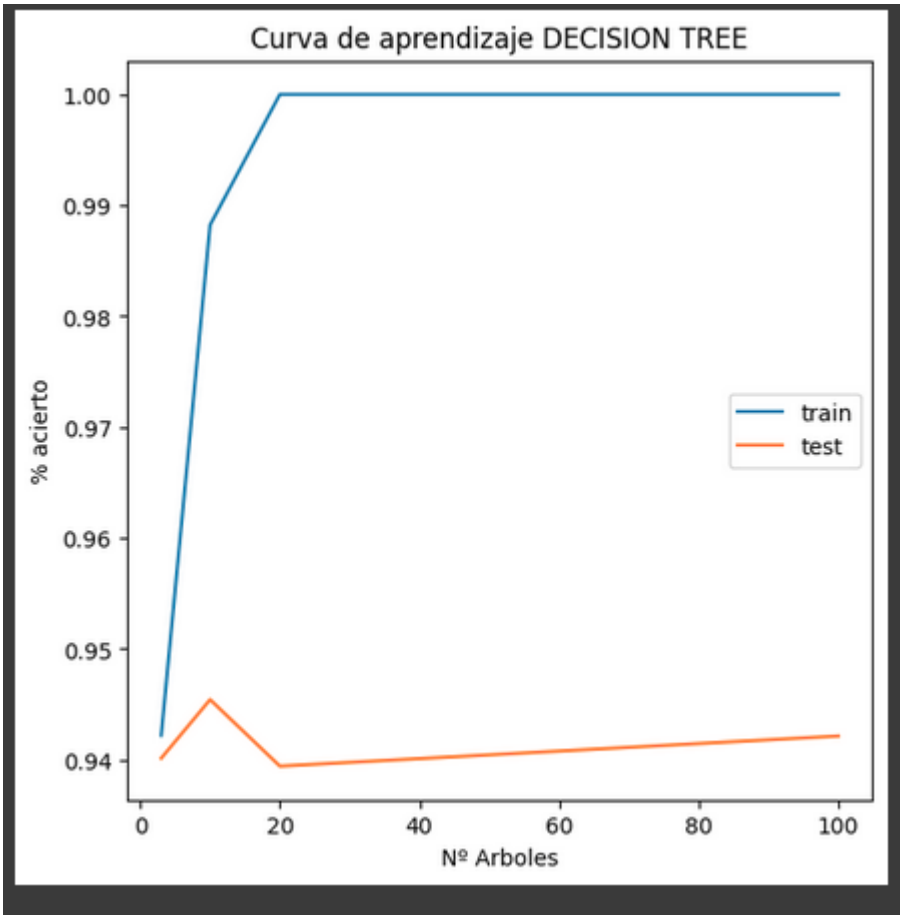
```python
def experimentar_dt(depths,X, Y):
    """funcion que realiza experimentos de arboles de decision
    depths: list[int] lista con la profundidad de arboles a experimentar
    normalize bool: indica si se aplica normalización a los datos
    X: matriz con las caractersiticas
    Y: matriz de 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)
```

Se puede notar que, en el caso del algoritmo de árbol de decisión, se logran los mejores resultados cuando se utiliza una profundidad de árbol de 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	0.942922	0.002025	0.9369	0.017541	0.9369
1	10.0	0.987411	0.000781	0.9560	0.008099	0.9560
2	50.0	1.000000	0.000000	0.9455	0.016336	0.9455
3	100.0	1.000000	0.000000	0.9475	0.014699	0.9475

A continuación, se realiza la curva de aprendizaje y se obtiene el siguiente resultado:



Al analizar el gráfico, se observa que la línea de entrenamiento se encuentra significativamente por encima de la línea de prueba. Esto podría indicar una alta variabilidad en el modelo o un posible sobreajuste. Para abordar esta situación, se sugiere considerar la incorporación de más datos o la eliminación de variables menos relevantes.

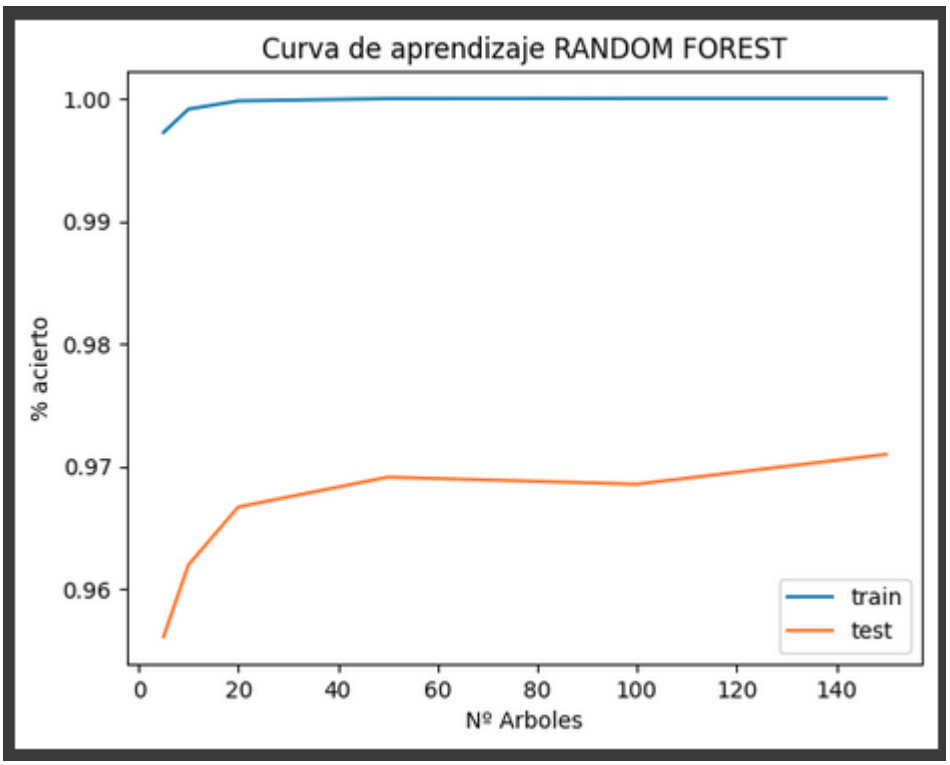
3.1.2 Random forest

En el caso del algoritmo de Random Forest, se utilizaron diferentes cantidades de árboles (5, 10, 20, 100 y 150) como hiperparámetro. Se encontró que los mejores resultados se obtienen 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	0.997222	0.000508	0.956143	0.010421	0.956143	0.554659
1	10.0	5.0	0.999127	0.000415	0.962000	0.019485	0.962000	0.565779
2	20.0	5.0	0.999794	0.000225	0.966714	0.010615	0.966714	0.564710
3	50.0	5.0	0.999984	0.000048	0.969143	0.011679	0.969143	0.570742
4	100.0	5.0	1.000000	0.000000	0.968571	0.012825	0.968571	0.566683
5	150.0	5.0	1.000000	0.000000	0.971000	0.010881	0.971000	0.569406

Posteriormente, se realiza la curva de aprendizaje, dando como resultado lo siguiente:



Al analizar el gráfico, se observa que la línea de entrenamiento está notablemente por encima de la línea de prueba, lo cual indica una posible alta variabilidad o sobreajuste. Para solucionar esto, se recomienda considerar la adición de más datos o la eliminación de variables menos significativas.

3.2 Hiper parámetros para dos algoritmos predictivos + no supervisado PCA

En el contexto de los hiperparámetros para los algoritmos predictivos mencionados anteriormente, así como el algoritmo no supervisado PCA, se llevó a cabo un proceso de iteración en cada pliegue del modelo para encontrar los mejores hiperparámetros. Además, se aplicó una transformación PCA. Se obtuvo la eficiencia de entrenamiento y prueba para cada combinación.

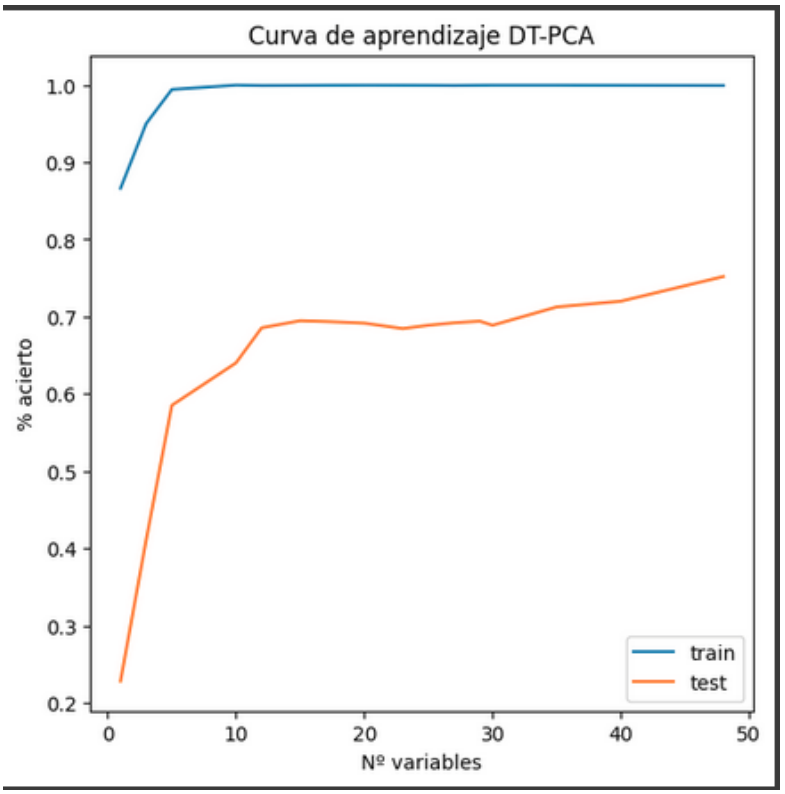
3.2.1 Decision tree con PCA

En el caso del algoritmo Decision Tree junto con PCA, se encontró que los mejores resultados se obtienen con 48 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	0.866238	0.032403	0.228000	0.055053	0.220477
1	3.0	0.950190	0.025353	0.411000	0.081037	0.324895
2	5.0	0.994286	0.002020	0.585143	0.130200	0.409797
3	10.0	0.999905	0.000095	0.640143	0.151987	0.445193
4	12.0	0.999381	0.001072	0.685571	0.140126	0.462358
5	15.0	0.999524	0.000547	0.694714	0.137838	0.466411
6	17.0	0.999714	0.000495	0.693714	0.119869	0.459670
7	20.0	0.999810	0.000330	0.691571	0.139977	0.463648
8	23.0	0.999762	0.000412	0.684571	0.136465	0.460777
9	25.0	0.999667	0.000577	0.688857	0.139734	0.460323
10	27.0	0.999429	0.000990	0.692000	0.145931	0.467918
11	29.0	0.999714	0.000316	0.694143	0.135705	0.467793
12	30.0	0.999810	0.000233	0.688857	0.138247	0.465982
13	35.0	0.999810	0.000190	0.712571	0.151793	0.481243
14	40.0	0.999667	0.000390	0.720000	0.136461	0.480357
15	48.0	0.999381	0.000493	0.752000	0.110573	0.487555

Posteriormente, se realizó la curva de aprendizaje, y se obtuvo el siguiente resultado:



Al analizar el gráfico, se observa que la línea de entrenamiento se encuentra considerablemente por encima de la línea de prueba. Además, el porcentaje de precisión es igual al del algoritmo sin PCA, lo cual indica que la aplicación de PCA no afecta la precisión del modelo. También se puede notar una alta variabilidad o posible sobreajuste. Para abordar esto, se sugiere considerar la adición de más datos o la eliminación de variables menos significativas.

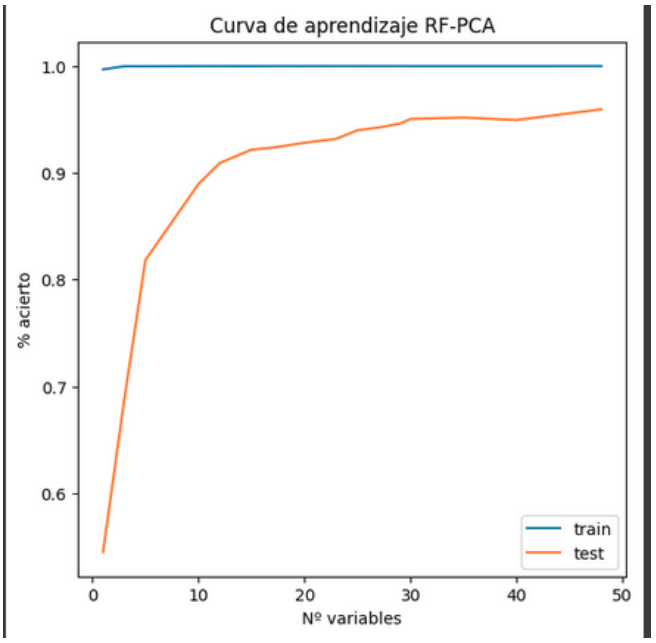
3.2.2 Random forest con PCA

En el caso del algoritmo Random Forest junto con PCA, se encontró que los mejores resultados se obtienen con 48 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	0.866238	0.032403	0.228000	0.055053	0.220477
1	3.0	0.950190	0.025353	0.411000	0.081037	0.324895
2	5.0	0.994286	0.002020	0.585143	0.130200	0.409797
3	10.0	0.999905	0.000095	0.640143	0.151987	0.445193
4	12.0	0.999381	0.001072	0.685571	0.140126	0.462358
5	15.0	0.999524	0.000547	0.694714	0.137838	0.466411
6	17.0	0.999714	0.000495	0.693714	0.119869	0.459670
7	20.0	0.999810	0.000330	0.691571	0.139977	0.463648
8	23.0	0.999762	0.000412	0.684571	0.136465	0.460777
9	25.0	0.999667	0.000577	0.688857	0.139734	0.460323
10	27.0	0.999429	0.000990	0.692000	0.145931	0.467918
11	29.0	0.999714	0.000316	0.694143	0.135705	0.467793
12	30.0	0.999810	0.000233	0.688857	0.138247	0.465982
13	35.0	0.999810	0.000190	0.712571	0.151793	0.481243
14	40.0	0.999667	0.000390	0.720000	0.136461	0.480357
15	48.0	0.999381	0.000493	0.752000	0.110573	0.487555

Posteriormente, se realizó la curva de aprendizaje, y se obtuvo el siguiente resultado:



Al analizar el gráfico, se observa que la línea de entrenamiento se encuentra por encima de la línea de prueba. Sin embargo, parece que se ha disminuido la brecha entre ambas líneas en comparación con el modelo sin PCA, lo que indica que el PCA ha reducido la variabilidad. Aunque aún existe cierto grado de sobreajuste.

Recomendaciones para mejorar el desempeño

Para mejorar el rendimiento obtenido, se pueden considerar las siguientes recomendaciones:

- Incluir otros tipos de algoritmos de aprendizaje automático, como redes neuronales o máquinas de soporte vectorial, para evaluar si proporcionan mejores resultados.
- Realizar pruebas con diferentes hiperparámetros en los algoritmos de Decision Tree y Random Forest, con el objetivo de encontrar combinaciones óptimas que mejoren el desempeño.
- Realizar un estudio exhaustivo para identificar las variables más significativas para el modelo, por ejemplo, utilizando técnicas de selección secuencial de características.

4. Retos y condiciones para desplegar en producción el modelo

En términos de despliegue del modelo en producción, uno de los desafíos principales sería extraer los datos de bases de datos externas. Esto implicaría

buscar información relevante, como la antigüedad de la página, verificar si está en listas negras o si tiene los permisos adecuados, para cada URL escaneada por la aplicación.

Para implementar el modelo en producción, se podría utilizar un servicio basado en una API que responda a las solicitudes a través de POST, con los parámetros de entrada necesarios. La respuesta proporcionaría la predicción y la probabilidad de que la página sea phishing, con el objetivo de alertar al cliente.

Por último, para monitorear el rendimiento del modelo, se recomienda almacenar registros con las predicciones obtenidas y los valores reales posteriores. Esto permitirá realizar un seguimiento del desempeño y evaluar cómo se comporta el modelo en el tiempo.

5. Conclusión

En conclusión, este análisis proporcionó información sobre los mejores hiperparámetros, diferencias al aplicar PCA y algunas recomendaciones para mejorar el rendimiento de los modelos predictivos utilizados, así como los desafíos y consideraciones para implementarlos en un entorno de producción.

Referencias

1. Kaggle - Phishing Dataset for Machine Learning: 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