RESULTADOS

En este informe haremos una comparación de clasificadores para un conjunto de datos sobre diagnósticos de cáncer de mama del" Repository of Machine Learning Databases" de UC Irvine.

Hemos utilizado el conjunto de datos "Breast Cancer Wisconsin (Diagnostic)" donado en 1995 y por supuesto sujeto al área de Salud y Medicina. Este conjunto de datos incluye características calculadas a partir de una imagen digitalizada de una aspiración con aguja fina de masa mamaria.

-Número de muestras: 569

-Número de características: 30

-Número de clases: 2, clasificadas como "B" o "M" (Benigno – 357, Maligno - 212)

Entre las características calculadas, para cada núcleo celular se incluyen medidas como radio (media de distancias desde el centro a puntos en el perímetro), textura (desviación estándar de los valores de escala de grises), perímetro, área, suavidad (variación local en longitudes de radio), compacidad (perímetro^2 / área - 1,0), concavidad (severidad de las porciones cóncavas del contorno), puntos cóncavos (número de porciones cóncavas del contorno), simetría, dimensión fractal ("aproximación de la línea costera" - 1).

En cuanto a las métricas de rendimiento utilizadas para comparar los distintos modelos de clasificación se encuentran exactitud, precisión, recall y f1-score.

MÉTODOS Y MODELOS UTILIZADOS

Se han implementado y evaluado a los siguientes clasificadores utilizando validación cruzada 10-fold para asegurar robustez en nuestros resultados:

-Clasificador Naive Bayes

Clasificador Nearest Neighbors

-Clasificador árbol de decisión

-Clasificador Support Vector Machine

CLASIFICADOR NAIVE BAYES

```
estimador_gnb = GaussianNB()
y_pred_train = estimador_gnb.fit(X_train, y_train).predict(X_train)
y_pred_test = estimador_gnb.predict(X_test)
```

```
accuracy_train = accuracy_score(y_train, y_pred_train)
accuracy_test = accuracy_score(y_test, y_pred_test)
precision_train = precision_score(y_train, y_pred_train)
precision_test = precision_score(y_test, y_pred_test)
recall_train = recall_score(y_train, y_pred_train)
recall_test = recall_score(y_test, y_pred_test)
f1_train = f1_score(y_train, y_pred_train)
f1_test = f1_score(y_test, y_pred_test)
metrics_dict['GaussianNB'] = {'Train Accuracy': accuracy_train,
'Test Accuracy': accuracy_test, 'Train Precision': precision_train,
'Test Precision': precision_test, 'Train Recall': recall_train,
'Test Recall': recall_test, 'Train F1-Score': f1_train, 'Test F1-
Score': f1_test}
print("\nClasificador Naive Bayes")
print("Exactitud(train): ", accuracy_train)
print("Exactitud(test): ", accuracy_test)
print("Precision(train): ", precision_train)
print("Precision(test): ", precision_test)
print("Recall(train): ", recall_train)
print("Recall(test): ", recall_test)
print("F1-Score(train): ", f1_train)
print("F1-Score(test): ", f1_test)
```

En cuanto a los parámetros escogidos para este clasificador, el modelo GaussianNB no necesita ajuste.

Como resultados obtenemos:

```
Clasificador Naive Bayes
Exactitud(train):  0.9507042253521126
Exactitud(test):  0.9368421052631579
Precision(train):  0.9801980198019802
Precision(test):  0.9191919191919192
Recall(train):  0.8918918918918919
Recall(test):  0.900990099009901
F1-Score(train):  0.9339622641509434
F1-Score(test):  0.91
```

CLASIFICADOR NEAREST NEIGHBORS

```
estimador_neigh = KNeighborsClassifier(n_neighbors=5)
estimador_neigh.fit(X_train,y_train)
predict_proba_df = pd.DataFrame(X_test,
columns=caracteristicas_columnas)

# Definimos lista de números de vecinos a probar
neighbors_list = range(1, 31)
cv_NN = []
```

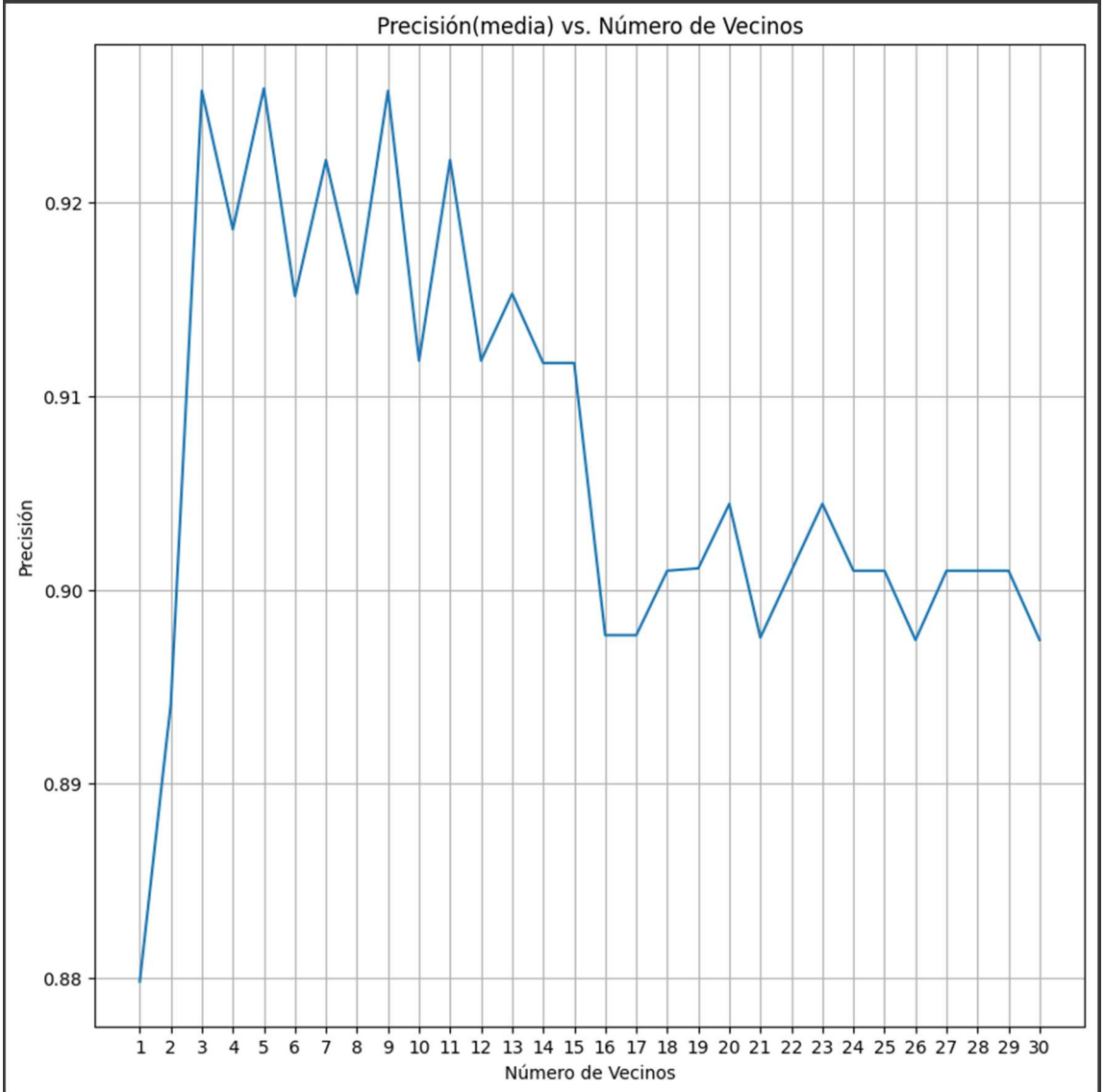
```
stratified_kfold = StratifiedKFold(n_splits=10, shuffle=True,
random_state=30)

# Realizar la validación cruzada estratificada 10-fold para cada
número de vecinos
for k in neighbors_list:
    kn = KNeighborsClassifier(n_neighbors=k)
    res = cross_val_score(kn, X_train, y_train,
cv=stratified_kfold, scoring='accuracy')
    cv_NN.append(res.mean())

# Encontrar el mejor número de vecinos a partir de mayor valor de
cv
bestK = neighbors_list[np.argmax(cv_NN)]
print("\nClasificador Nearest Neighbors")
print("El mejor número de vecinos es:", bestK)

# Creamos gráfica
plt.figure(figsize=(10, 10))
plt.plot(neighbors_list, cv_NN)
plt.title('Precisión(media) vs. Número de Vecinos')
plt.xlabel('Número de Vecinos')
plt.ylabel('Precisión')
plt.xticks(neighbors_list)
plt.grid(True)
plt.show()

# Calcular las métricas adicionales
y_pred_train = estimador_neigh.predict(X_train)
y_pred_test = estimador_neigh.predict(X_test)
accuracy_train = accuracy_score(y_train, y_pred_train)
accuracy_test = accuracy_score(y_test, y_pred_test)
precision_train = precision_score(y_train, y_pred_train)
precision_test = precision_score(y_test, y_pred_test)
recall_train = recall_score(y_train, y_pred_train)
recall_test = recall_score(y_test, y_pred_test)
f1_train = f1_score(y_train, y_pred_train)
f1_test = f1_score(y_test, y_pred_test)
print("Exactitud(train):", accuracy_train)
print("Exactitud(test):", accuracy_test)
print("Precision(train):", precision_train)
print("Precision(test):", precision_test)
print("Recall(train):", recall_train)
print("Recall(test):", recall_test)
print("F1-Score(train):", f1_train)
print("F1-Score(test):", f1_test)

# Guardar las métricas en el diccionario
```

```
metrics_dict['KNeighborsClassifier'] = {'Train Accuracy':
accuracy_train, 'Test Accuracy': accuracy_test, 'Train Precision':
precision_train, 'Test Precision': precision_test, 'Train Recall':
recall_train, 'Test Recall': recall_test, 'Train F1-Score':
f1_train, 'Test F1-Score': f1_test}
```

En cuanto a los parámetros escogidos para este clasificador, hemos asignado el número de vecinos a 5, ya que realizamos una búsqueda del mejor número de vecinos y se observó que el modelo alcanzaba su mejor rendimiento con este valor.

Como resultados obtenemos:

```
El mejor número de vecinos es: 5
```




```
Exactitud(train): 0.9436619718309859
Exactitud(test): 0.9368421052631579
Precision(train): 0.9357798165137615
Precision(test): 0.9108910891089109
Recall(train): 0.918918918918919
Recall(test): 0.9108910891089109
```

```
F1-Score(train): 0.9272727272727272
F1-Score(test): 0.9108910891089109
```

CLASIFICADOR DE ÁRBOL DE DECISIÓN

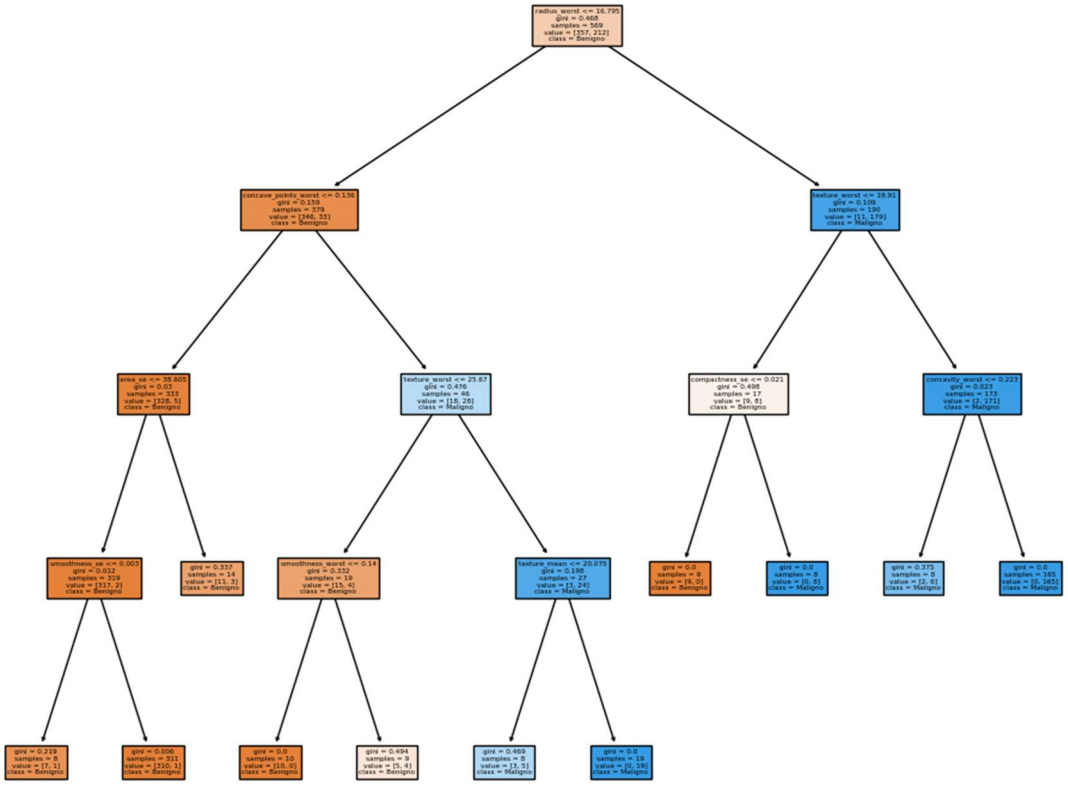
```
estimador_arbol = DecisionTreeClassifier(max_depth=2,
min_samples_leaf=1, min_samples_split=2, random_state=30)

# Entrenar el clasificador
estimador_arbol.fit(X_train, y_train)

# Predecir las etiquetas de los datos de entrenamiento y prueba
y_pred_train = estimador_arbol.predict(X_train)
y_pred_test = estimador_arbol.predict(X_test)

# Calcular la precisión para los datos de entrenamiento y prueba
accuracy_train = accuracy_score(y_train, y_pred_train)
accuracy_test = accuracy_score(y_test, y_pred_test)

# Calcular las métricas adicionales para los datos de entrenamiento
y prueba
precision_train = precision_score(y_train, y_pred_train)
precision_test = precision_score(y_test, y_pred_test)
recall_train = recall_score(y_train, y_pred_train)
recall_test = recall_score(y_test, y_pred_test)
f1_train = f1_score(y_train, y_pred_train)
f1_test = f1_score(y_test, y_pred_test)
print("\nClasificador de árbol de decisión")
print("Exactitud(train): ", accuracy_train)
print("Exactitud(test): ", accuracy_test)
print("Precision(train): ", precision_train)
print("Precision(test): ", precision_test)
print("Recall(train): ", recall_train)
print("Recall(test): ", recall_test)
print("F1-Score(train): ", f1_train)
print("F1-Score(test): ", f1_test, "\n")

# max_depth es la profundidad máxima del árbol
# min_samples_split es el número mínimo de muestras necesarias para
dividir un nodo interno
# min_samples_leaf es el número mínimo de muestras necesarias para
estar en un nodo hoja(dejo provisional a 4 sera 12)
param_grid = {'max_depth': range(1, 10),
              'min_samples_split': range(2, 10),
              'min_samples_leaf': range(1, 10)}

# Creo el objeto GridSearchCV
```

```
grid_searchcv = GridSearchCV(estimador_arbol, param_grid, cv=10)
grid_searchcv.fit(X, y)

# Imprimir los mejores hiperparámetros
print("Parámetros ideales para árbol: ",
grid_searchcv.best_params_)

# Obtener el mejor modelo de GridSearchCV y hacer gráfica
best_decision_tree = grid_searchcv.best_estimator_
plt.figure(figsize=(12, 10))
plot_tree(best_decision_tree,
feature_names=caracteristicas_columnas, class_names=['Benigno',
'Maligno'], filled=True)
plt.show()

# Guardar las métricas en el diccionario
metrics_dict['DecisionTreeClassifier'] = {'Train Accuracy':
accuracy_train, 'Test Accuracy': accuracy_test, 'Train Precision':
precision_train, 'Test Precision': precision_test, 'Train Recall':
recall_train, 'Test Recall': recall_test, 'Train F1-Score':
f1_train, 'Test F1-Score': f1_test}
```

En cuanto a los parámetros escogidos para este clasificador, hemos asignado la profundidad máxima del árbol a 2, el número mínimo de muestras por hojas a 1 y el número mínimo de muestras requeridas para dividir un nodo interno a 2.

Como resultados obtenemos:

```
Clasificador de árbol de decisión
Exactitud(train):  0.9612676056338029
Exactitud(test):  0.9263157894736842
Precision(train):  0.9716981132075472
Precision(test):  0.9444444444444444
Recall(train):  0.9279279279279279
Recall(test):  0.8415841584158416
F1-Score(train):  0.9493087557603687
F1-Score(test):  0.8900523560209423

Parámetros ideales para árbol:  {'max_depth': 4, 'min_samples_leaf':
8, 'min_samples_split': 2}
```



CLASIFICADOR SUPPORT VECTOR MACHINE

```
# Clasificador Support Vector Machine
estimador_svc = SVC()

# Entrenar el clasificador
estimador_svc.fit(X_train, y_train)

# Predecir las etiquetas de los datos de entrenamiento y prueba
y_pred_train = estimador_svc.predict(X_train)
y_pred_test = estimador_svc.predict(X_test)

# Calcular la precisión para los datos de entrenamiento y prueba
accuracy_train = accuracy_score(y_train, y_pred_train)
accuracy_test = accuracy_score(y_test, y_pred_test)

# Calcular las métricas adicionales para los datos de entrenamiento
y prueba
precision_train = precision_score(y_train, y_pred_train)
precision_test = precision_score(y_test, y_pred_test)
```

```
recall_train = recall_score(y_train, y_pred_train)
recall_test = recall_score(y_test, y_pred_test)
f1_train = f1_score(y_train, y_pred_train)
f1_test = f1_score(y_test, y_pred_test)
print("\nClasificador Support Vector Machine")
print("Exactitud(train): ", accuracy_train)
print("Exactitud(test): ", accuracy_test)
print("Precision(train): ", precision_train)
print("Precision(test): ", precision_test)
print("Recall(train): ", recall_train)
print("Recall(test): ", recall_test)
print("F1-Score(train): ", f1_train)
print("F1-Score(test): ", f1_test)

# Definir los parámetros para la búsqueda en malla
param_grid = {'C': [0.1, 1],
              'gamma': [1, 0.1],
              'kernel': ['rbf', 'linear', 'sigmoid']}

# Crear el objeto GridSearchCV con validación cruzada estratificada
grid_search = GridSearchCV(SVC(), param_grid,
cv=StratifiedKFold(n_splits=10, shuffle=True, random_state=30))

# Ajustar el modelo a los datos de entrenamiento
grid_search.fit(X_train, y_train)

# Imprimir los mejores hiperparámetros encontrados
print("Mejores parámetros para SVM: ", grid_search.best_params_)

# Guardar las métricas en el diccionario
metrics_dict['SVC'] = {'Train Accuracy': accuracy_train, 'Test
Accuracy': accuracy_test, 'Train Precision': precision_train, 'Test
Precision': precision_test, 'Train Recall': recall_train, 'Test
Recall': recall_test, 'Train F1-Score': f1_train, 'Test F1-Score':
f1_test}
```

En cuanto a los parámetros escogidos para este clasificador, hemos utilizado el modelo por defecto sin especificar ningún hiperparámetro en particular, después de ello, obtenemos que los mejores son: 'C':0.1, 'gamma':1,'kernel':linear.

Los resultados obtenidos han sido:

```
Clasificador Support Vector Machine
exactitud(train):  0.897887323943662
Exactitud(test):  0.9157894736842105
Precision(train):  0.9767441860465116
Precision(test):  0.9873417721518988
```

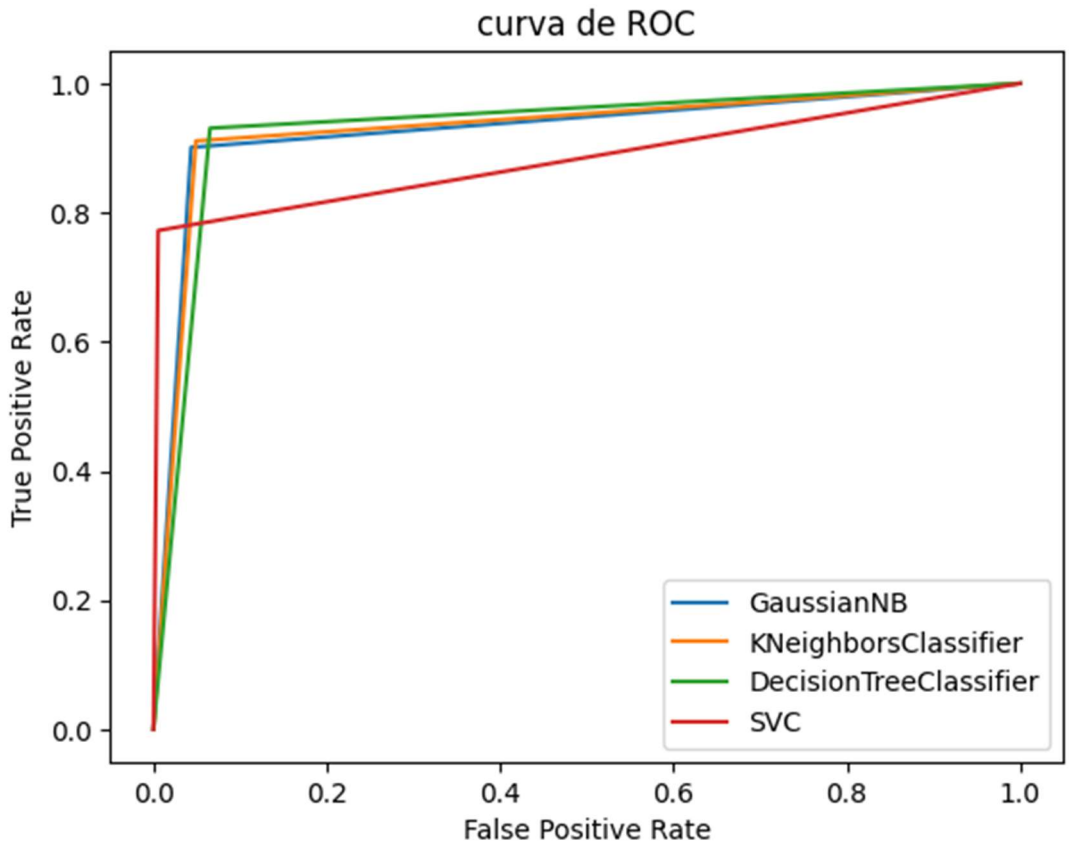
```
Recall(train):  0.7567567567567568
Recall(test):  0.7722772277227723
F1-Score(train):  0.8527918781725888
F1-Score(test):  0.8666666666666666
Mejores parámetros para SVM:  {'C': 0.1, 'gamma': 1, 'kernel':
'linear'}
```

CURVAS DE ROC

Ahora se presentan las curvas ROC para los clasificadores evaluados, las curvas ROC permiten visualizar el rendimiento de cada clasificador en términos de su capacidad para distinguir entre casos malignos y benignos

```
# Definir nombres y clasificadores
names = ['Naive Bayes', 'K-Nearest Neighbors', 'Decision Tree',
'Support Vector Machine']
classifiers = [GaussianNB(),
               KNeighborsClassifier(n_neighbors=5),
               DecisionTreeClassifier(max_depth=6,
min_samples_leaf=10, min_samples_split=3, random_state=30),
               SVC(probability=True)]




plt.figure()

for name, classifier in zip(names, classifiers):
    # Entrenar el clasificador
    classifier.fit(X_train, y_train)

    # Predecir las etiquetas de los datos de prueba
    y_pred = classifier.predict(X_test)

    # Calcular los valores de fpr, tpr y umbral para la curva ROC
    fpr, tpr, _ = roc_curve(y_test, y_pred)

    # Calcular el área bajo la curva ROC (AUC)
    roc_auc = roc_auc_score(y_test, y_pred)

    # Graficar la curva ROC para cada clasificador
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

# Configurar y mostrar la gráfica
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('Curva ROC')
plt.legend()
plt.show()
```



EVALUACIÓN FINAL

En este caso hemos considerado diversas medidas de rendimiento y aunque todas son relevantes, en el caso del cáncer, hay algunas que son especialmente importantes como el “recall”, ya que es fundamental detectar correctamente los casos malignos aunque esto signifique que algunos casos benignos se clasifiquen de manera errónea y también es importante considerar de más a “precisión” para asegurarnos de que los casos positivos realmente sean malignos. Por lo que, podríamos calcular una combinación de ambas a través de “F1-SCORE”.

MODELO	EXACTITUD	F1-SCORE
Naive Bayes	0.9368	0.91

Nearest Neighbors	0.9368	0.9108
Árbol De Decisión	0.9333	0.9082
Support Vector Machine	0.9157	0.8666

Basándonos en estas puntuaciones, vemos que el modelo “KNeighborsClassifier” tiene la puntuación más alta en los datos, lo que indica un buen equilibrio entre precisión y recall. Por lo tanto, el “KNeighborsClassifier” es efectivo para este conjunto de datos en particular.

REFERENCIAS

Enlace a Colab.

Enlace extra a mi primer proyecto personal sobre IA.

Repositorio.