PROYECTO DE MODELOS DE CLASIFICACIÓN

Juan Carlos Martinez F.

Universidad del Cauca

Introduccion a las Ciencia de Datos 2024-II

En primer lugar se muestra lo presentado el dia de la sustentación, sin embargo se hicieron las correcciones y estan adjuntas en una segunda parte de esta presentación.

Conjunto de Datos

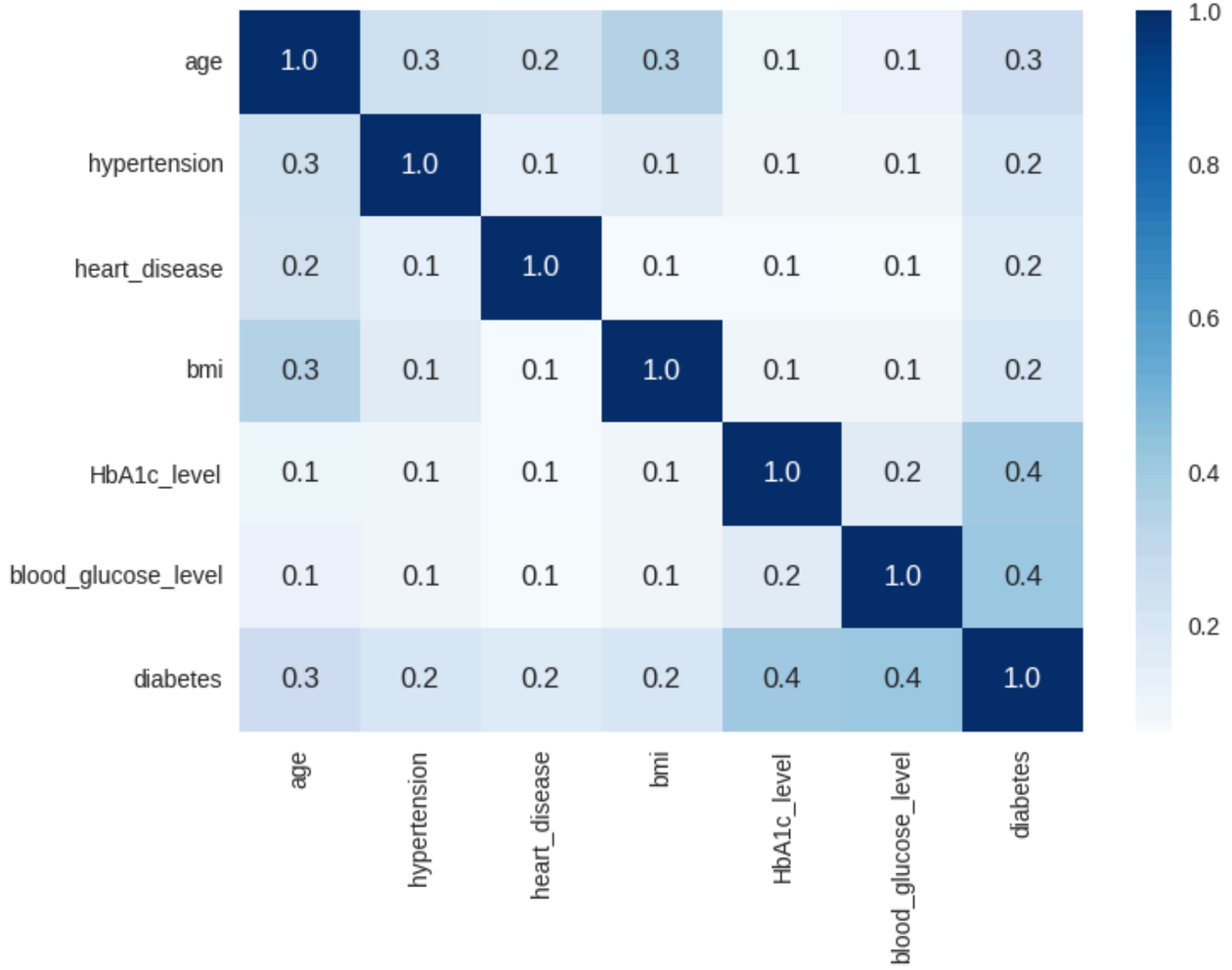
La base de datos tiene origen en Kaggle, el conjunto de datos cuenta con 100mil datos y 9 descriptores, los datos contienen información como la edad, el sexo, el índice de masa corporal (IMC), la hipertensión, las enfermedades cardíacas, el historial de tabaquismo, el nivel de HbA1c y el nivel de glucosa en sangre

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0
...	...	...	...	...	...	...	...	...	...
99995	Female	80.0	0	0	No Info	27.32	6.2	90	0
99996	Female	2.0	0	0	No Info	17.37	6.5	100	0
99997	Male	66.0	0	0	former	27.83	5.7	155	0
99998	Female	24.0	0	0	never	35.42	4.0	100	0
99999	Female	57.0	0	0	current	22.43	6.6	90	0

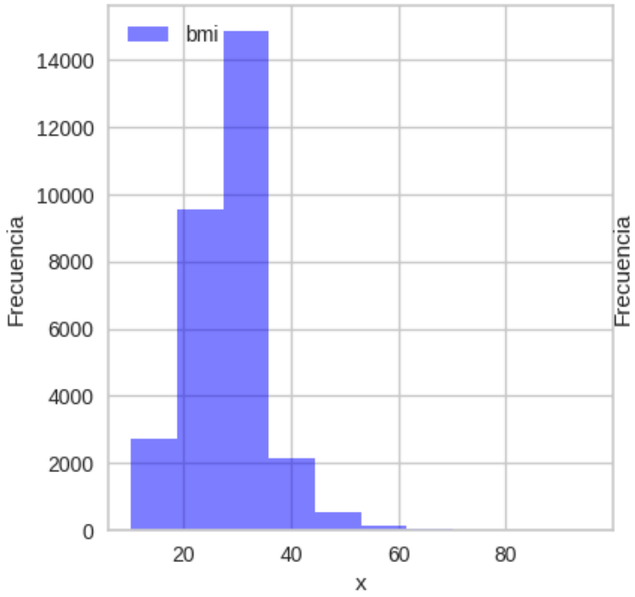
100000 rows × 9 columns

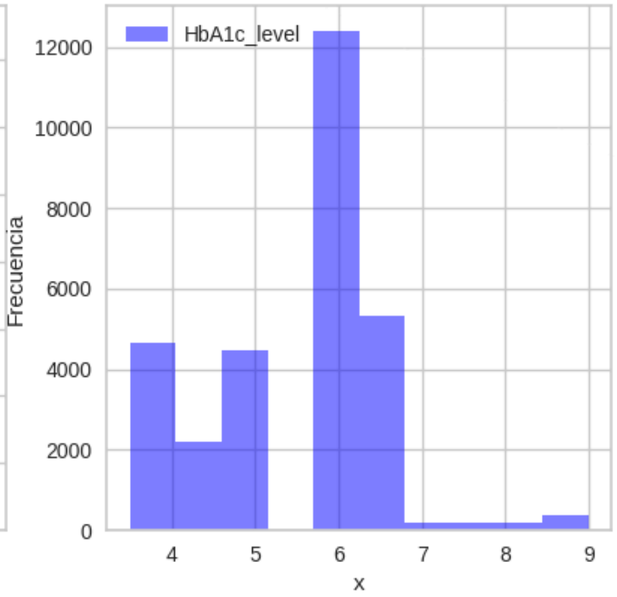
Analsis Exploratorio de Datos (EDA)

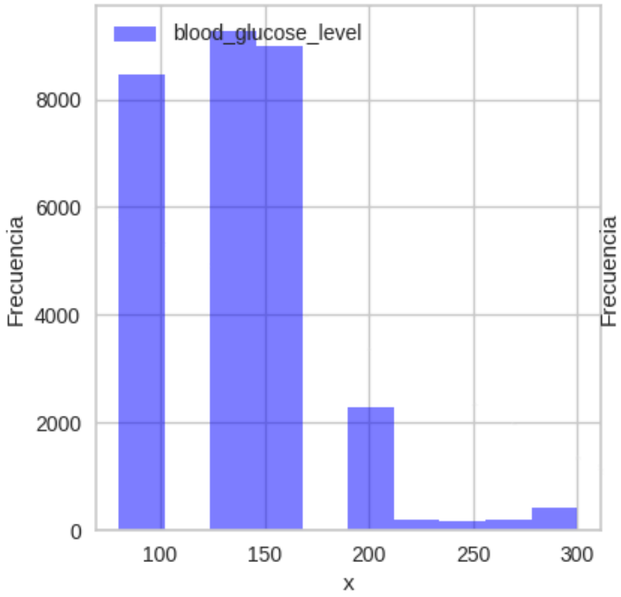
```
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   gender               100000 non-null  object 
 1   age                  100000 non-null  float64
 2   hypertension         100000 non-null  int64  
 3   heart_disease        100000 non-null  int64  
 4   smoking_history      100000 non-null  object 
 5   bmi                  100000 non-null  float64
 6   HbA1c_level          100000 non-null  float64
 7   blood_glucose_level  100000 non-null  int64  
 8   diabetes             100000 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 6.9+ MB
```

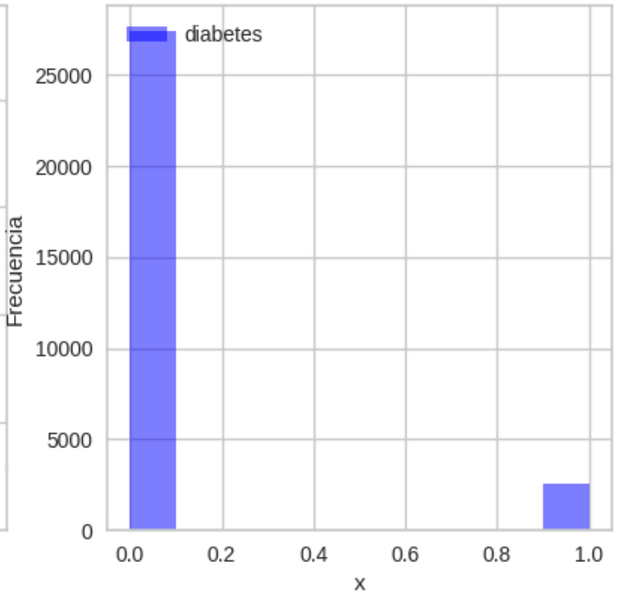


Analsis Exploratorio de Datos (EDA)

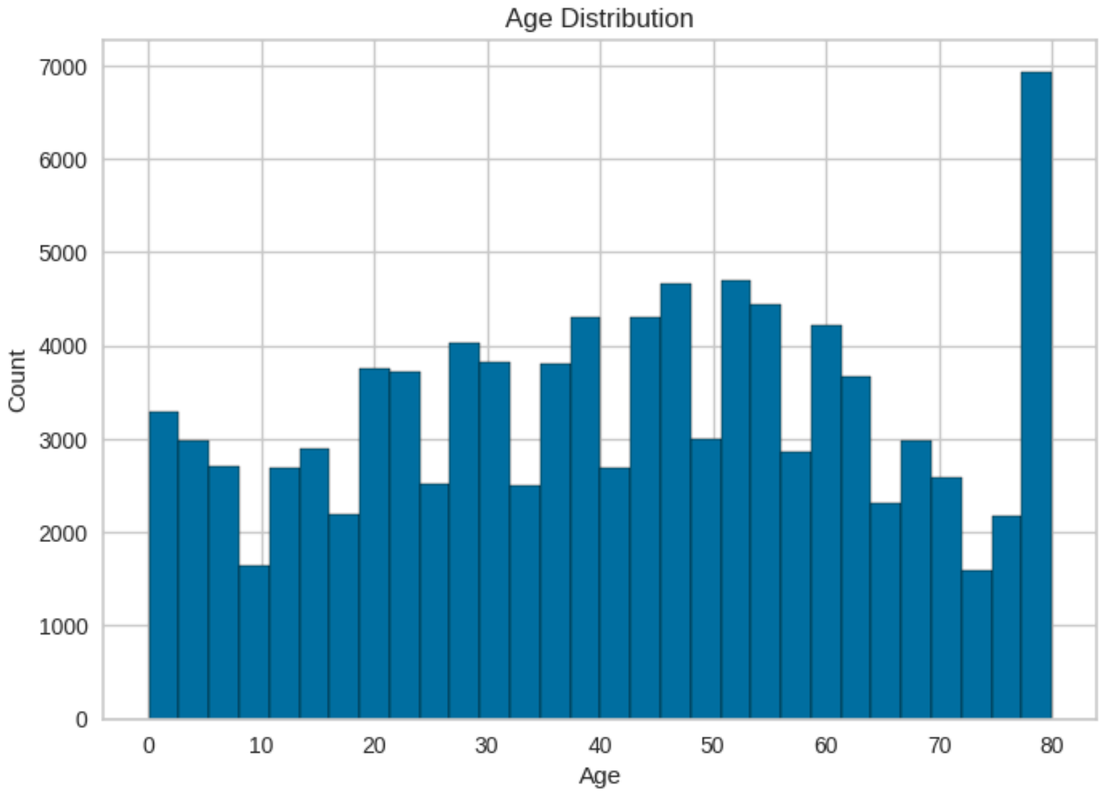


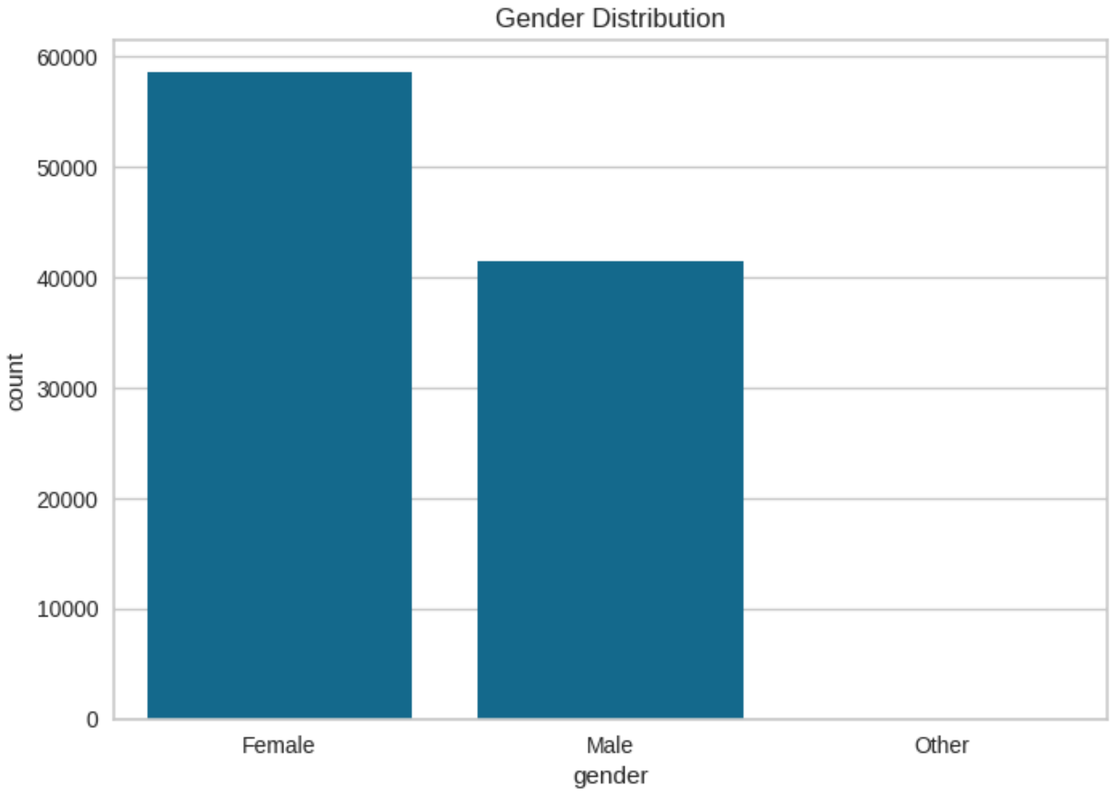




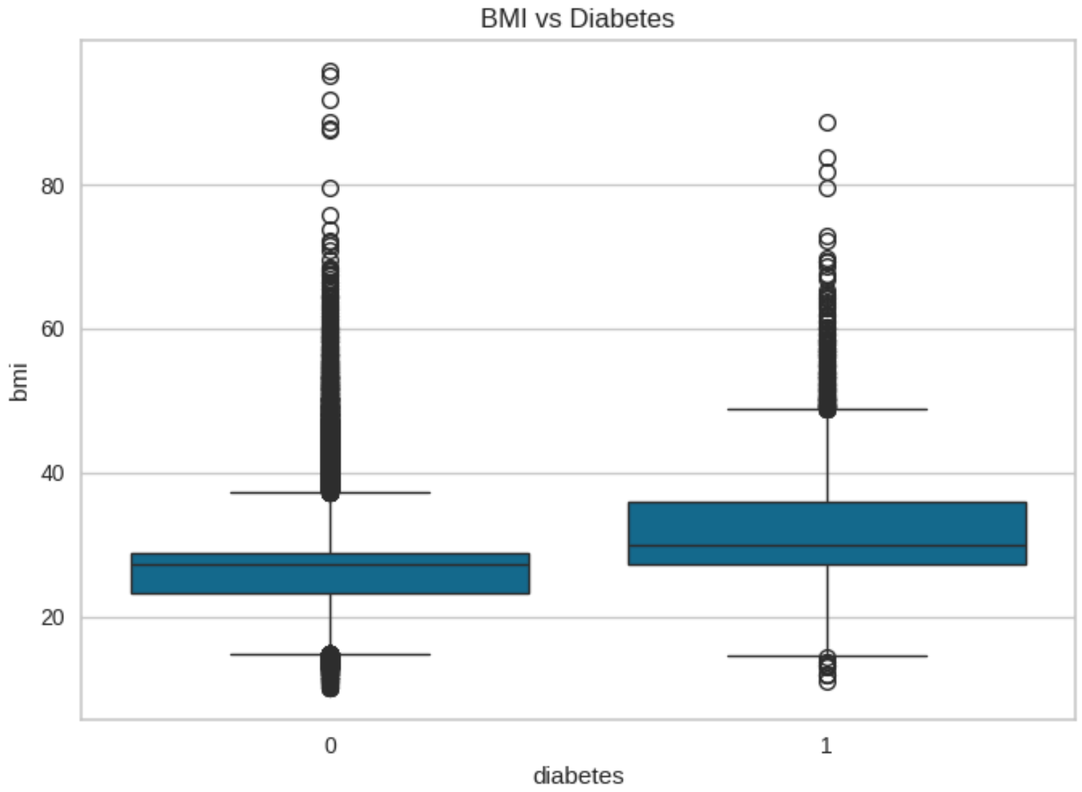


Analsis Exploratorio de Datos (EDA)





Analsis Exploratorio de Datos (EDA)



Ajuste de los Datos

```
# Seleccionar aleatoriamente 30,000 filas del dataset
df = df.sample(n=30000, random_state=42)
```

```
df.shape
```

```
(30000, 7)
```

```
# seed sera nuestro valor para nuestro random state
seed = 42

# División de los datos
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=seed)
x_train, x_val,y_train, y_val = train_test_split(x_train, y_train, test_size=0.25, random_state=seed)
```

```
# Entrenando el escalador
scaler = MinMaxScaler()
scaler.fit(x_train)

# Usando el escalador entrenado para transformar los datos
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
x_val_scaled = scaler.transform(x_val)
```

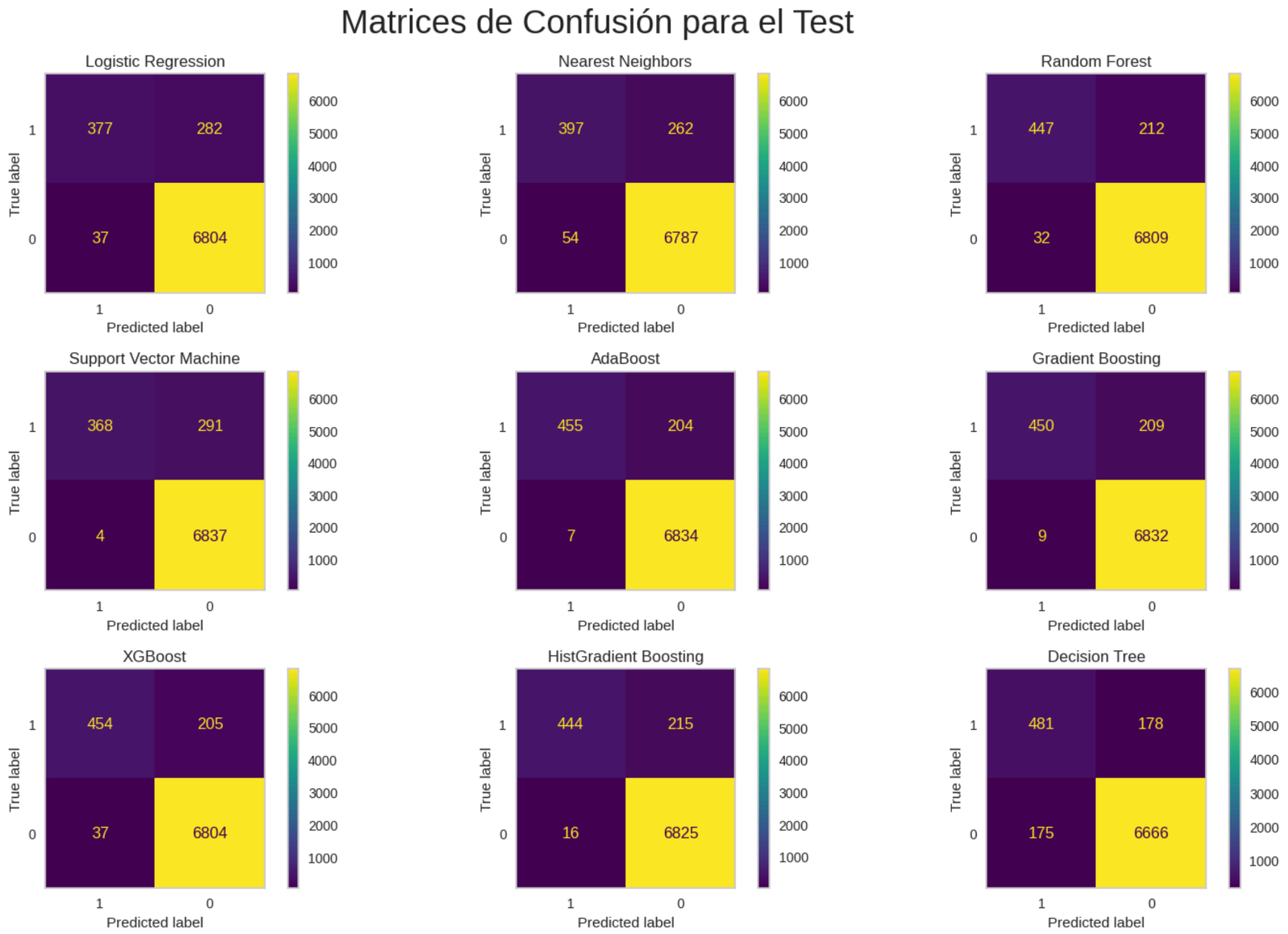
Modelos de Calsificación

```
names = [
    'Logistic Regression',
    'Nearest Neighbors',
    'Random Forest',
    'Support Vector Machine',
    'AdaBoost',
    'Gradient Boosting',
    'XGBoost',
    'HistGradient Boosting',
    'Decision Tree'
]

# Modelos de Clasificación

classifierss = [
    LogisticRegression(solver="liblinear", random_state=seed),
    KNeighborsClassifier(5),  # El 5 es el número de vecinos cercanos que queremos que halle
    RandomForestClassifier(random_state=seed),
    SVC(probability=True),
    AdaBoostClassifier(random_state=seed),
    GradientBoostingClassifier(random_state=seed),
    xgb.XGBClassifier(random_state=seed),
    HistGradientBoostingClassifier(random_state=seed),
    DecisionTreeClassifier(random_state=seed)
]
```

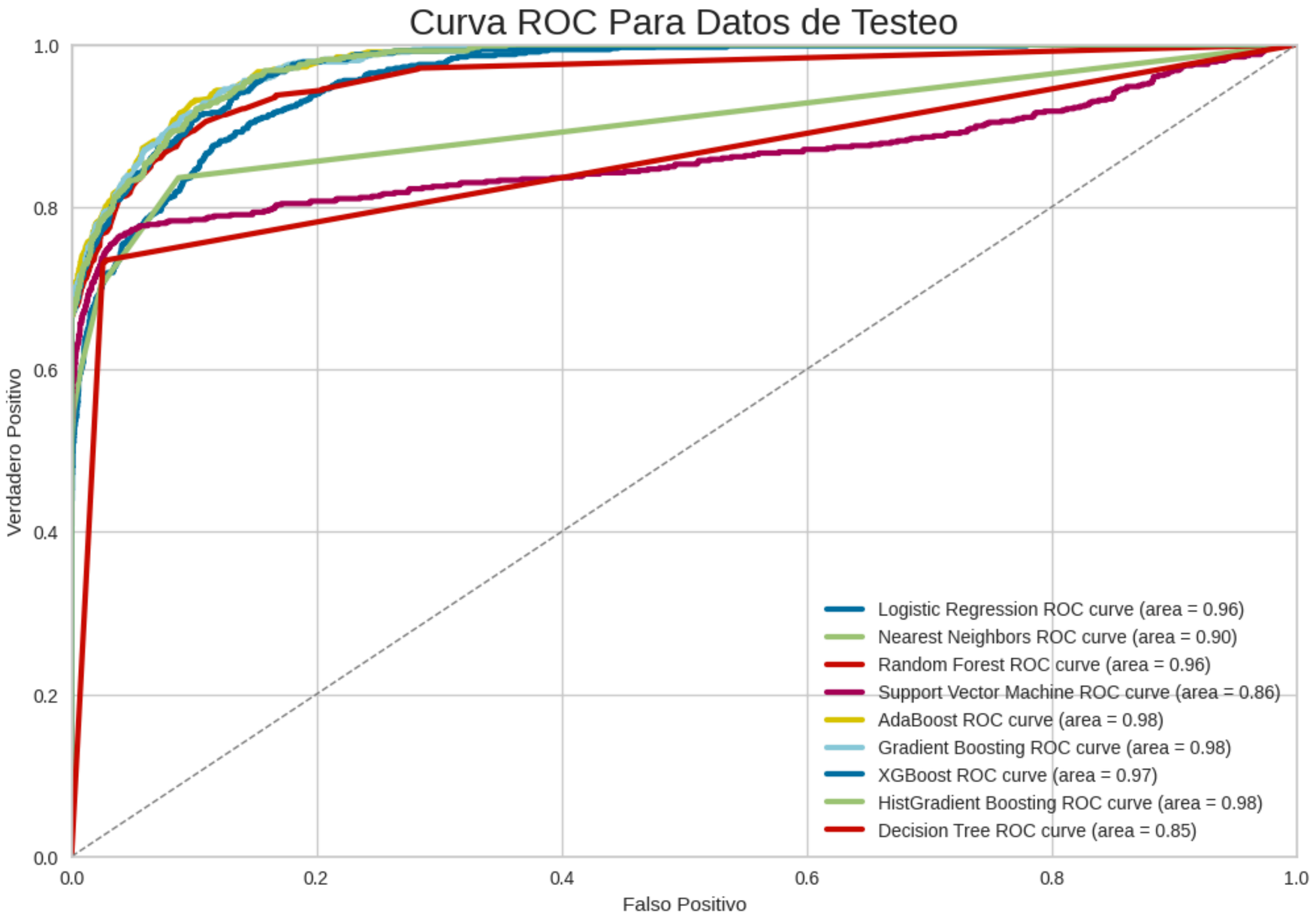
Voting para el Test

	Clasificador	Accuracy	ROC_AUC	Recall	Precision	F1
0	Logistic Regression	95.75	0.96	0.57	0.91	0.70
1	Nearest Neighbors	95.79	0.90	0.60	0.88	0.72
2	Random Forest	96.75	0.96	0.68	0.93	0.79
3	Support Vector Machine	96.07	0.86	0.56	0.99	0.71
4	AdaBoost	97.19	0.98	0.69	0.98	0.81
5	Gradient Boosting	97.09	0.98	0.68	0.98	0.81
6	XGBoost	96.77	0.97	0.69	0.92	0.79
7	HistGradient Boosting	96.92	0.98	0.67	0.97	0.79
8	Decision Tree	95.29	0.85	0.73	0.73	0.73

Matrices de Confusión



Curvas ROC para el Test

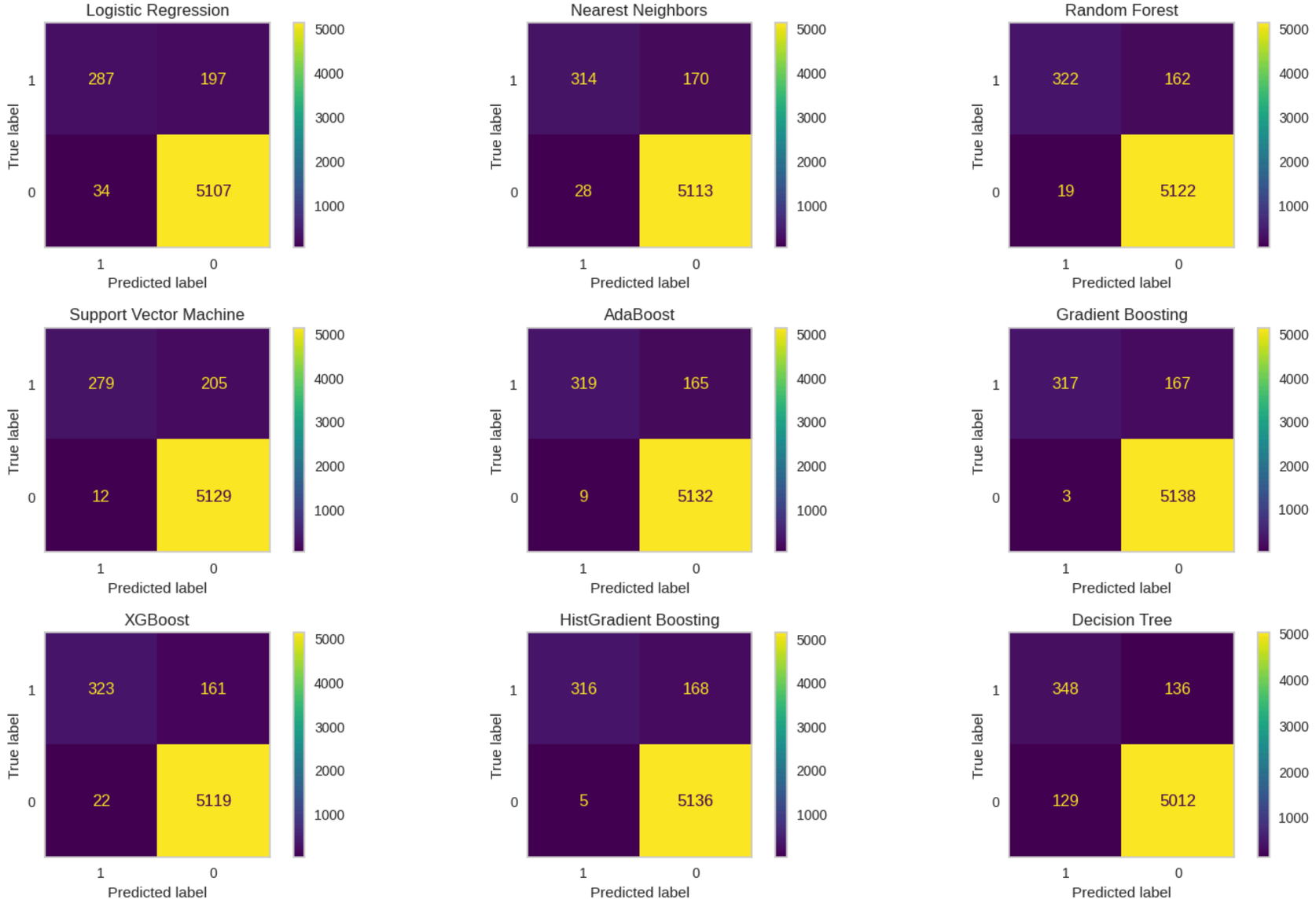


Voting para datos de Validación

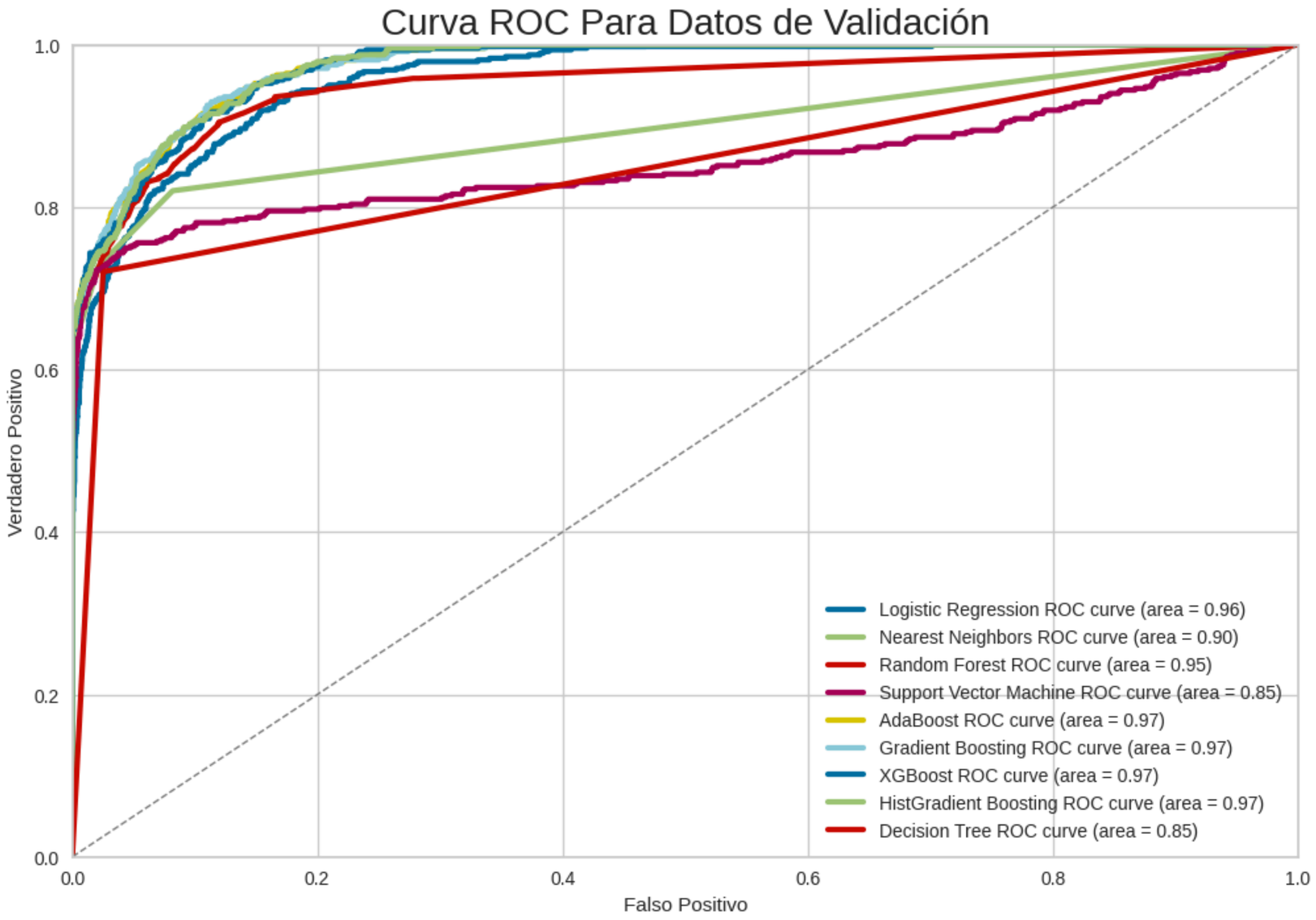
	Clasificador	Accuracy	ROC_AUC	Recall	Precision	F1
0	Logistic Regression	95.89	0.96	0.59	0.89	0.71
1	Nearest Neighbors	96.48	0.90	0.65	0.92	0.76
2	Random Forest	96.78	0.95	0.67	0.94	0.78
3	Support Vector Machine	96.14	0.85	0.58	0.96	0.72
4	AdaBoost	96.91	0.97	0.66	0.97	0.79
5	Gradient Boosting	96.98	0.97	0.65	0.99	0.79
6	XGBoost	96.75	0.97	0.67	0.94	0.78
7	HistGradient Boosting	96.92	0.97	0.65	0.98	0.79
8	Decision Tree	95.29	0.85	0.72	0.73	0.72

Matrices de Confusión

Matrices de Confusión para la Validación



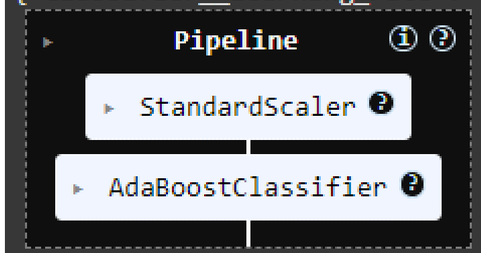
Curvas ROC para datos de Validación



Optimización de los Clasificadores

```python
# Parámetros para AdaBoostClassifier
params_adaboost = {
    'classifier__n_estimators': randint(50, 200),
    'classifier__learning_rate': [0.01, 0.1, 0.5, 1.0]
}

# Crear el pipeline
pipeline_adaboost = Pipeline([
    ('scale', StandardScaler()),
    ('classifier', AdaBoostClassifier(random_state=seed))
])

# RandomizedSearchCV para AdaBoostClassifier
adaboost_gs = RandomizedSearchCV(pipeline_adaboost, param_distributions=params_adaboost, n_iter=10, cv=7, scoring='accuracy', random_state=seed)
adaboost_gs.fit(x_train_scaled, y_train)

print(f"Mejores parámetros para AdaBoostClassifier (CV score=%0.3f):" % adaboost_gs.best_score_)
print(adaboost_gs.best_params_)

best_model_adaboost = adaboost_gs.best_estimator_
best_model_adaboost
```

```
Mejores parámetros para AdaBoostClassifier (CV score=0.973):
{'classifier__learning_rate': 0.1, 'classifier__n_estimators': 137}
```



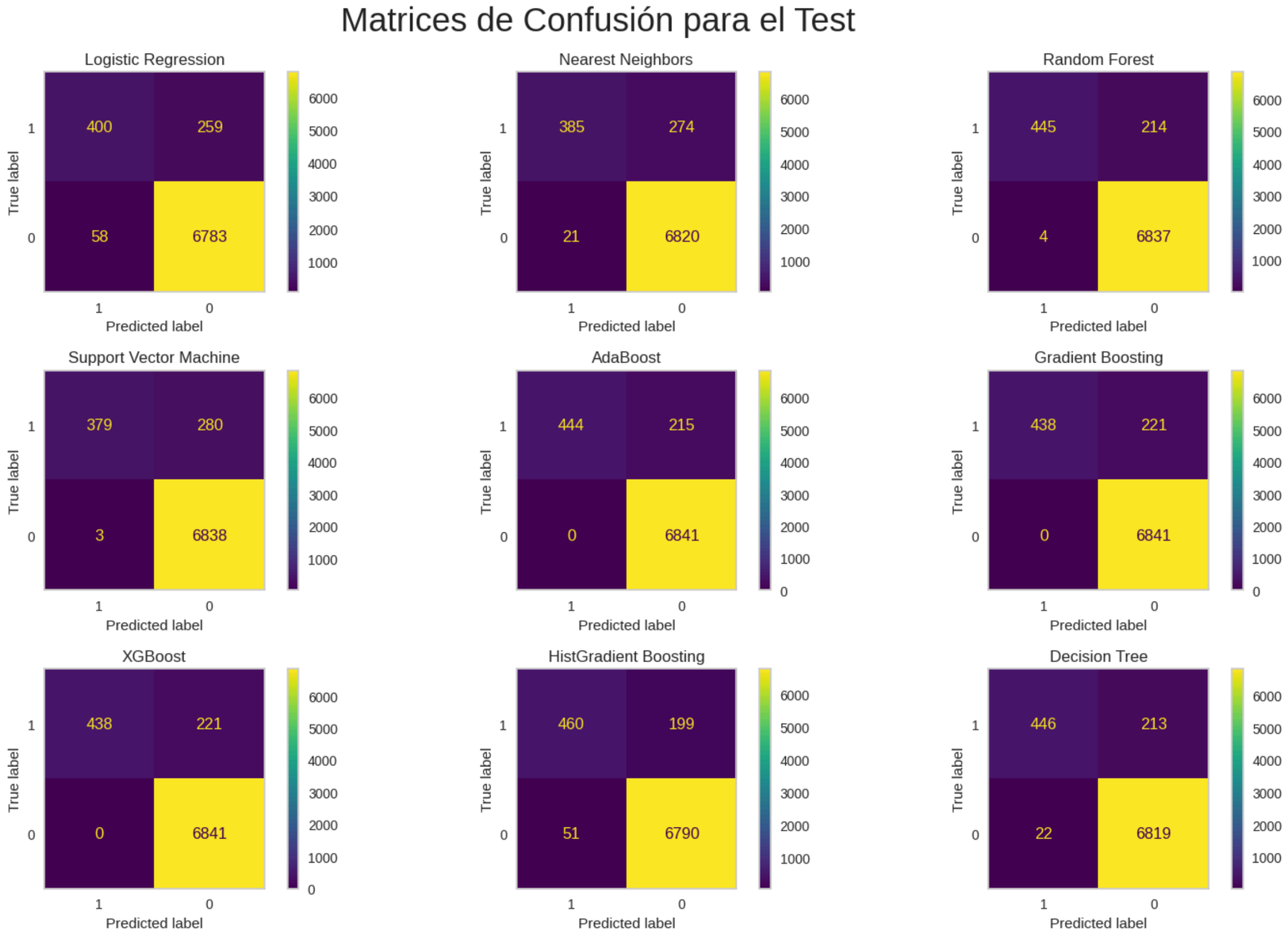
Re-Evaluación de los Clasificadores

```python
# Definir los clasificadores optimizados
names = [
    'Logistic Regression',
    'Nearest Neighbors',
    'Random Forest',
    'Support Vector Machine',
    'AdaBoost',
    'Gradient Boosting',
    'XGBoost',
    'HistGradient Boosting',
    'Decision Tree'
]

# Clasificadores Optimizados
classifiers_optim = [
    rgs.best_estimator_,            # Logistic Regression
    kgs.best_estimator_,            # Nearest Neighbors
    rfgs.best_estimator_,           # Random Forest
    svm_gs.best_estimator_,         # Support Vector Machine
    adaboost_gs.best_estimator_,    # AdaBoost
    gb_gs.best_estimator_,          # Gradient Boosting
    xgb_gs.best_estimator_,         # XGBoost
    hgb_gs.best_estimator_,         # HistGradient Boosting
    dt_gs.best_estimator_           # Decision Tree
]

# Evaluar cada clasificador optimizado por separado
for clf, label in zip(classifiers_optim, names):
    scores = cross_val_score(clf, x_train_scaled, y_train, scoring='accuracy', cv=5)
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

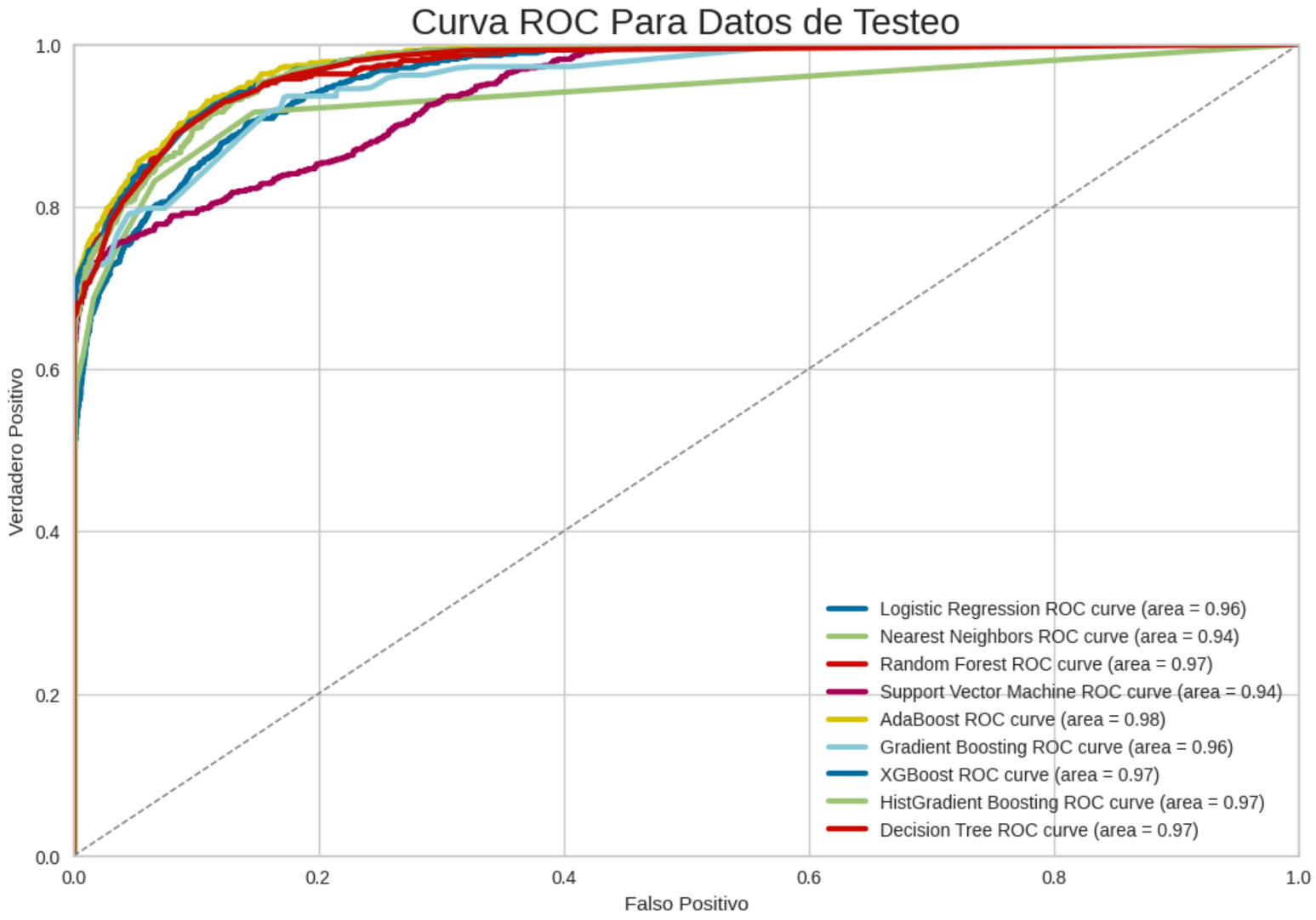
Metricas para Datos de Test

	Clasificador	Accuracy	ROC_AUC	Recall	Precision	F1
0	Logistic Regression	95.77	0.96	0.61	0.87	0.72
1	Nearest Neighbors	96.07	0.94	0.58	0.95	0.72
2	Random Forest	97.09	0.97	0.68	0.99	0.80
3	Support Vector Machine	96.23	0.94	0.58	0.99	0.73
4	AdaBoost	97.13	0.98	0.67	1.00	0.81
5	Gradient Boosting	97.05	0.96	0.66	1.00	0.80
6	XGBoost	97.05	0.97	0.66	1.00	0.80
7	HistGradient Boosting	96.67	0.97	0.70	0.90	0.79
8	Decision Tree	96.87	0.97	0.68	0.95	0.79

Matrices de Confusión



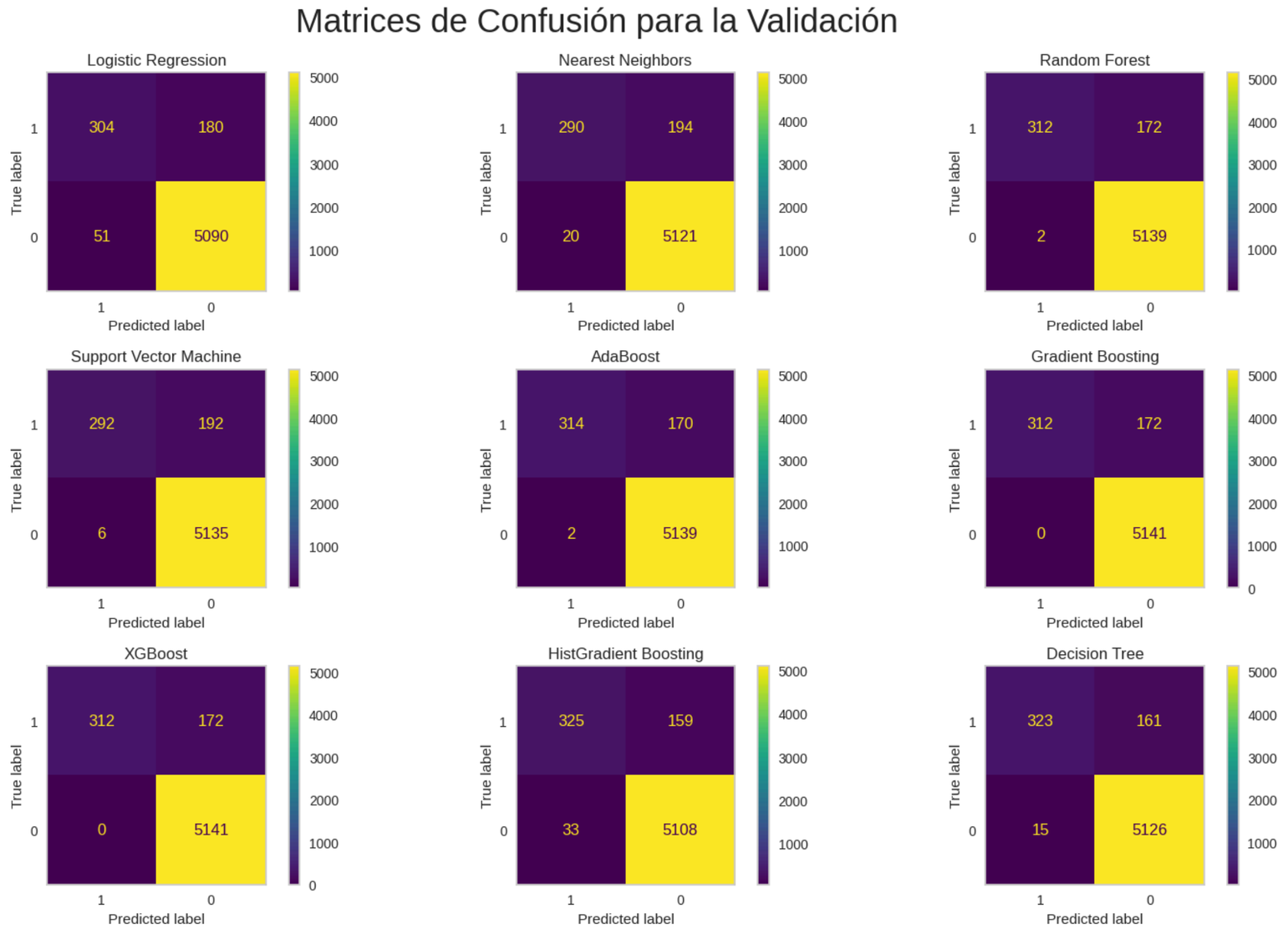
Curvas ROC para Datos de Test



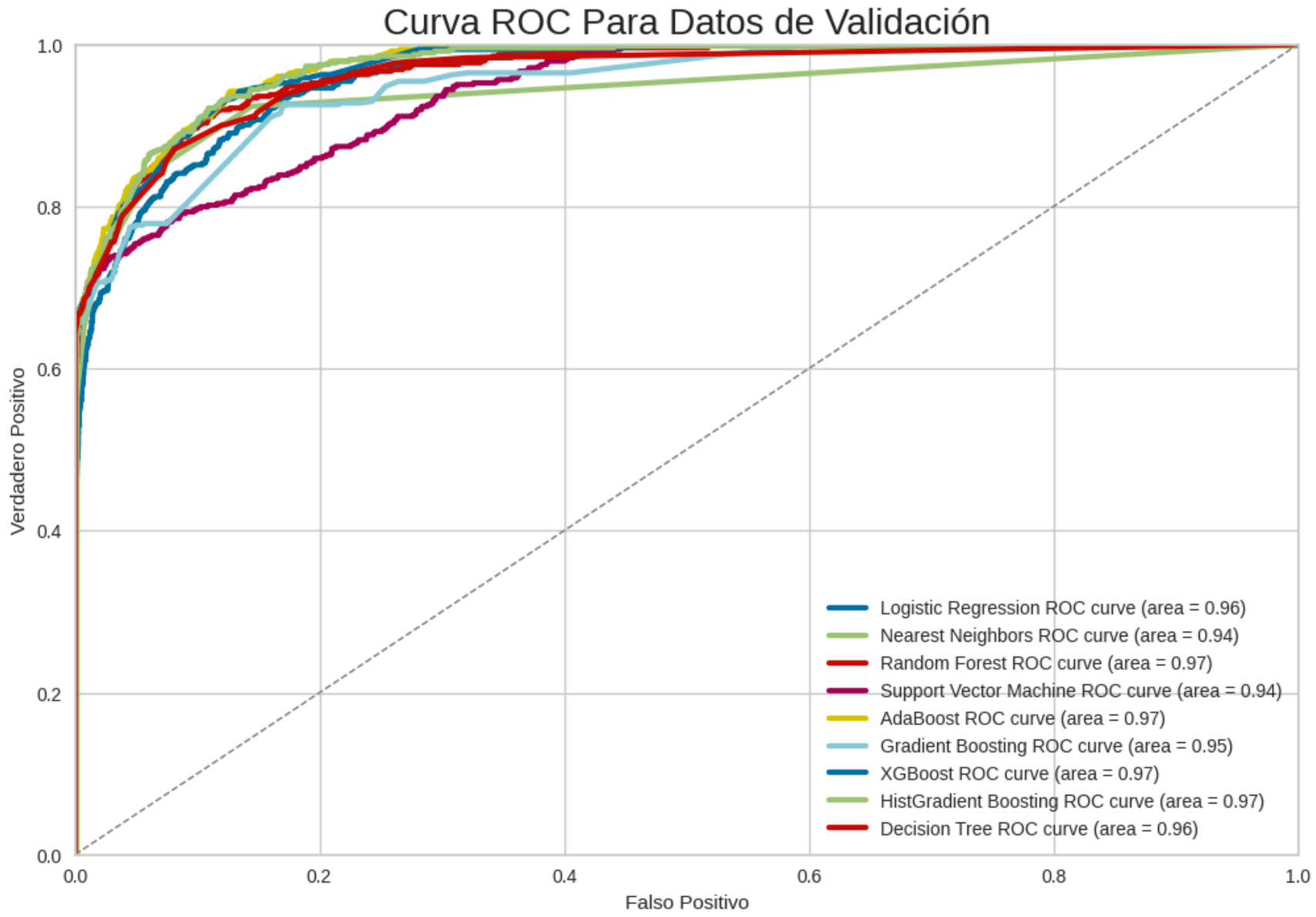
Metricas para Datos de Validación

	Clasificador	Accuracy	ROC_AUC	Recall	Precision	F1
0	Logistic Regression	95.89	0.96	0.63	0.86	0.72
1	Nearest Neighbors	96.20	0.94	0.60	0.94	0.73
2	Random Forest	96.91	0.97	0.64	0.99	0.78
3	Support Vector Machine	96.48	0.94	0.60	0.98	0.75
4	AdaBoost	96.94	0.97	0.65	0.99	0.78
5	Gradient Boosting	96.94	0.95	0.64	1.00	0.78
6	XGBoost	96.94	0.97	0.64	1.00	0.78
7	HistGradient Boosting	96.59	0.97	0.67	0.91	0.77
8	Decision Tree	96.87	0.96	0.67	0.96	0.79

Matrices de confusión para Datos de Validación



Curvas ROC para Datos de Validación



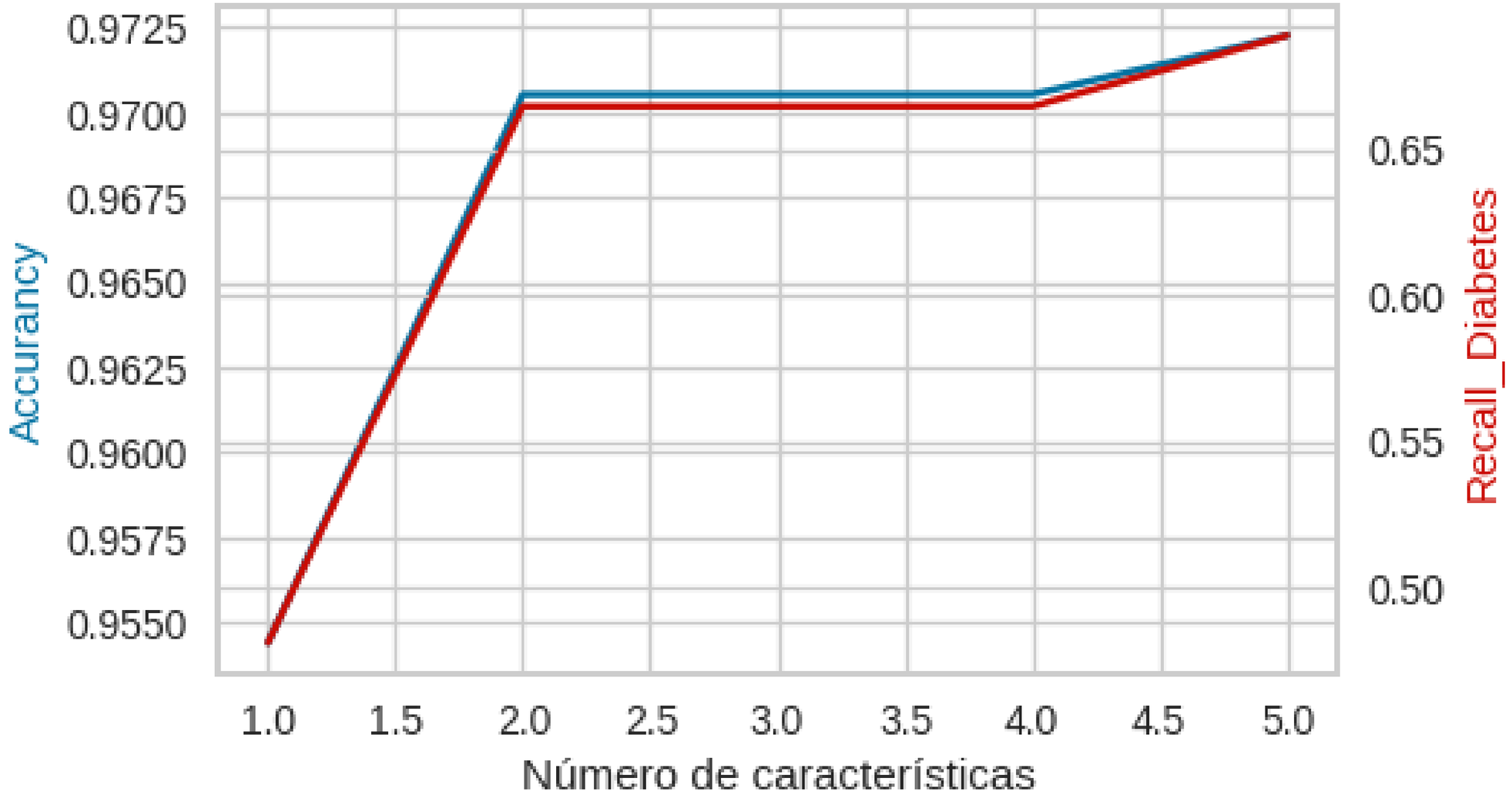
Metricas de Clasificación Múltiple para Datos de Test

	Clasificador	F1 Micro	F1 Macro	F1 Weighted
0	Logistic Regression	0.957733	0.846685	0.954236
1	Nearest Neighbors	0.960667	0.850917	0.956352
2	Random Forest	0.970933	0.893778	0.968399
3	Support Vector Machine	0.962267	0.853936	0.957621
4	AdaBoost	0.971333	0.894803	0.968761
5	Gradient Boosting	0.970533	0.891323	0.967799
6	XGBoost	0.970533	0.891323	0.967799
7	HistGradient Boosting	0.966667	0.884124	0.964737
8	Decision Tree	0.968667	0.887271	0.966227

Metricas de Clasificación Múltiple para Datos de Validación

	Clasificador	F1 Micro	F1 Macro	F1 Weighted
0	Logistic Regression	0.958933	0.851242	0.956031
1	Nearest Neighbors	0.961956	0.855006	0.958104
2	Random Forest	0.969067	0.882654	0.966023
3	Support Vector Machine	0.964800	0.863944	0.960927
4	AdaBoost	0.969422	0.884270	0.966457
5	Gradient Boosting	0.969422	0.883733	0.966370
6	XGBoost	0.969422	0.883733	0.966370
7	HistGradient Boosting	0.965867	0.876762	0.963519
8	Decision Tree	0.968711	0.884505	0.966151

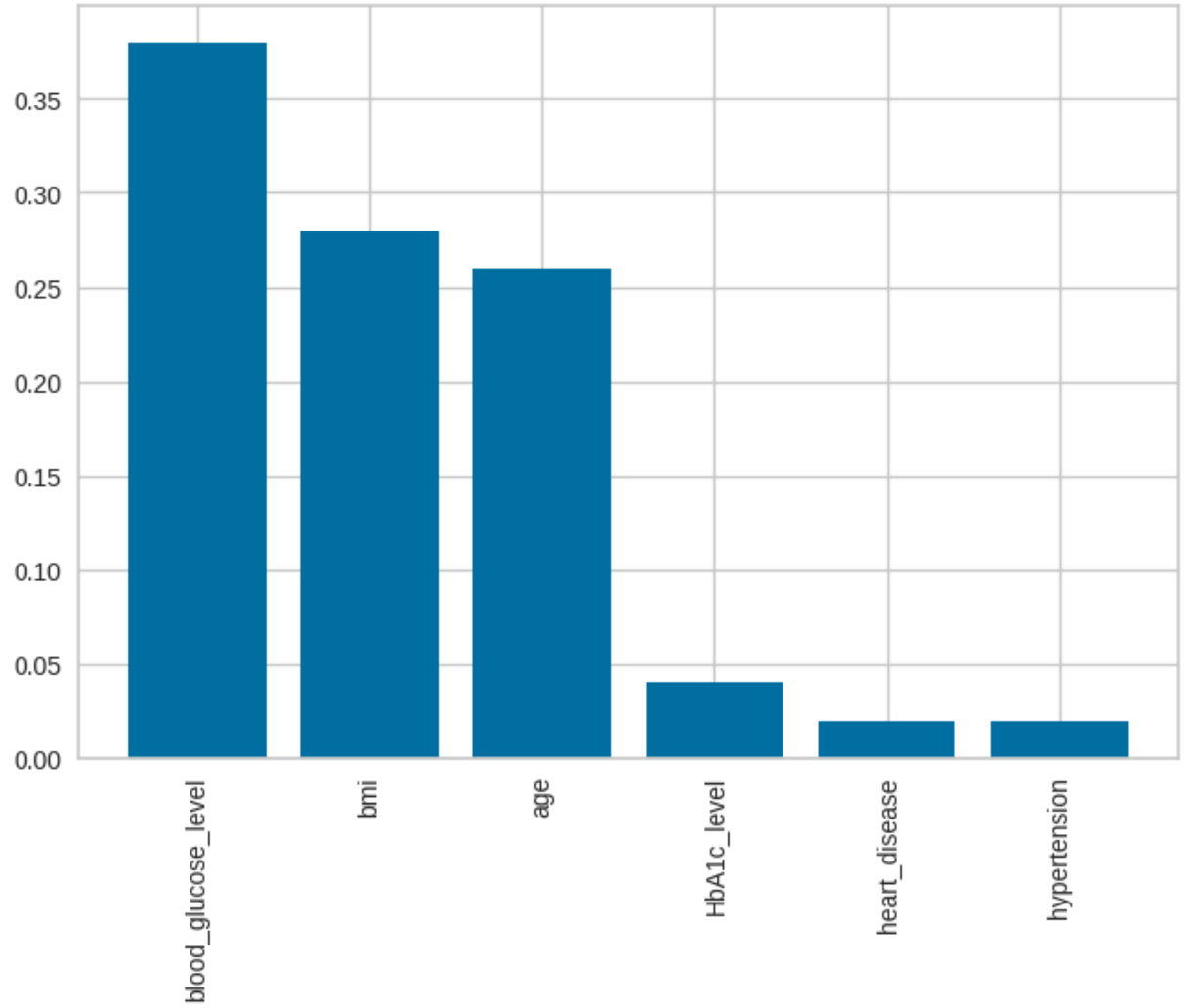
Selección de Caracteristicas



Selección de Caracteristicas

```python
# Escalador y pipeline
scaler = StandardScaler()
pipeline = make_pipeline(scaler, Ada_Boosting)
modelFit = pipeline.fit(x_train_scaled, y_train)
y_pred = pipeline.predict(x_test_scaled)

# Calcular nuevas métricas
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred, multi_class='ovr')
recall = recall_score(y_test, y_pred, average='weighted')
precision = precision_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Almacenar resultados
resultado_test = {
    'Model': Ada_Boosting,
    'Parameters': pipeline,
    'Accuracy': accuracy,
    'ROC_AUC': roc_auc,
    'Recall': recall,
    'Precision': precision,
    'F1': f1,
}
```

```python
importancia=Ada_Boosting.feature_importances_
```



Selección de Caracteristicas SeleckBest

```
k=6  #6 Número de características a seleccionar
selector = SelectKBest(score_func = mutual_info_classif, k=k)
selector.fit(x_train_scaled, y_train)
mask = selector.get_support()
new_x_train = x_train[x.columns[mask]] # Columnas definidas por
new_x_test = x_test[x.columns[mask]]
```

```
new_x_train.columns
```

```
Index(['age', 'hypertension', 'heart_disease', 'bmi', 'HbA1c_level',
       'blood_glucose_level'],
      dtype='object')
```

Selección de Caracteristicas Wrapper

```
#Usando Ada Boosting
rfe = RFE(estimator=AdaBoostClassifier(),n_features_to_select=8, step=1)
X_trans = rfe.fit(x, y)
rfe.get_support(True)
```

```
keep_list=rfe.get_support() # Listado True-False
columns_to_remove = x.columns.values[np.logical_not(keep_list)]
X_filter=x.drop(columns=columns_to_remove)

X_filter.columns
```

```
Index(['age', 'hypertension', 'heart_disease', 'bmi', 'HbA1c_level',
       'blood_glucose_level'],
      dtype='object')
```

CORRECCIONES

Se tomo en cuenta la petición de equilibrar las clases de salida, entonces se hizo un conteo de cuantos no enfermos (1) y enfermos (0) habian en total y se llego a que en la clase (1) solo habian 8500 datos entonces lo que se hizo fue tomar aleatoriamente otros 8500 de la clase (0) y asi trabajar con un total de 17mil datos

```python
# Separar las clases
data_class_0 = df[df['diabetes'] == 0]
data_class_1 = df[df['diabetes'] == 1]

# Submuestrear cada clase para que contengan la misma cantidad de instancias
data_class_0_sampled = resample(data_class_0, replace=False, n_samples=8500, random_state=42)
data_class_1_sampled = resample(data_class_1, replace=False, n_samples=8500, random_state=42)

# Combinar las muestras submuestreadas
df = pd.concat([data_class_0_sampled, data_class_1_sampled])

# Separar nuevamente x e y
x = df.drop('diabetes', axis=1)
y = df['diabetes']

print("Distribución balanceada de y:")
print(y.value_counts())
```

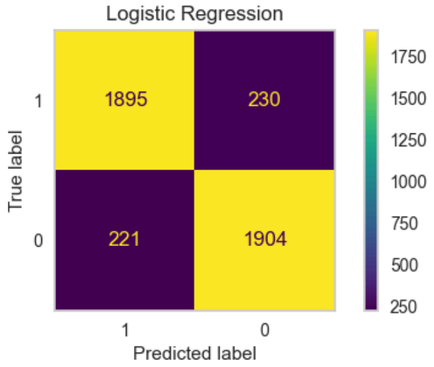
```
Distribución balanceada de y:
diabetes
0    8500
1    8500
Name: count, dtype: int64
```

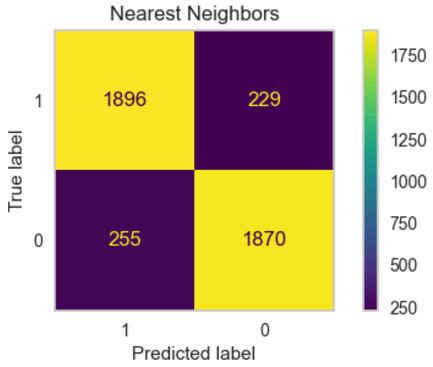
Se volvieron a correr los codigos buscando que los resultaros cambiaran ahora que las clases de salida estaban mas equilibradas y los resultados fueron los siguientes:

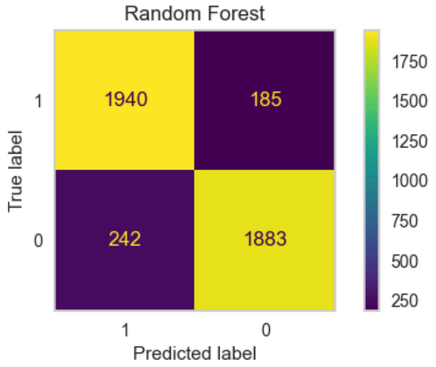
Para el test podemos observar un notorio cambio en los valores de las metricas con las clases desbalanceadas todos los metodos de clasificación estaban por encima de los 90, ahora solo los metodos de boosting logran estar por encima de 90 esto para el Accuracy pero tambien vemos mejoras como en el F1 que todos estan entre 0.89 y 0.92

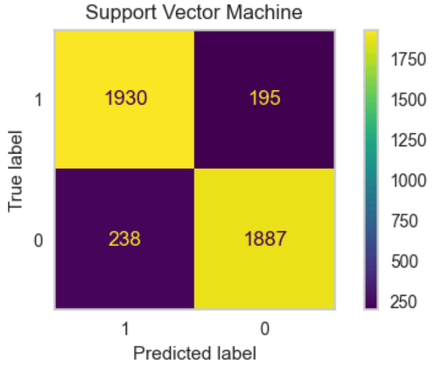
	Clasificador	Accuracy	ROC_AUC	Recall	Precision	F1
0	Logistic Regression	89.39	0.97	0.89	0.90	0.89
1	Nearest Neighbors	88.61	0.96	0.89	0.88	0.89
2	Random Forest	89.95	0.97	0.91	0.89	0.90
3	Support Vector Machine	89.81	0.97	0.91	0.89	0.90
4	AdaBoost	91.27	0.98	0.94	0.89	0.91
5	Gradient Boosting	91.29	0.98	0.94	0.89	0.92
6	XGBoost	90.78	0.98	0.92	0.90	0.91
7	HistGradient Boosting	91.11	0.98	0.93	0.90	0.91
8	Decision Tree	87.86	0.88	0.88	0.87	0.88

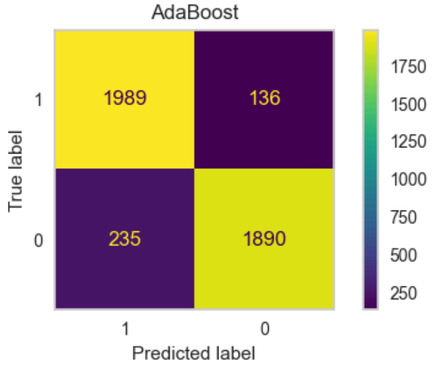
Matrices de Confusión para el Test

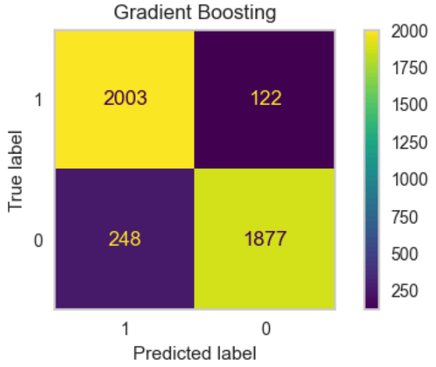


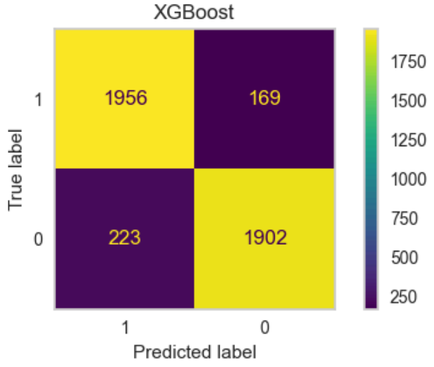


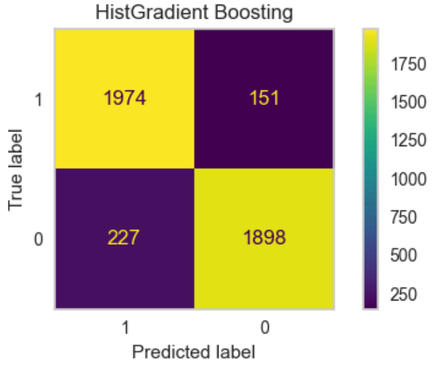


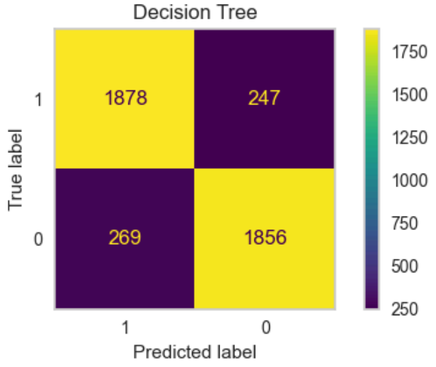








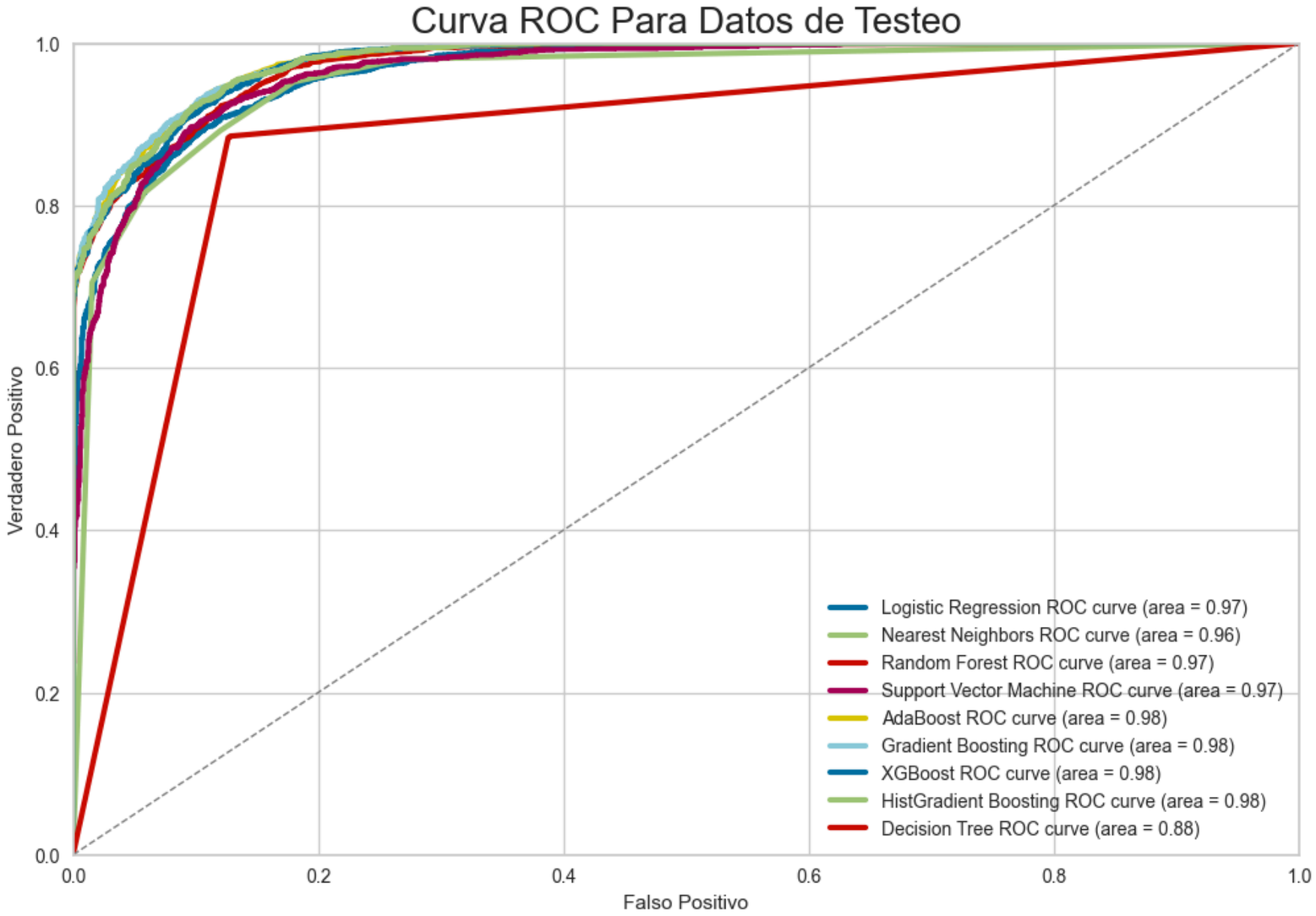





```
x_test.shape
```

```
(4250, 6)
```

Se puede observar como hay una mejora significativa en cuanto a la predicción de las dos clases, antes solo predecia bien una clase, ahora hay buenas predicciones para las dos clases



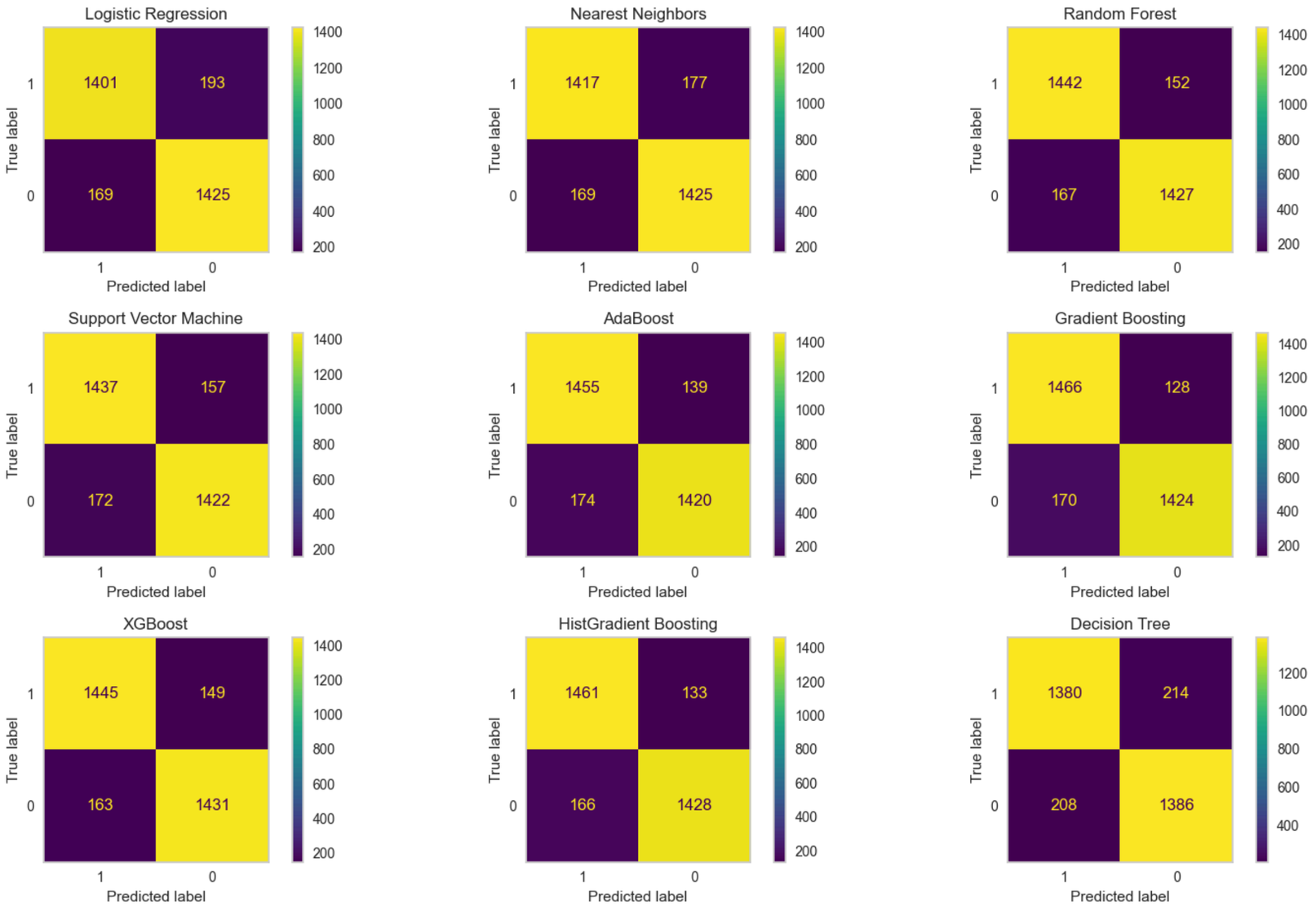
Las curvas muestran una mejoria muy buena tambien a excepción de los arboles de decision que sigue presentando ese cambio abrupto pero sigue teniendo una buena relación todo lo que implica la tener verdaderos positivos.

PARA LA VALIDACIÓN

	Clasificador	Accuracy	ROC_AUC	Recall	Precision	F1
0	Logistic Regression	88.64	0.96	0.88	0.89	0.89
1	Nearest Neighbors	89.15	0.95	0.89	0.89	0.89
2	Random Forest	89.99	0.97	0.90	0.90	0.90
3	Support Vector Machine	89.68	0.96	0.90	0.89	0.90
4	AdaBoost	90.18	0.98	0.91	0.89	0.90
5	Gradient Boosting	90.65	0.98	0.92	0.90	0.91
6	XGBoost	90.21	0.97	0.91	0.90	0.90
7	HistGradient Boosting	90.62	0.98	0.92	0.90	0.91
8	Decision Tree	86.76	0.87	0.87	0.87	0.87

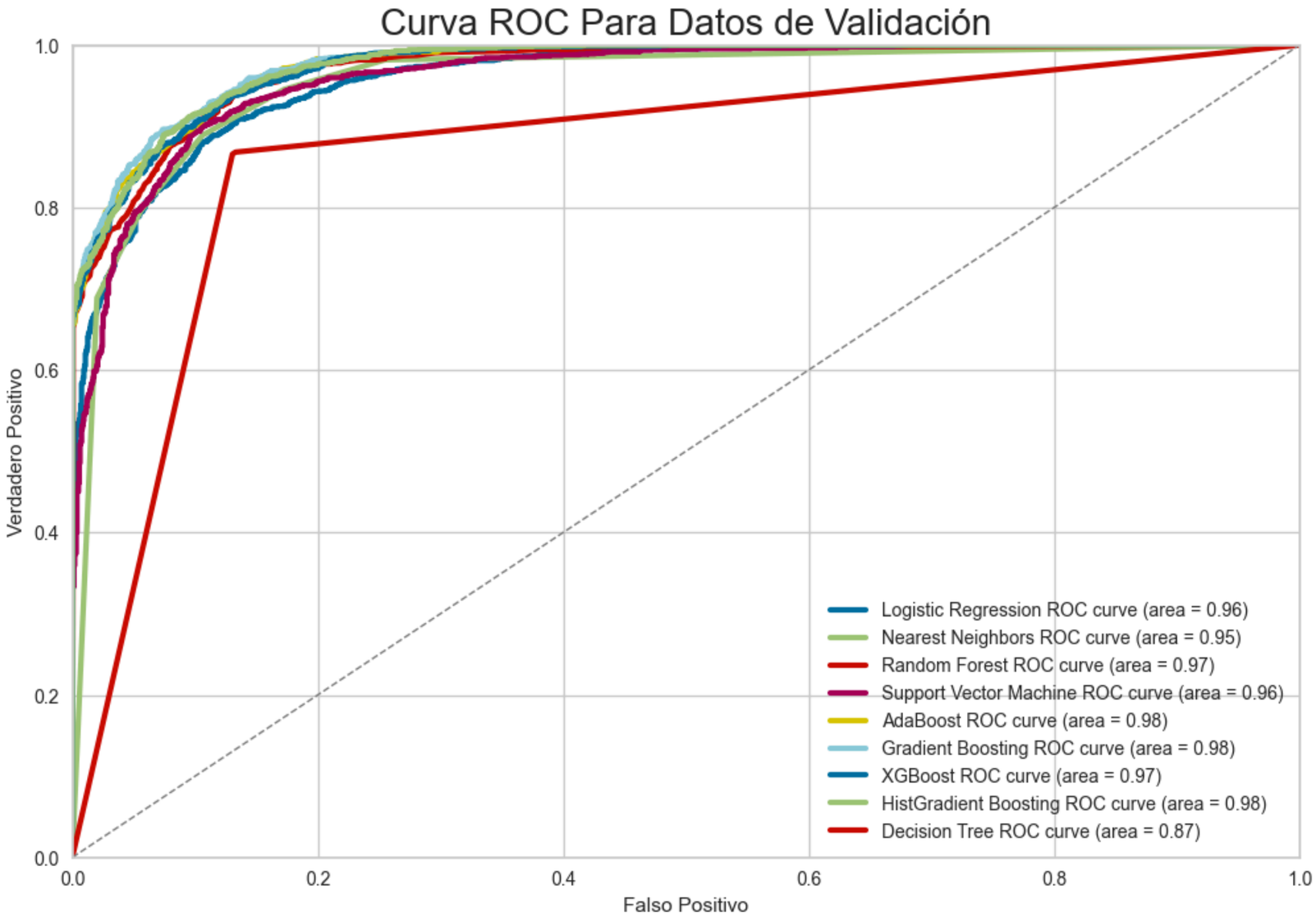
Tenemos el mismo comportamiento que en los datos de testeo, los valores de precisión bajaron peor siguen siendo buenos, y los valores de F1 subieron.

Matrices de Confusión para la Validación




```
x_val.shape

(3188, 6)
```

Tenemos mejores resultados que en el testeo, ya que la mayoria de clasificadores estan por debajo de 200 falsos negativos.

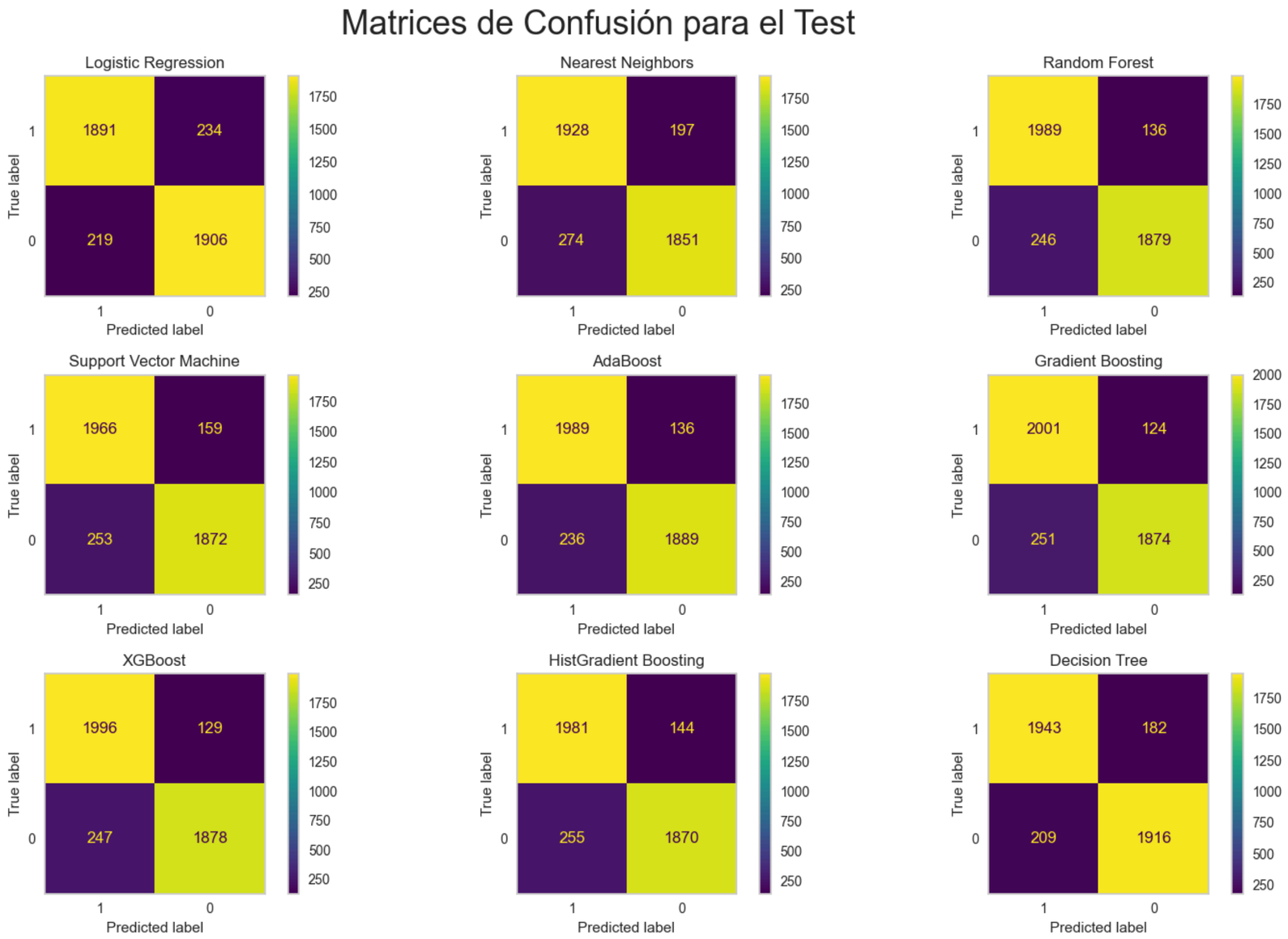


Para las curvas ROC vemos que tenemos comportamientos muy similares con los datos de testeo, no hay mas que agragar ya que tambien tienen un buen desempeño.

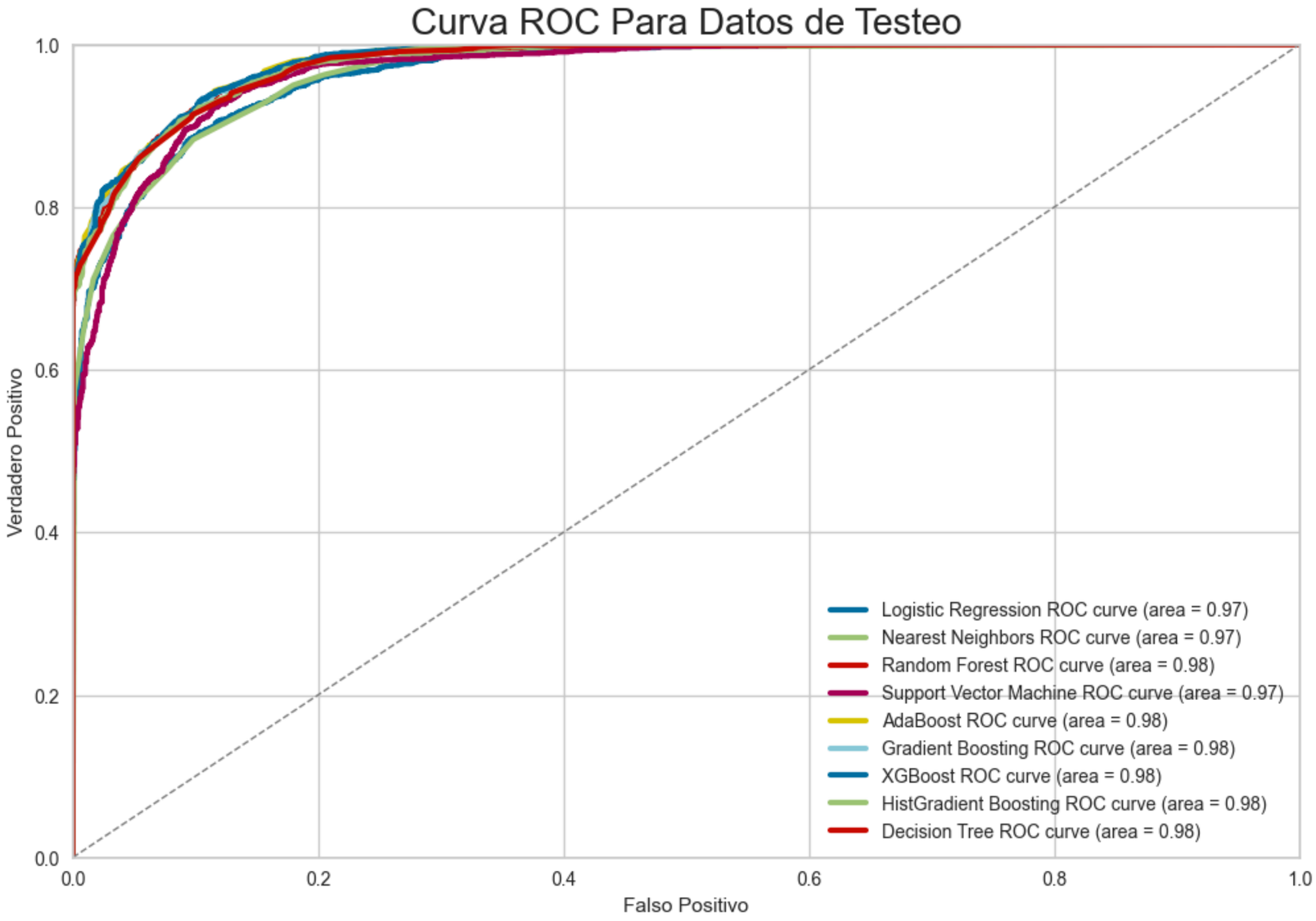
OPTIMIZACIÓN TESTEO

	Clasificador	Accuracy	ROC_AUC	Recall	Precision	F1
0	Logistic Regression	89.34	0.97	0.89	0.90	0.89
1	Nearest Neighbors	88.92	0.97	0.91	0.88	0.89
2	Random Forest	91.01	0.98	0.94	0.89	0.91
3	Support Vector Machine	90.31	0.97	0.93	0.89	0.91
4	AdaBoost	91.25	0.98	0.94	0.89	0.91
5	Gradient Boosting	91.18	0.98	0.94	0.89	0.91
6	XGBoost	91.15	0.98	0.94	0.89	0.91
7	HistGradient Boosting	90.61	0.98	0.93	0.89	0.91
8	Decision Tree	90.80	0.98	0.91	0.90	0.91

Para los datos de testeo vemos como hay una mejora considerable y la gran mayoria de los clasificadores ahora tienen valores bastante buenos, los de menor desempeño son la regresión logistica y el KNN.



La verdad en cuanto a las predicciones no se nota realmente un cambio, ya que siguen teniendo la misma cantidad de valores acertados, si se nota que en algunos casos hay pores resultados aumentando los casos de falsos negativos y la mejor clase que se desempeña es la clase (1) enfermos.

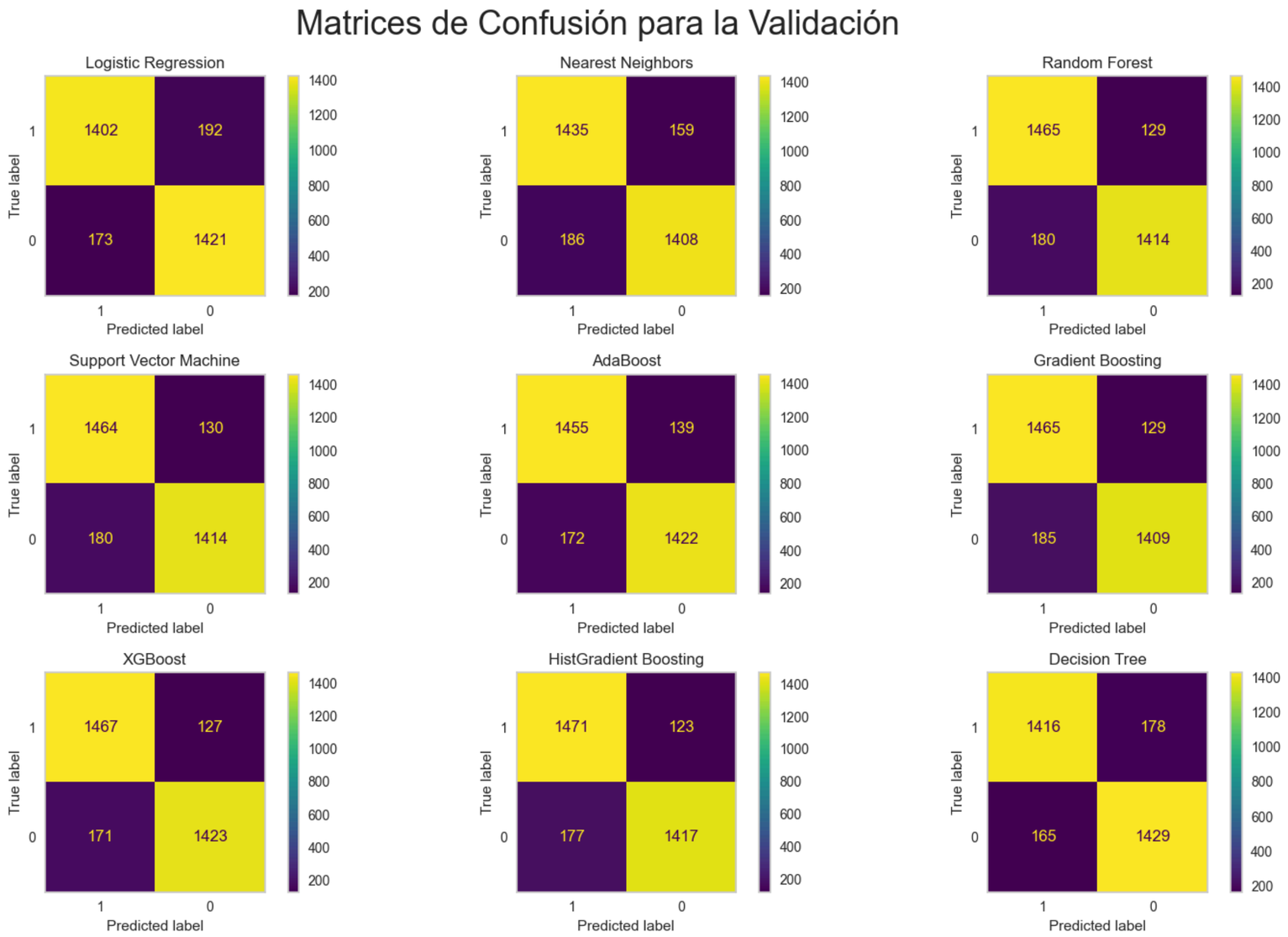


Las curvas muestran una muy buena mejoria sobretodo en la curva del arbol de desición que ya se ajusta mucho mejor

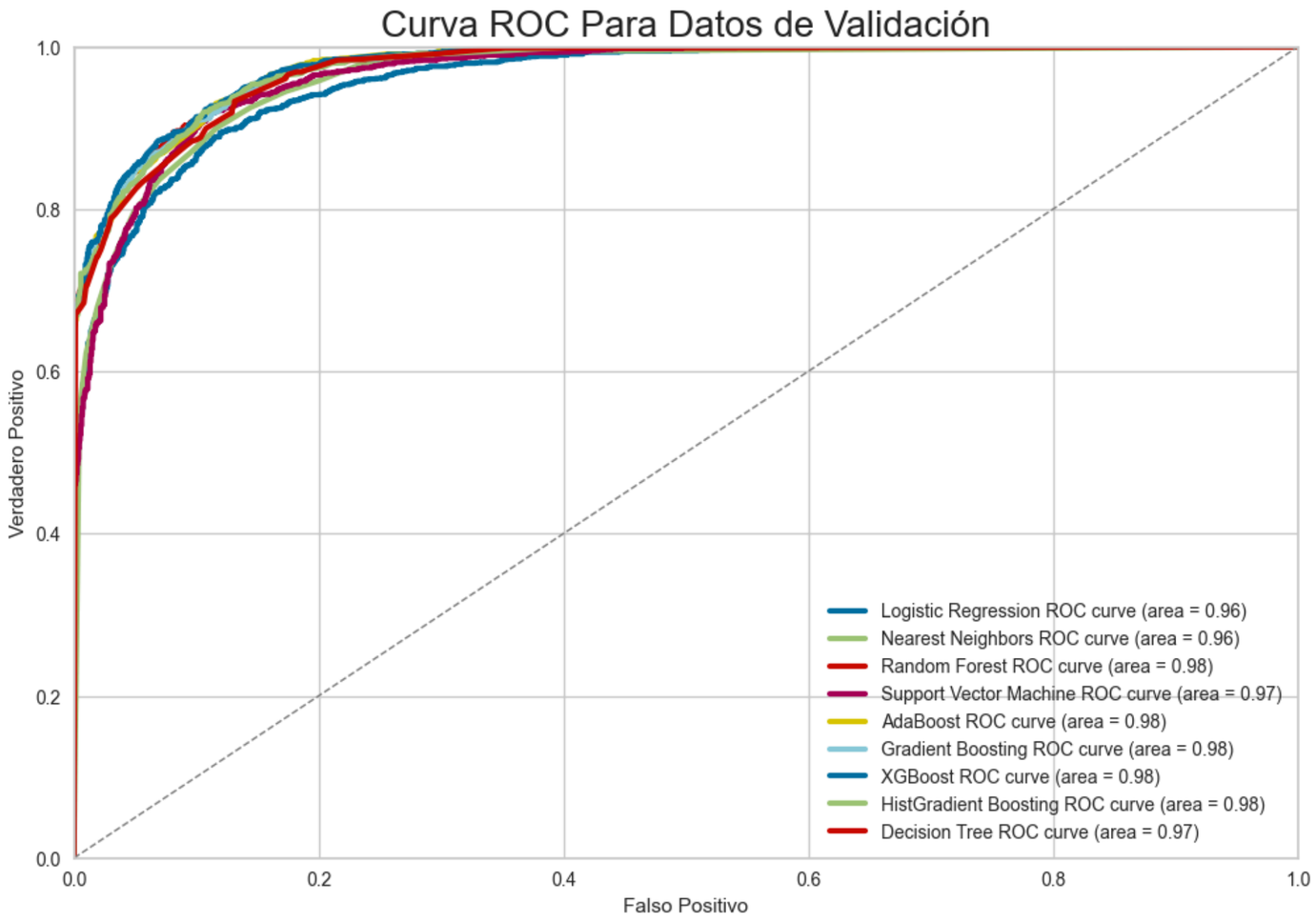
OPTIMIZACIÓN VALIDACIÓN

	Clasificador	Accuracy	ROC_AUC	Recall	Precision	F1
0	Logistic Regression	88.55	0.96	0.88	0.89	0.88
1	Nearest Neighbors	89.18	0.96	0.90	0.89	0.89
2	Random Forest	90.31	0.98	0.92	0.89	0.90
3	Support Vector Machine	90.28	0.97	0.92	0.89	0.90
4	AdaBoost	90.24	0.98	0.91	0.89	0.90
5	Gradient Boosting	90.15	0.98	0.92	0.89	0.90
6	XGBoost	90.65	0.98	0.92	0.90	0.91
7	HistGradient Boosting	90.59	0.98	0.92	0.89	0.91
8	Decision Tree	89.24	0.97	0.89	0.90	0.89

Para los datos de validación vemos tambien que hay una leve mejora en casi todos los clasificadores, peor vemos como metricas como la precisión bajaron a 0.89 en casi todos.



Haciendo la comparación tambien nos damos cuenta que no hay una gran mejora al momento de predecir, en algunos casos mejora y en otros empeora pero no son cambios despreciables.



Por ultiomo las graficas tambien presentan un muy buen comportamiento a comparación de sus predecesores.

METRICAS F1 PARA EL TEST

	Clasificador	F1 Micro	F1 Macro	F1 Weighted
0	Logistic Regression	0.893412	0.893410	0.893410
1	Nearest Neighbors	0.889176	0.889140	0.889140
2	Random Forest	0.910118	0.910057	0.910057
3	Support Vector Machine	0.903059	0.903011	0.903011
4	AdaBoost	0.912471	0.912422	0.912422
5	Gradient Boosting	0.911765	0.911686	0.911686
6	XGBoost	0.911529	0.911461	0.911461
7	HistGradient Boosting	0.906118	0.906054	0.906054
8	Decision Tree	0.908000	0.907996	0.907996

Haciendo una comparativa con las metricas con clases desbalanceadas hay mejoras en algunos clasificadores en la medidas F1 macro mejoran pero por lo menos en la medida F1 micro todos bajan de una media de 0.96 a una media de 0.90

METRICAS F1 PARA LA VALIDACIÓN

	Clasificador	F1 Micro	F1 Macro	F1 Weighted
0	Logistic Regression	0.885508	0.885504	0.885504
1	Nearest Neighbors	0.891782	0.891774	0.891774
2	Random Forest	0.903074	0.903049	0.903049
3	Support Vector Machine	0.902760	0.902736	0.902736
4	AdaBoost	0.902447	0.902436	0.902436
5	Gradient Boosting	0.901506	0.901475	0.901475
6	XGBoost	0.906524	0.906507	0.906507
7	HistGradient Boosting	0.905897	0.905870	0.905870
8	Decision Tree	0.892409	0.892407	0.892407

Tenemos el mismo comportamiento que para los datos de testeo, en todas las medidas de F1 hay bajones en sus puntaciones y son considerables ya que para la medida F1 micro teniamos medias de 0.95 y ahora tenemos medias de 0.89.

VOTING CLASSIFIER TEST

```python
from sklearn.ensemble import VotingClassifier

# Definir el VotingClassifier
voting_clf = VotingClassifier(
    estimators=[
        ('Logistic Regression', rgs.best_estimator_),
        ('Nearest Neighbors', kgs.best_estimator_),
        ('Random Forest', rfgs.best_estimator_),
        ('Support Vector Machine', svm_gs.best_estimator_),
        ('AdaBoost', adaboost_gs.best_estimator_),
        ('Gradient Boosting', gb_gs.best_estimator_),
        ('XGBoost', xgb_gs.best_estimator_),
        ('HistGradient Boosting', hgb_gs.best_estimator_),
        ('Decision Tree', dt_gs.best_estimator_)
    ],
    voting='hard'
)

# Entrenar el VotingClassifier
voting_clf.fit(x_train_scaled, y_train)

# Predecir con el VotingClassifier
y_pred = voting_clf.predict(x_test_scaled)

# Evaluar las métricas
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred, multi_class='ovr')
recall = recall_score(y_test, y_pred, average='weighted')
precision = precision_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
```

```
'Accuracy': 0.9117647058823529,
'ROC_AUC': np.float64(0.911764705882353),
'Recall': np.float64(0.9117647058823529),
'Precision': np.float64(0.912974191542288 5),
'F1': np.float64(0.9117000544343195)}
```

Se añadio el Voting Classifier para hacer tambien la comparativa con los demas metodos de clasificación en donde termina siendo muy bueno pero el que mejor se desempeña es el AdaBoosting.

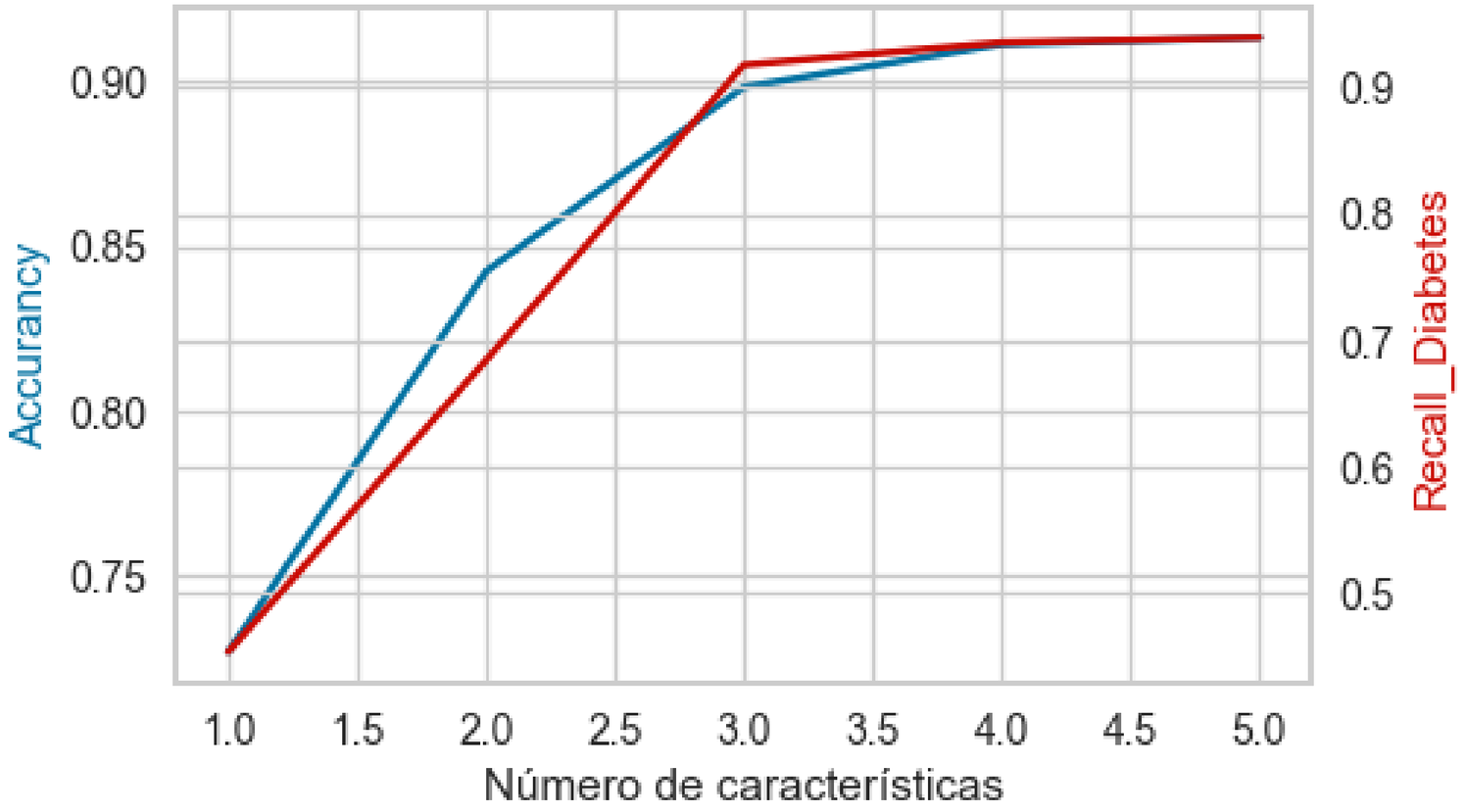
VOTING CLASSIFIER VALIDACIÓN

```python
# Definir el VotingClassifier
voting_clf = VotingClassifier(
    estimators=[
        ('Logistic Regression', rgs.best_estimator_),
        ('Nearest Neighbors', kgs.best_estimator_),
        ('Random Forest', rfgs.best_estimator_),
        ('Support Vector Machine', svm_gs.best_estimator_),
        ('AdaBoost', adaboost_gs.best_estimator_),
        ('Gradient Boosting', gb_gs.best_estimator_),
        ('XGBoost', xgb_gs.best_estimator_),
        ('HistGradient Boosting', hgb_gs.best_estimator_),
        ('Decision Tree', dt_gs.best_estimator_)
    ],
    voting='hard'
)

# Entrenar el VotingClassifier
voting_clf.fit(x_val_scaled, y_val)

# Predecir con el VotingClassifier
y_pred = voting_clf.predict(x_val_scaled)

# Evaluar las métricas
accuracy = accuracy_score(y_val, y_pred)
roc_auc = roc_auc_score(y_val, y_pred, multi_class='ovr')
recall = recall_score(y_val, y_pred, average='weighted')
precision = precision_score(y_val, y_pred, average='weighted')
f1 = f1_score(y_val, y_pred, average='weighted')
```

```
'Accuracy': 0.9190715181932246,
'ROC_AUC': np.float64(0.9190715181932245),
'Recall': np.float64(0.9190715181932246),
'Precision': np.float64(0.9192404792165719),
'F1': np.float64(0.9190633634839288)}
```

Haciendo la comparación con la clasificación para los datos de test podemos ver que sus resultados en las metricas son practicamente iguales y solo difieren en el tercer decimal hacia adelante.

Selección de Caracteristicas

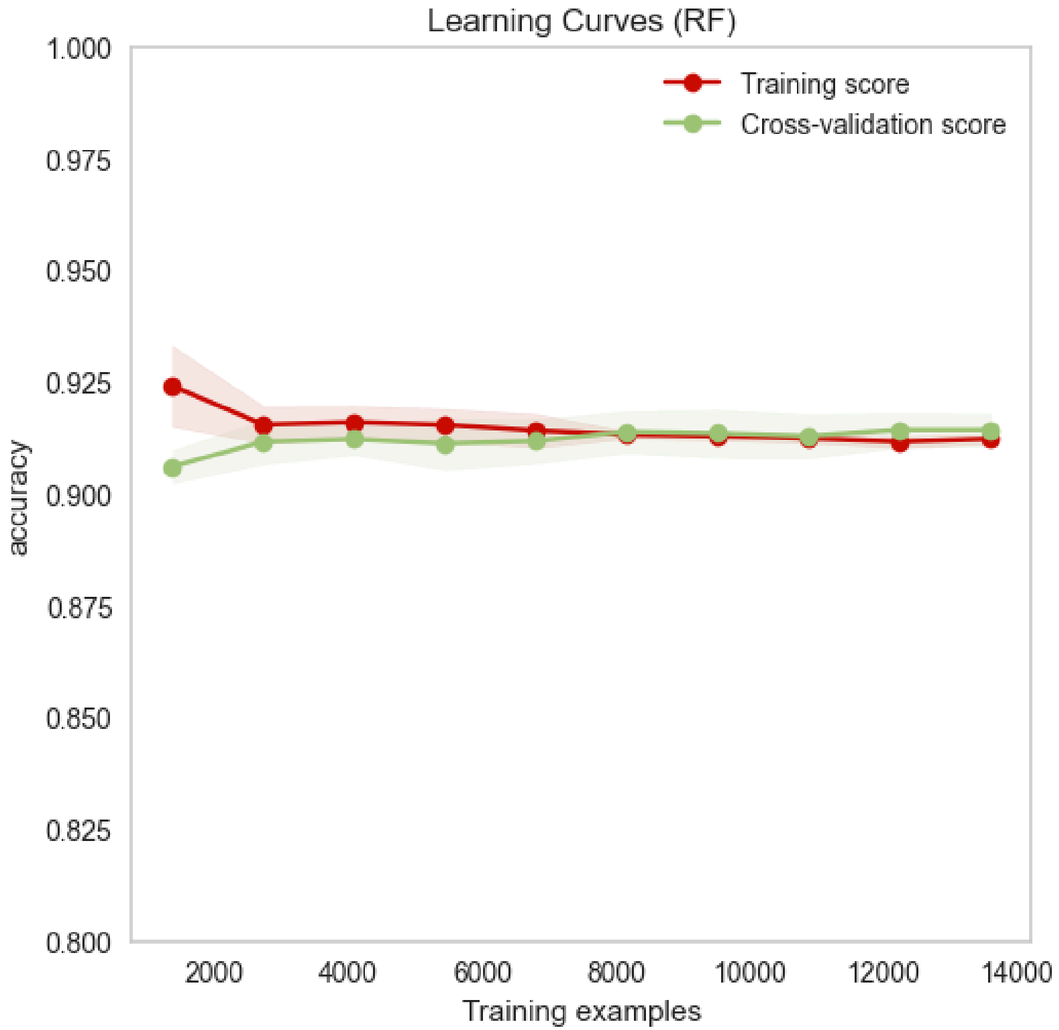


En el caso con las clases desbalanceadas la grafica llegaba a tener un valor de accurancy de 0.97 y ahora solo pasa del 0.90 que no es malo pero si hubo una disminución en la exactitud.

CONCLUSIÓN

Se concluye que el modelo de clasificación que mejor se desempeña es el AdaBoosting siendo el que mejor sobre sale entre todos los metodos, no opstante los demas metodos de Boosting solo se quedan por detras por unas cuantas decimas, las curvas ROC tiene comportamientos muy buenos lo que nos indica que saben distinguir de mejor manera al momento de clasificar un paciente como sano o enfermo. Por otro lado teniendo en cuenta que los resultados de las matrices la cantidad en porcentaje de los falsos positivos y negativos se matienen entre un 15% y 20% de el total de los datos asignados.

CURVA DE APRENDIAZJE PARA EL MEJOR MODELO



Para el mejor modelo que es el AdaBoos, vemos como su curva de aprendizaje se ubica en un puntaje de exactitud muy bueno no superior al 0.925. y lo que mas nos agrada es ver como el overfithing es casi inexistente y va disminuyendo con el aunmento de valores de entrenamiento. Esto solo corrobora como este metodo de clasificación es el mejor entre todos los usados.

Gracias!!

Referencias

Diabetes prediction dataset

A Comprehensive Dataset for Predicting Diabetes with Medical & Demographic Data

kaggle.com