3 - Desarrollo de Modelos I - Optimización de Hiperparámetros

December 29, 2020

1 3. Desarrollo de modelos I. Optimización de Hiperparámetros

Este cuaderno contiene el procedimiento de desarrollo de modelos de clasificación sencillos prestando atención a la selección de Hiperparámetros (en el próximo se continuará con Ensambles). Se parte de un juego de datasets con distintos tipos de remuestreo que ya fue preparado en el cuaderno anterior. El procedimiento está organizado en las siguientes secciones:

1. Carga de dataset con distintos preprocesamientos.
2. Desarrollo y entrenamiento de modelos con estimación bayesiana de hiperparámetros usando librerías hyperopt y optuna.
3. Comparación de resultados y conclusiones.

1.1 3.1 Carga de dataset con distintos preprocesamientos

En el cuaderno anterior se generaron los siguientes archivos CSV: - **creditcard_train.csv**: partición de dataset original para entrenamiento con modificación de columnas (row_id y tiempo). - **creditcard_test.csv**: partición de dataset original para evaluación con modificación de columnas (row_id y tiempo). - **creditcard_downsampled.csv**: dataset balanceado por método de undersampling. - **creditcard_train_oversampled_adasyn.csv**: partición de entrenamiento balanceado por upsampling (ADASYN). - **creditcard_train_oversampled_smote.csv**: partición de entrenamiento balanceado por upsampling (SMOTE). - **creditcard_train_oversampled_blsmote.csv**: partición de entrenamiento balanceado por upsampling (Borderline SMOTE).

En esta sección se los carga y particiona para poder utilizarlos en el entrenamiento de modelos.

[1]:
```
TMP_PATH = "./tmp/" # Path temporal usado para imágenes generadas, archivos␣
 ↪intermedios, etc.
MODELS_PATH = "/models/"
```

[2]:
```
import pandas as pd
import numpy as np
import math as m
import joblib
from collections import OrderedDict
import matplotlib.pyplot as plt
```

[3]:
```
DATASET_PATH = '/data/credit_fraud/'
```

Se utilizará un diccionario para facilitar la selección de un dataset durante la configuración de los entrenamientos .

[4]:
```
train_ds_dict = {
    "downsampled": pd.read_csv(DATASET_PATH+"creditcard_downsampled.csv"),
    "os_adasyn": pd.read_csv(DATASET_PATH+"creditcard_train_oversampled_adasyn.
↪csv"),
    "os_blsmote": pd.
↪read_csv(DATASET_PATH+"creditcard_train_oversampled_blsmote.csv"),
    "os_smote": pd.read_csv(DATASET_PATH+"creditcard_train_oversampled_smote.
↪csv")
}
```

Dataset de evaluación.

[5]:
```
test_df = pd.read_csv(DATASET_PATH+"creditcard_test.csv")
```

Selección de columnas de features.

[6]:
```
non_feature_cols = ['Unnamed: 0','time','row_id','class']
feature_cols = [x for x in test_df.columns if x not in non_feature_cols]
feature_cols
```

[6]:
```
['v1',
 'v2',
 'v3',
 'v4',
 'v5',
 'v6',
 'v7',
 'v8',
 'v9',
 'v10',
 'v11',
 'v12',
 'v13',
 'v14',
 'v15',
 'v16',
 'v17',
 'v18',
 'v19',
 'v20',
 'v21',
 'v22',
 'v23',
 'v24',
 'v25',
```

```
 'v26',
 'v27',
 'v28',
 'amount']
```

Para los datos a los cuáles se ha aplicado undersampling es necesario particionar en train y split. Como el undersampling se hizo eliminando muestras de la clase mayoritaria, en este caso los datos ya están balanceados y se puede aplicar cualquier métrica sobre el test set.

[7]:
```
from sklearn.model_selection import train_test_split

TEST_SIZE = 0.3

X_train_downsampled,X_test_downsampled,y_train_downsampled,y_test_downsampled =␣
 ↪train_test_split(
    train_ds_dict['downsampled'][feature_cols],
    train_ds_dict['downsampled']['class'],
    test_size=TEST_SIZE, random_state=42)
```

[8]:
```
X_train_downsampled.head(3)
```

[8]:
```
            v1         v2         v3        v4         v5        v6  \
398  -0.112195   0.401013  -1.368654  1.325461   1.812514 -1.655252
523  -4.727713   3.044469  -5.598354  5.928191  -2.190770 -1.529323
809 -26.457745  16.497472 -30.177317  8.904157 -17.892600 -1.227904

            v7         v8        v9        v10  …       v20       v21  \
398   1.887604  -0.971989  1.356304   0.874950  … -0.038966  0.164669
523  -4.487422   0.916392 -1.307010  -4.138891  … -0.207759  0.650988
809 -31.197329 -11.438920 -9.462573 -22.187089  …  2.812241 -8.755698

          v22       v23       v24       v25       v26       v27       v28  \
398  1.618395  0.465093 -0.081923 -1.065862 -0.418760  0.439829 -0.040883
523  0.254983  0.628843 -0.238128 -0.671332 -0.033590 -1.331777  0.705698
809  3.460893  0.896538  0.254836 -0.738097 -0.966564 -7.263482 -1.324884

     amount
398   45.00
523   30.39
809    1.00

[3 rows x 29 columns]
```

[9]:
```
X_test_downsampled.head(3)
```

[9]:
```
            v1        v2         v3        v4        v5        v6         v7  \
613 -10.645800  5.918307 -11.671043  8.807369 -7.975501 -3.586806 -13.616797
```

```
451  -2.218541  1.211222  -0.326345  0.763670 -0.741354  1.914052   0.943716
731  -4.198735  0.194121  -3.917586  3.920748 -1.875486 -2.118933  -3.614445

           v8        v9        v10  …       v20       v21       v22  \
613  6.428169 -7.368451 -12.888158  … -0.046170  2.571970  0.206809
451 -5.294108  1.432909   2.441081  … -1.347714  2.981848 -1.551763
731  1.687884 -2.189871  -4.684233  …  1.003350  0.801312 -0.183001

          v23       v24       v25       v26       v27       v28  amount
613 -1.667801  0.558419 -0.027898  0.354254  0.273329 -0.152908    0.00
451  0.922801  0.722661 -1.848255 -0.816578 -0.757258 -1.143818  282.98
731 -0.440387  0.292539 -0.144967 -0.251744  1.249414 -0.131525  238.90

[3 rows x 29 columns]
```

[10]:
```
y_train_downsampled.value_counts(),y_test_downsampled.value_counts()
```

[10]:
```
(1    346
 0    342
 Name: class, dtype: int64,
 0    150
 1    146
 Name: class, dtype: int64)
```

Para los datos con oversampling no es necesario realizar esta partición pues ya se ha hecho previamente (y sólo se han incorporado muestras a la partición de train). No obstante, los datos del test no han sido balanceados, por lo tanto deben balancearse o seleccionar una métrica apropiada.

Ejemplo de selección del dataset remuestrado utilizando la librería hyperopt:

[11]:
```
from hyperopt import hp
resampling_space = hp.choice('resampling_strategy',['undersampling',␣
 ↪'os_smote', 'os_adasyn', 'os_blsmote'])

def choose_dataset(resampling_strategy):
    if resampling_strategy == 'undersampling':
        X_train = X_train_downsampled
        y_train = y_train_downsampled
        X_test = X_test_downsampled
        y_test = y_test_downsampled
    else:
        X_train = train_ds_dict[resampling_strategy][feature_cols]
        y_train = train_ds_dict[resampling_strategy]['class']
        X_test = test_df[feature_cols]
        y_test = test_df['class']
    return X_train, y_train, X_test, y_test

# Ejemplo:
```

```
X_train, y_train, X_test, y_test= choose_dataset('undersampling') #␣
↪args['resampling_strategy']
```

1.2 3.2 Entrenamiento de modelos

En esta sección se procederá a entrenar distintos tipos de modelos de aprendizaje supervisado intentando encontrar la mejor combinación de hiperparámetros para cada uno.

- **Árboles de decisión**.
- **Random Forest**.
- **Regresión Logística**.
- **Support Vector Machine**.
- **Multi-layer Perceptron**.
- **XGBoost**.
- **Nearest Neighbors**.

Nota: Si bien se presentó la forma de poder seleccionar el dataset de entrenamiento entre los 4 posibles (uno de undersampling y tres de oversampling) se omitieron los datasets ampliados porque se extienden demasiado los tiempos de entrenamiento en el HW disponible.

1.2.1 3.2.1 Criterio de evaluación

Para todos los modelos se obtendrán métricas relevantes para la clasificación binaria:

- **Accuracy**: Ratio de observaciones correctas sobre total de observaciones. $\frac{TP+TN}{TP+FP+FN+TN}$. Dado que se ha aplicado undersampling a los datos para balancearlos, esta métrica puede usarse. De mantenerse el dataset imbalanceado esta métrica puede dar una interpretación errónea del desempeño del algoritmo.
- **Precision**: $\frac{TP}{TP+FP}$. Se relaciona con una baja tasa de falsos positivos.
- **Recall**: $\frac{TP}{TP+FN}$. Mide la cantidad de predicciones correctas para cada clase. De todos los casos de fraude, ¿cuántos fueron correctamente identificados?.
- **Curva ROC**: La curva ROC indica qué tan capaz es un modelo de distinguir clases relacionando la tasa de falsos positivos con la tasa de verdaderos positivos.
- **AUC**: el Área bajo la Curva ROC es un indicador de qué tan bueno es un clasificador independientemente del umbral de clasificación elegido.
- **f1-score**: Promedio entre Precision y Recall. $2\frac{RecallxPrecision}{Recall+Precision}$
- **Matriz de Confusión:**: es una forma de visualizar para cada clase TP,TN,FP y FN.

Siendo: - TP (True Positives): casos de Fraude identificados como Fraude. - TN (True Negatives): casos de No fraude identificados como No Fraude. - FP (False Positives): casos de No Fraude identificados como Fraude. - FN (False Negatives): casos de Fraude identificados como No Fraude.

No obstante, se utilizará AUC como la métrica principal y la que utilizará la función objetivo de las librerías de búsqueda bayesiana.

Se provee una función que realiza un reporte con las anteriores métricas para un modelo y un dataset de evaluación.

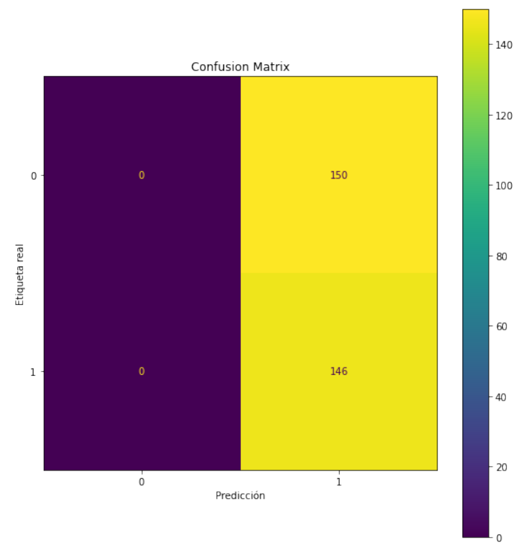
[12]:
```
from sklearn import metrics

def model_evaluation_report(model,y_test,y_pred,y_pred_prob,description):
```

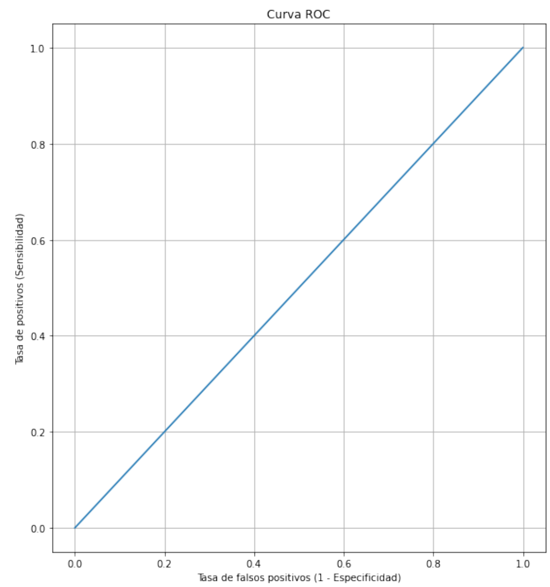
```
    accuracy = metrics.accuracy_score(y_test, y_pred)
    precision = metrics.precision_score(y_test, y_pred,zero_division=False)
    recall = metrics.recall_score(y_test, y_pred)

    #fig, axes = plt.subplots(1, 2, figsize=(8, 4),␣
↪gridspec_kw=dict(width_ratios=[4, 3]))
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob[:,1])
    auc = metrics.auc(fpr, tpr)
    f1_score = metrics.f1_score(y_test, y_pred)

    fig, axes = plt.subplots(1,2,figsize=(20,10))
    metrics.plot_confusion_matrix(model, X_test, y_test,ax=axes[0])  # doctest:␣
↪+SKIP
    axes[0].set_title("Confusion Matrix")
    axes[0].set_xlabel('Predicción')
    axes[0].set_ylabel('Etiqueta real')
    #plt.show()

    #plt.figure()
    axes[1].plot(fpr, tpr)
    axes[1].grid(which='Both')
    axes[1].set_title("Curva ROC")
    axes[1].set_xlabel('Tasa de falsos positivos (1 - Especificidad)')
    axes[1].set_ylabel('Tasa de positivos (Sensibilidad)')
    plt.show()

    print("AUC:", auc )
    print("Accuracy:", accuracy )
    print("Precision:", precision )
    print("Recall:", recall)
    print("f1-score: ", f1_score)

    model_summary={
        "accuracy": accuracy,
        "precision": precision,
        "recall": recall,
        "auc": auc,
        "f1-score": f1_score,
        "description": description
    }
    return model_summary
```

Ejemplo para el clasificador de baseline (Dummy).

[13]:
```
from sklearn.dummy import DummyClassifier

model = DummyClassifier(strategy='most_frequent')
```

```
model = model.fit(X_train,y_train)
y_pred = model.predict(X_test)
y_pred_prob = model.predict_proba(X_test)

model_name = "base"
description = "Modelo base"
base_model_metrics =␣
↪model_evaluation_report(model,y_test,y_pred,y_pred_prob,description)
```






```
AUC: 0.5
Accuracy: 0.49324324324324326
Precision: 0.49324324324324326
Recall: 1.0
f1-score:  0.6606334841628959
```

1.2.2 3.2.2 Búsqueda de hiperparámetros

Para el entrenamiento de modelos se utilizará Búsqueda Bayesiana con el algoritmo TPE (Tree-structured Parzen Estimator).

Este algoritmo utiliza un modelo secuencial (por sus siglas en inglés: Sequential Model-Based Optimization). Estos métodos. Estos métodos utilizan la historia pasada de los hiperparámetros y la métrica otenida para elegir un nuevo conjunto de hiperparámetros a ensayar que maximice la probabilidad de mejorar esa métrica.

El algoritmo TPE modela $P(x|y)$ y $P(y)$, donde x representa un conjunto de valores de hiperparámetros e y su puntaje.

Se implementa la clase HyperoptTrainer como boilerplate para el código de entrenamiento, selección de mejor modelo y preparación de dataframe de resultados con la librería Hyperopt.

[14]:
```python
from hyperopt import tpe, fmin, hp, Trials,space_eval
from hyperopt.pyll import scope

from sklearn import metrics
from functools import partial
from sklearn.metrics import plot_confusion_matrix

import json

class HyperoptTrainer:
    def __init__(self, model_class, model_hp_space, model_name,␣
↪model_description, instance_model_callback=None):
        self.model_class = model_class
        self.model_hp_space = model_hp_space
        self.model_name = model_name
        self.model_description = model_description
        if instance_model_callback:
            self.instance_model_callback = instance_model_callback
        else:
            self.instance_model_callback = lambda args: eval(self.
↪model_class)(**args)
        pass

    def objective_func(self,args):
        """ Instancia un modelo con los parámetros sugeridos y
            devuelve el puntaje negativo al evaluarlo sobre el test set.
        """
        # 1. Instanciar el modelo
        model = self.instance_model_callback(args)

        # 2. Entrenar
        model = model.fit(self.X_train,self.y_train)

        # 3. Evaluar en test set
        y_pred = model.predict(self.X_test)
        y_pred_prob = model.predict_proba(self.X_test)
        fpr, tpr, thresholds = metrics.roc_curve(self.y_test, y_pred_prob[:,1])
        auc_score = metrics.auc(fpr, tpr)

        # 4. Guardar logs
        self.logs['val_score'].append(auc_score)

        return -auc_score

    def fit(self, X_train, y_train, X_test, y_test,max_evals=10):
        self.X_train = X_train
        self.y_train = y_train
```

```
        self.X_test = X_test
        self.y_test = y_test
        trials = Trials()
        self.logs = {
            'args':list(),
            'val_score': list()
        }
        best = fmin(
            self.objective_func,
            self.model_hp_space,
            algo=tpe.suggest,
            max_evals=max_evals,
            verbose=True,
            show_progressbar=True,
            trials = trials
        )

        # Mejor set de parámetros
        self.best = space_eval(self.model_hp_space, best)

        # Tabla de resultados (dataframe)
        self.trials_df = pd.DataFrame([pd.Series(t["misc"]["vals"]).
↪apply(lambda x: x[0] if x else np.nan) for t in trials])
        self.trials_df["loss"] = [t["result"]["loss"] for t in trials]
        self.trials_df["trial_number"] = self.trials_df.index

        # Entrenar el mejor modelo
        self.best_model = self.instance_model_callback(self.best)
        self.best_model = self.best_model.fit(self.X_train,self.y_train)

        # Almacenarlo
        model_filename = MODELS_PATH+ self.model_name+".pkl"
        model_params_filename = MODELS_PATH+ self.model_name+".json"
        joblib.dump(self.best_model, model_filename)
        print("Almacenado modelo: ",model_filename)
        with open(model_params_filename, 'w') as fp:
            fp.write(json.dumps(self.best))
        print("Almacenados parámetros de modelo: ",model_params_filename)
        return

    def plot_hp_search(self,additional_fields):
        fig, axes = plt.
↪subplots(1+len(additional_fields),1,sharex=True,figsize=(22,12))
        axes[0].grid(which="Both")
        axes[0].plot(self.logs['val_score'])
        axes[0].set_title('Val score (AUC)')
```

```
        for index, field in enumerate(additional_fields):
            axes[1+index].plot(self.trials_df[field],c=np.random.rand(3,))
            axes[1+index].set_title(field)
            axes[1+index].grid(which="Both")

        plt.xlabel("Iteración")
        plt.tight_layout()

    def get_best_parameters(self):
        return self.best

    def get_best_model(self):
        return self.best_model
```

1.2.3 3.3 Entrenamiento de modelos

[15]:
```
X_train, y_train, X_test, y_test= choose_dataset('undersampling')
```

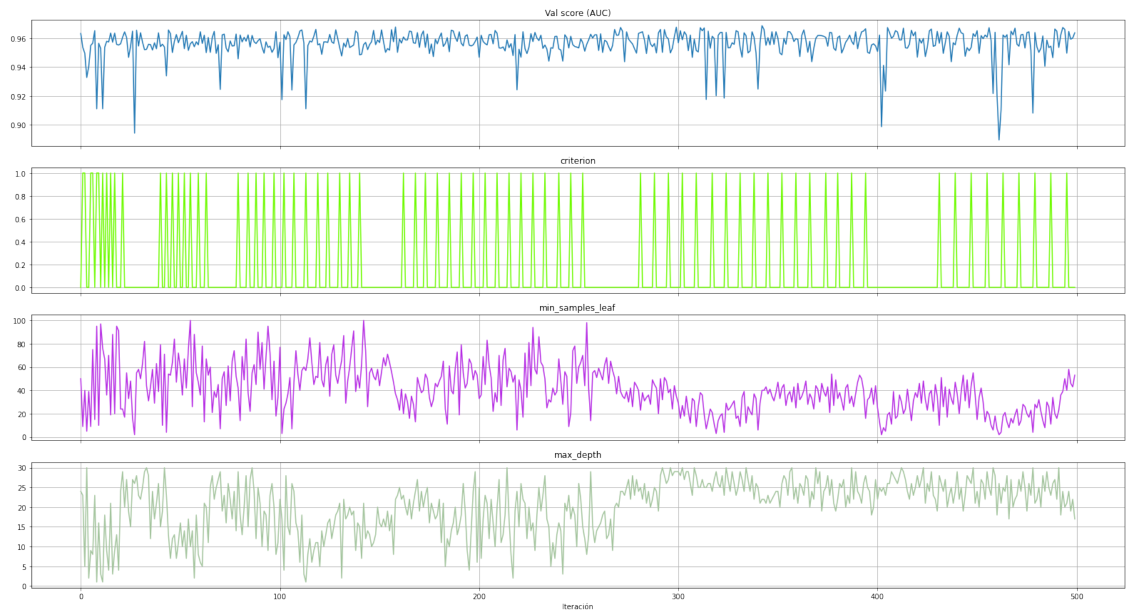
[16]:
```
# Para futura tabla comparativa
model_metrics_list = {
    "base": base_model_metrics
}
```

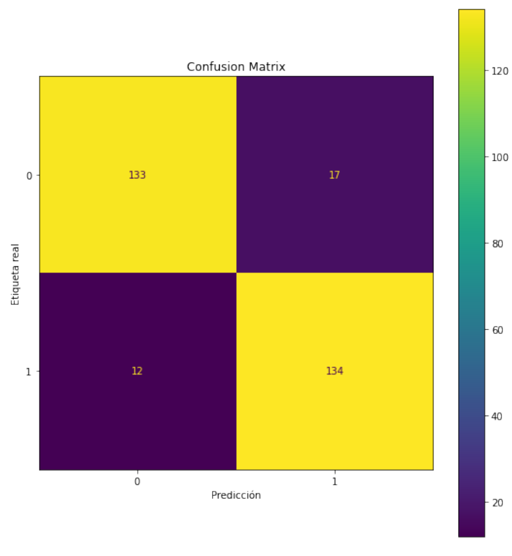
3.3.1 Árbol de Decisión (con Hyperopt) Se comenzará entrenando modelos de árboles de decisión ensayando algunos juegos de parámetros de criterio y profundidad.

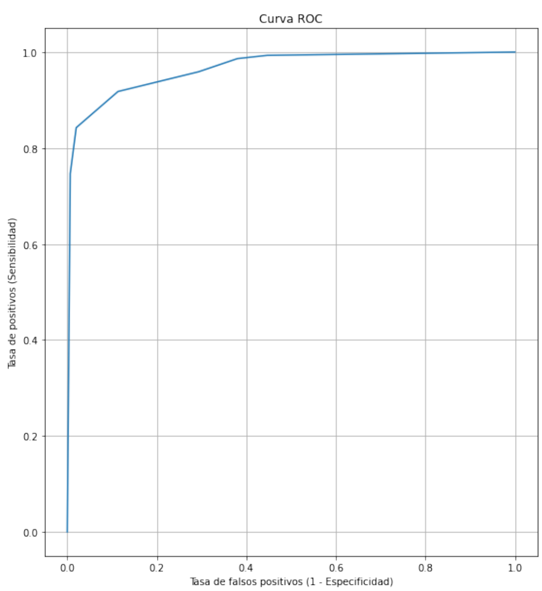
[17]:
```
from sklearn.tree import DecisionTreeClassifier

model_class = 'DecisionTreeClassifier'
model_name = "DecisionTree"
model_description = "Decision Tree"

model_hp_space = OrderedDict([
        ('criterion', hp.choice('criterion',['gini', 'entropy']) ),
        ('min_samples_leaf', hp.uniformint('min_samples_leaf', 2, 100, q=1)),
        ('max_depth', scope.int(hp.quniform('max_depth', 1, 30, q=1)))
])
hp_search_additional_fields = ["criterion","min_samples_leaf","max_depth"]

hpo_trainer = HyperoptTrainer(model_class, model_hp_space, model_name,␣
 ↪model_description)
hpo_trainer.fit(X_train,y_train,X_test,y_test,max_evals=500)
hpo_trainer.plot_hp_search(hp_search_additional_fields)
best_model = hpo_trainer.get_best_model()
y_pred = best_model.predict(X_test)
y_pred_prob = best_model.predict_proba(X_test)
```

```
model_metrics_list[model_name]=model_evaluation_report(best_model,y_test,y_pred,y_pred_prob,mo
```

```
100%|      | 500/500 [00:25<00:00, 19.63trial/s, best loss:
-0.9687899543378995]
Almacenado modelo:  /models/DecisionTree.pkl
Almacenados parámetros de modelo:  /models/DecisionTree.json
```







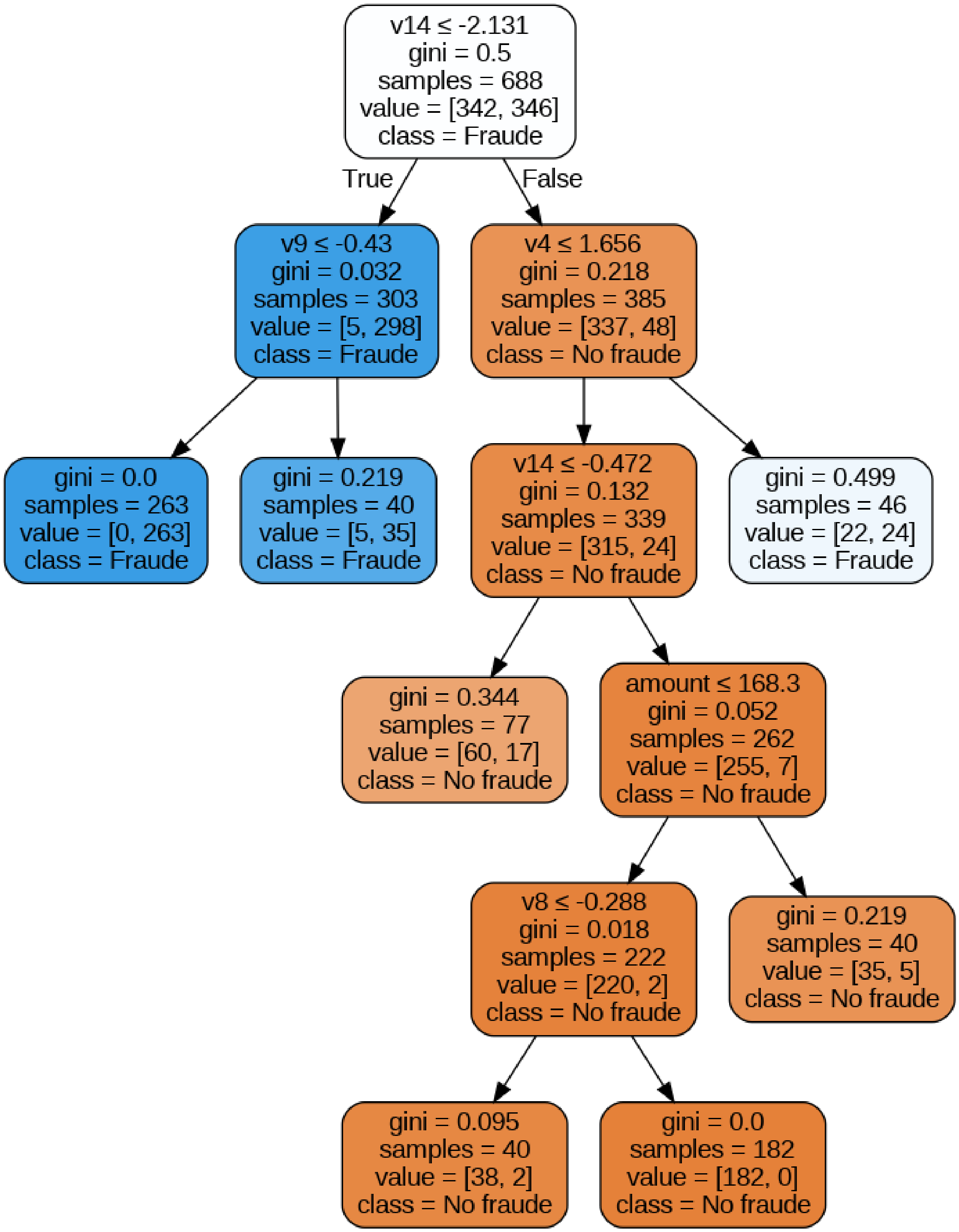

```
AUC: 0.9658447488584474
```

```
Accuracy: 0.9020270270270270
Precision: 0.8874172185430463
Recall: 0.9178082191780822
f1-score:  0.9023569023569024
```

Se observa que el resultado obtenido es bastante bueno. No es totalmente evidente a simple vista, pero pareciera ser que los valores muy bajos de *min_samples_leaf max_depth* penalizan el desempeño.

[18]:
```
from sklearn.tree import DecisionTreeClassifier
from six import StringIO
from IPython.display import display, Image,HTML
from sklearn.tree import export_graphviz
import pydotplus

def plot_and_save_tree_diagrams(model, name):
    dot_data = StringIO()
    export_graphviz( model, out_file=dot_data,  filled=True, rounded=True,␣
 ↪special_characters=True,
                    feature_names = feature_cols,
                    class_names=['No fraude','Fraude'])
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    graph.write_png(TMP_PATH+name+'_tree.png')
    display(Image(graph.create_png()))
```

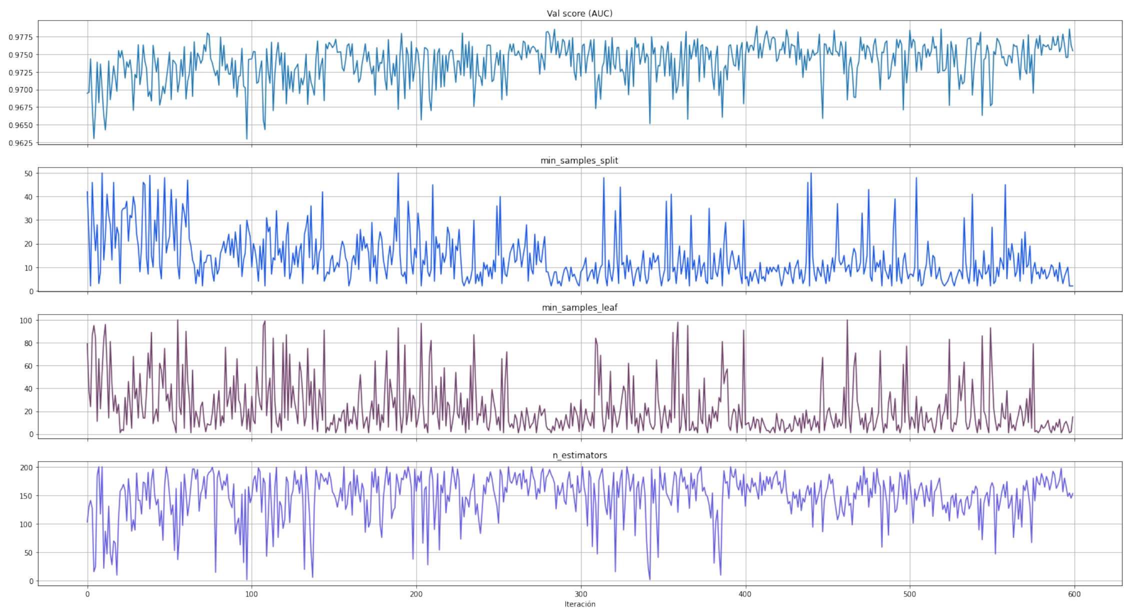
[19]:
```
plot_and_save_tree_diagrams(best_model,model_name)
```

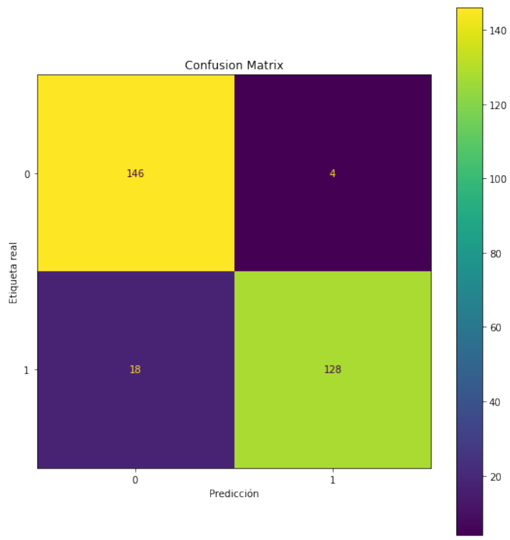


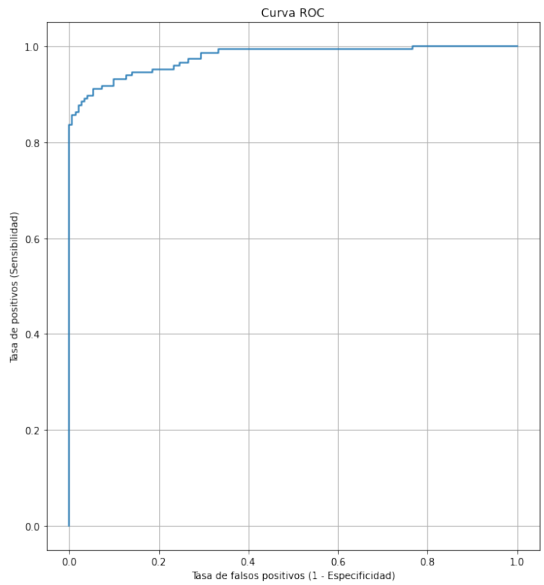
3.3.2 Random Forest (con Hyperopt) Uno de los inconvenientes que pueden tener los árboles de decisión es que memoricen las soluciones en lugar de generalizar el aprendizaje. Para mejorar este aspecto, se intentará mejorar el resultado anterior utilizando Random Forest, que hace uso de la técnica de bagging utilizando múltiples árboles como votadores del resultado final. Al igual que en el caso anterior, se hará lo optimización con Hyperopt.

[20]:
```
from sklearn.ensemble import RandomForestClassifier

model_class = 'RandomForestClassifier'
model_name = "RandomForest"
model_description = "RandomForestClassifier"
model_hp_space = OrderedDict([
        ('min_samples_split', hp.uniformint('min_samples_split', 2, 50, q=1)),
        ('min_samples_leaf', hp.uniformint('min_samples_leaf', 1, 100, q=1)),
        ('n_estimators', hp.uniformint('n_estimators', 1, 200, q=1))
])
hp_search_additional_fields =␣
 ↪["min_samples_split","min_samples_leaf","n_estimators"]

hpo_trainer = HyperoptTrainer(model_class, model_hp_space, model_name,␣
 ↪model_description)
hpo_trainer.fit(X_train,y_train,X_test,y_test,max_evals=600)
hpo_trainer.plot_hp_search(hp_search_additional_fields)
best_model = hpo_trainer.get_best_model()
y_pred = best_model.predict(X_test)
y_pred_prob = best_model.predict_proba(X_test)
model_metrics_list[model_name]=model_evaluation_report(best_model,y_test,y_pred,y_pred_prob,mo
```

```
100%|      | 600/600 [05:08<00:00,  1.95trial/s, best loss:
-0.9789954337899544]
Almacenado modelo:  /models/RandomForest.pkl
Almacenados parámetros de modelo:  /models/RandomForest.json
```








```
AUC: 0.9764383561643836
Accuracy: 0.9256756756756757
Precision: 0.9696969696969697
Recall: 0.8767123287671232
f1-score:  0.920863309352518
```

3.3.3 Regresión Logística (con Optuna) Para los parámetros que se quieren ensayar con regresión logística surgió una dificultad en el momento de expresar condicionales anidados -por ejemplo, para los distintos tipos de regularización (*penalty*) que dependiendo del caso requieren parámetros adicionales como *l1_ratio*-. Hyperopt define el espacio de búsqueda mediante diccionarios y si bien es posible hacer anidanimiento (de hecho hyperas lo implementa con directivas especiales para bloques condicionales), se decidió utilizar la librería optuna que facilita la definición de un espacio de búsqueda de manera dinámica. Esto se explica en Optuna: A Next-generation Hyperparameter Optimization Framework.

La API de Optuna cuenta con métodos para la visualización de resultados y devuelve los registros de los intentos como un dataframe, así que en este caso no se definirá una clase como se hizo anteriormente. Como se verá en los siguientes ejemplos, la mayor parte del código es la definición de la función objetivo.

[21]:
```
import optuna
from optuna.samplers import TPESampler
optuna.logging.set_verbosity(optuna.logging.INFO)
```

[22]:
```
from sklearn.linear_model import LogisticRegression
```

[23]:
```
def keep_best_model(study,trial):
    if study.best_trial.number == trial.number:
```

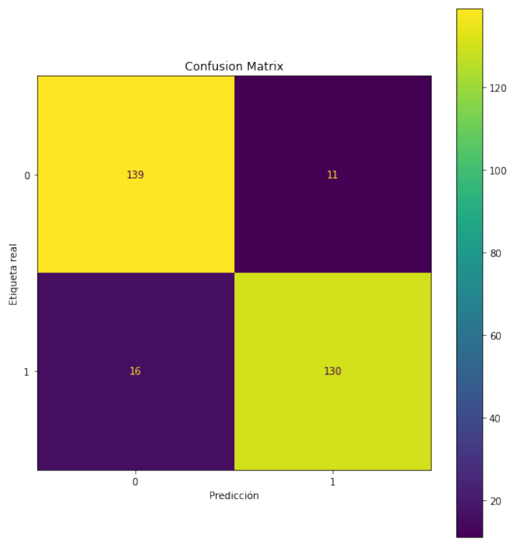
```
        study.set_user_attr(key="best", value=trial.user_attrs["best"])
```

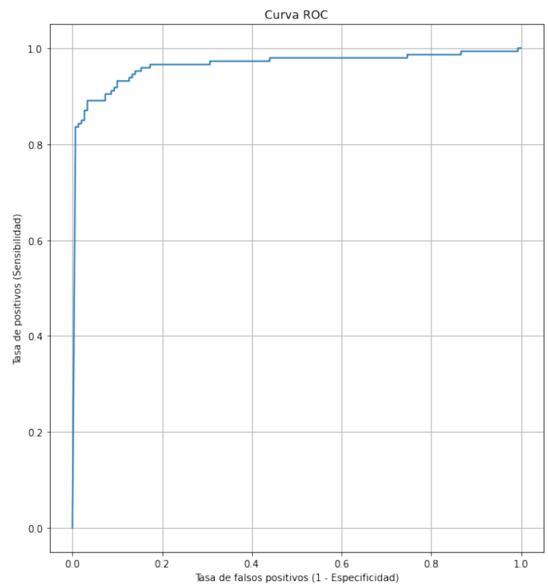
[24]:
```
def save_model(study,model_name):
    model_filename = MODELS_PATH+ model_name+".pkl"
    model_params_filename = MODELS_PATH+ model_name+".json"
    joblib.dump(study.user_attrs["best"], model_filename)
    print("Almacenado modelo: ",model_filename)
    with open(model_params_filename, 'w') as fp:
        fp.write(json.dumps(study.best_trial.params))
    print("Almacenados parámetros de modelo: ",model_params_filename)
```

[ ]:
```
model_name = "LogisticRegression"

def objective_func(trial):
    solver = trial.
 ↪suggest_categorical("solver",['sag','saga','lbfgs','newton-cg'])
    max_iter = trial.suggest_int('n_layers', 500, 2000)

    # 1. Instanciar el modelo
    if solver == 'sag':
        penalty = trial.suggest_categorical("penalty1",['l2','none'])
    elif solver == 'saga':
        penalty = trial.
 ↪suggest_categorical("penalty2",['l1','l2','elasticnet','none'])
    elif solver == 'lbfgs':
        penalty = trial.suggest_categorical("penalty3",['l2','none'])
    elif solver == 'newton-cg':
        penalty = trial.suggest_categorical("penalty4",['l2','none'])

    if penalty in ['elasticnet']:
        l1_ratio= trial.suggest_uniform("l1_ratio", 0, 1)
        model =␣
 ↪LogisticRegression(solver=solver,penalty=penalty,max_iter=max_iter,l1_ratio=l1_ratio)
    else:
        model =␣
 ↪LogisticRegression(solver=solver,penalty=penalty,max_iter=max_iter)

    # 2. Entrenar
    model = model.fit(X_train,y_train)

    # 3. Evaluar en test set
    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob[:,1])
    auc_score = metrics.auc(fpr, tpr)

    #trial.report()
```

```
    trial.set_user_attr(key="best", value=model)
    return auc_score

study = optuna.create_study(direction='maximize',sampler=TPESampler())
study.optimize(objective_func, n_trials=100,callbacks=[keep_best_model])
```

[26]:
```
best_model=study.user_attrs["best"]
y_pred = best_model.predict(X_test)
y_pred_prob = best_model.predict_proba(X_test)
model_metrics_list[model_name]=model_evaluation_report(best_model,y_test,y_pred,y_pred_prob,mo
save_model(study,model_name)
```






```
AUC: 0.9640867579908676
Accuracy: 0.9087837837837838
Precision: 0.9219858156028369
Recall: 0.8904109589041096
f1-score:  0.9059233449477352
Almacenado modelo:  /models/LogisticRegression.pkl
Almacenados parámetros de modelo:  /models/LogisticRegression.json
```

[27]:
```
trials_df = study.trials_dataframe(attrs=('number', 'value', 'params', 'state'))
trials_df.sort_values('value',ascending=False).head(5)
```

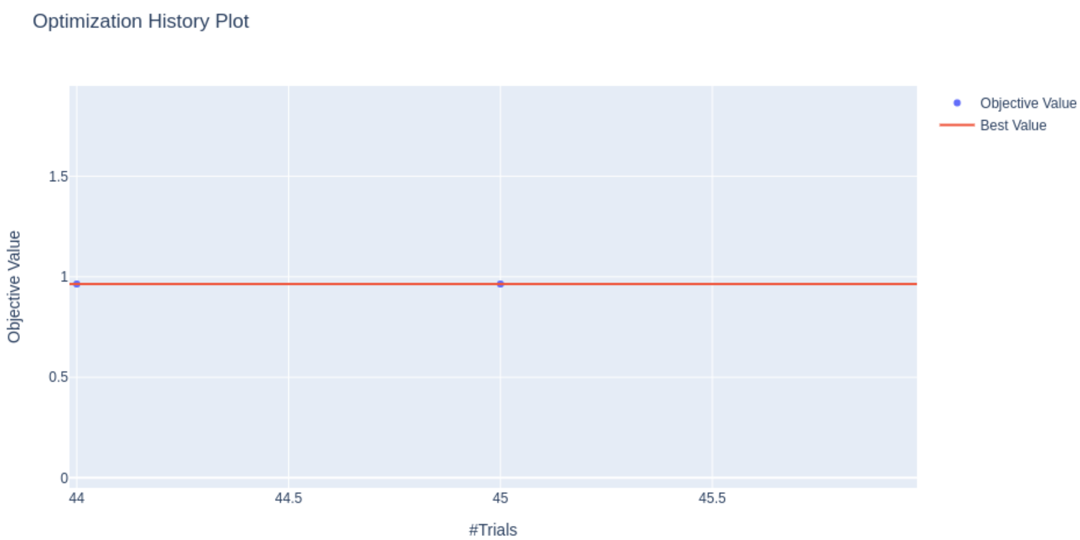
[27]:
```
    number     value  params_l1_ratio  params_n_layers params_penalty1  \
0        0  0.964087              NaN             1659             NaN
73      73  0.964087              NaN             1648             NaN
71      71  0.964087              NaN             1628             NaN
70      70  0.964087              NaN             1198             NaN
```

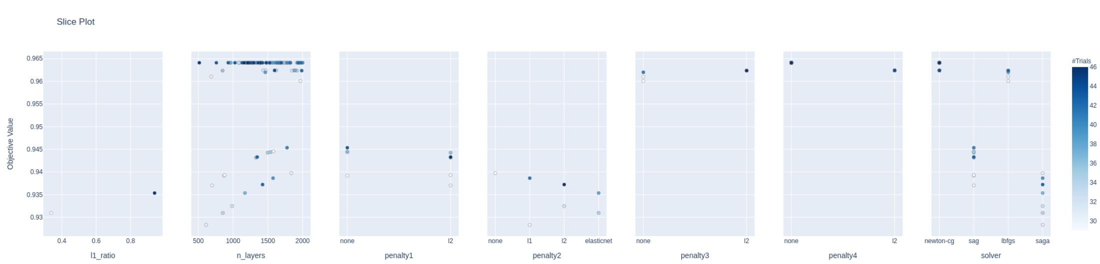
69	69	0.964087	NaN	1260	NaN

	params_penalty2	params_penalty3	params_penalty4	params_solver	state
0	NaN	NaN	none	newton-cg	COMPLETE
73	NaN	NaN	none	newton-cg	COMPLETE
71	NaN	NaN	none	newton-cg	COMPLETE
70	NaN	NaN	none	newton-cg	COMPLETE
69	NaN	NaN	none	newton-cg	COMPLETE

[28]:
```
optuna.visualization.plot_optimization_history(study)
```

[29]:
```
optuna.visualization.plot_slice(study)
```



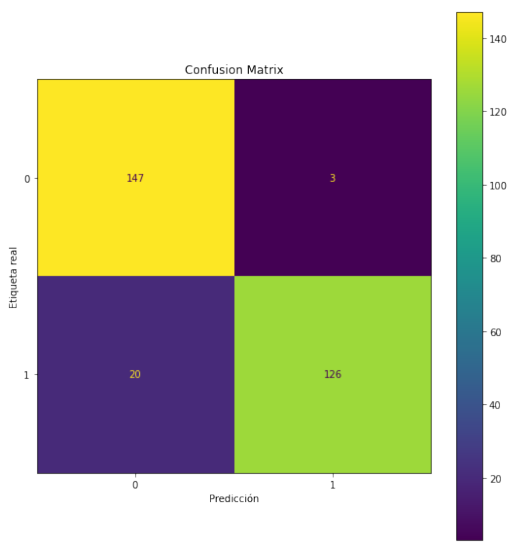


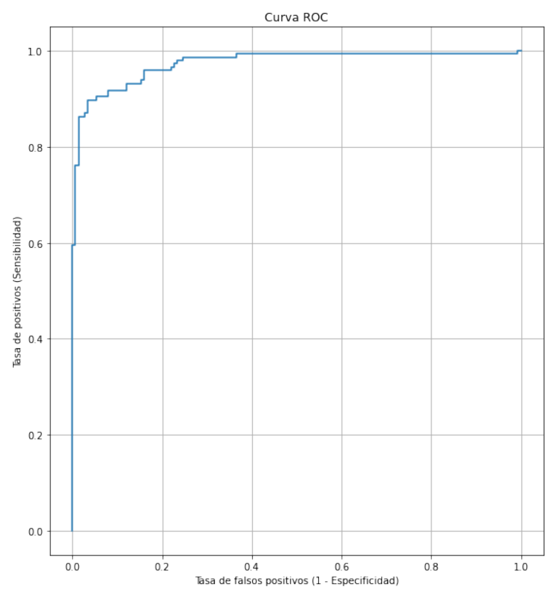
3.3.4 Support Vector Machine (con Optuna)

[ ]:
```
from sklearn import svm

model_name = "SVM"

def objective_func(trial):
    # 1. Instanciar el modelo
    c = trial.suggest_loguniform('C', 1e-10, 1e10)
```

```
    kernel = trial.suggest_categorical("kernel",['linear', 'poly', 'rbf',␣
↪'sigmoid'])
    gamma = trial.suggest_categorical("gamma",['scale', 'auto'])
    model = svm.SVC(kernel=kernel,C=c, gamma=gamma, max_iter=1000,␣
↪probability=True)

    # 2. Entrenar
    model = model.fit(X_train,y_train)

    # 3. Evaluar en test set
    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob[:,1])
    auc_score = metrics.auc(fpr, tpr)

    #trial.report()
    trial.set_user_attr(key="best", value=model)
    return auc_score

study = optuna.create_study(direction='maximize',sampler=TPESampler())
study.optimize(objective_func, n_trials=100,callbacks=[keep_best_model])
```

[31]:
```
best_model=study.user_attrs["best"]
y_pred = best_model.predict(X_test)
y_pred_prob = best_model.predict_proba(X_test)
model_metrics_list[model_name]=model_evaluation_report(best_model,y_test,y_pred,y_pred_prob,mo
save_model(study,model_name)
```





```
AUC: 0.9733333333333333
Accuracy: 0.9222972972972973
Precision: 0.9767441860465116
Recall: 0.863013698630137
f1-score:  0.9163636363636364
Almacenado modelo:  /models/SVM.pkl
Almacenados parámetros de modelo:  /models/SVM.json
```

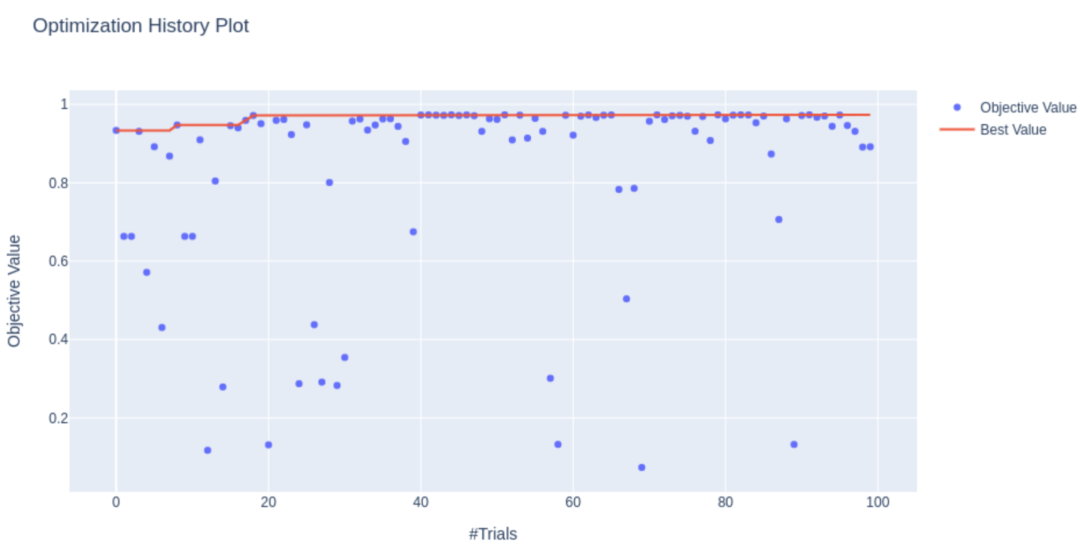
[32]:
```
trials_df = study.trials_dataframe(attrs=('number', 'value', 'params', 'state'))
trials_df.sort_values('value',ascending=False).head(5)
```

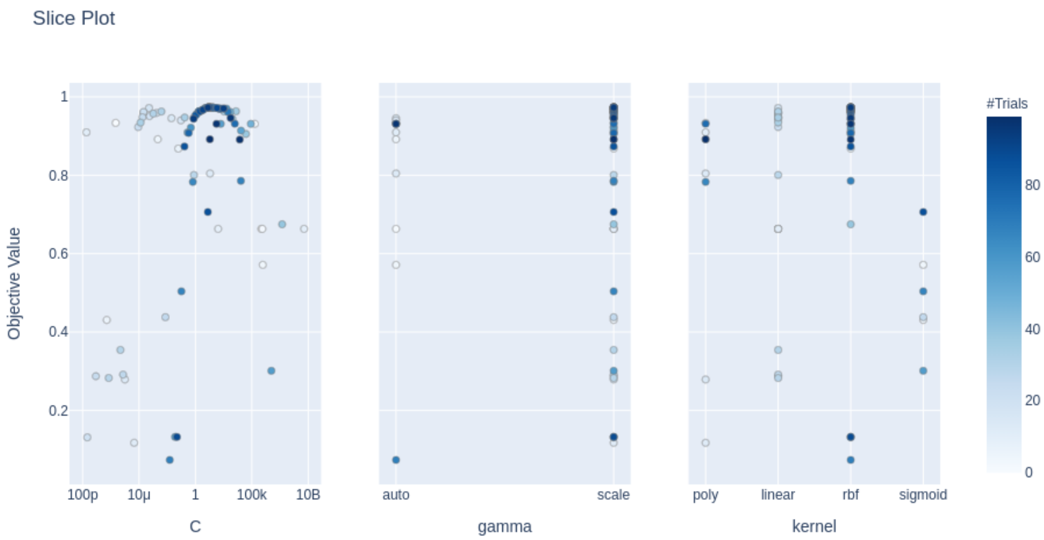
[32]:

	number	value	params_C	params_gamma	params_kernel	state
44	44	0.973333	17.483657	scale	rbf	COMPLETE
51	51	0.973333	17.439836	scale	rbf	COMPLETE
91	91	0.973333	17.537885	scale	rbf	COMPLETE
82	82	0.973288	17.213681	scale	rbf	COMPLETE
79	79	0.973288	17.808005	scale	rbf	COMPLETE

[33]:
```
optuna.visualization.plot_optimization_history(study)
```

[34]:
```
optuna.visualization.plot_slice(study)
```



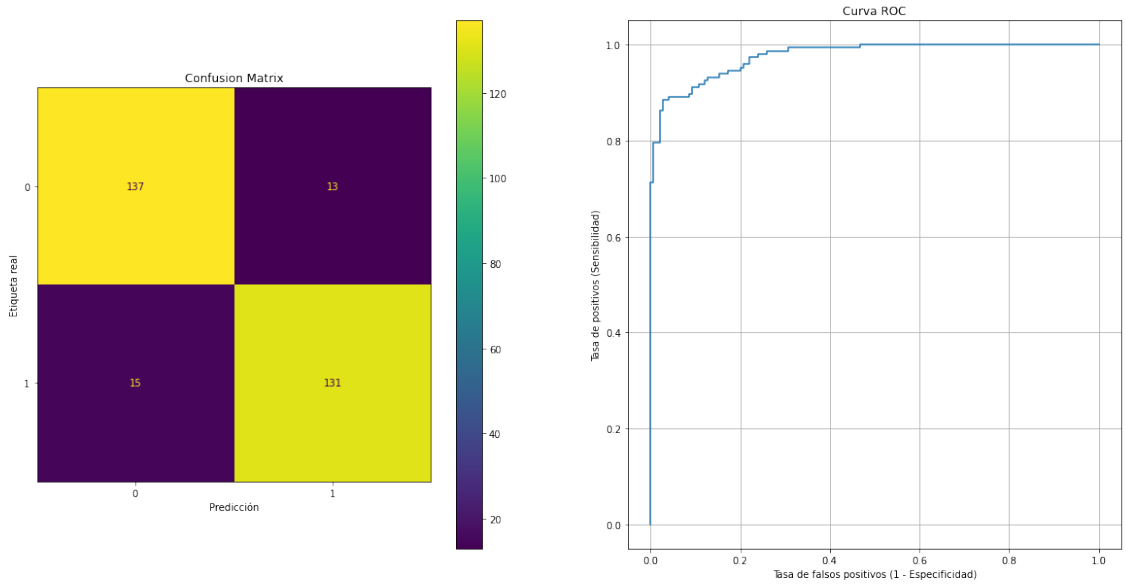


3.3.5 Multi Layer Perceptron (con Optuna)

[ ]:
```
from sklearn.neural_network import MLPClassifier

model_name = "MLP"

def objective_func(trial):
    # 1. Instanciar el modelo
    solver =  trial.suggest_categorical("solver",['sgd', 'lbfgs'])
    n_layers = trial.suggest_int('n_layers', 1, 5)
    learning_rate = trial.suggest_loguniform('learning_rate', 1e-5, 1e-3)
    hidden_layer_sizes = []
    for i in range(n_layers):
        layer_size = trial.suggest_int('n_units_{}'.format(i),4, 64)
        hidden_layer_sizes.append(layer_size)

    model = MLPClassifier( solver=solver, alpha=learning_rate,␣
 ↪hidden_layer_sizes=hidden_layer_sizes,max_iter=2000)

    # 2. Entrenar
    model = model.fit(X_train,y_train)

    # 3. Evaluar en test set
    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob[:,1])
    auc_score = metrics.auc(fpr, tpr)

    #trial.report()
    trial.set_user_attr(key="best", value=model)
```

```
    return auc_score

study = optuna.create_study(direction='maximize',sampler=TPESampler())
study.optimize(objective_func, n_trials=50,callbacks=[keep_best_model])
```

[36]:
```
best_model=study.user_attrs["best"]
y_pred = best_model.predict(X_test)
y_pred_prob = best_model.predict_proba(X_test)
model_metrics_list[model_name]=model_evaluation_report(best_model,y_test,y_pred,y_pred_prob,mo
save_model(study,model_name)
```




```
AUC: 0.9762100456621005
Accuracy: 0.9054054054054054
Precision: 0.9097222222222222
Recall: 0.8972602739726028
f1-score:  0.9034482758620691
Almacenado modelo:  /models/MLP.pkl
Almacenados parámetros de modelo:  /models/MLP.json
```

[37]:
```
trials_df = study.trials_dataframe(attrs=('number', 'value', 'params', 'state'))
trials_df.sort_values('value',ascending=False).head(5)
```

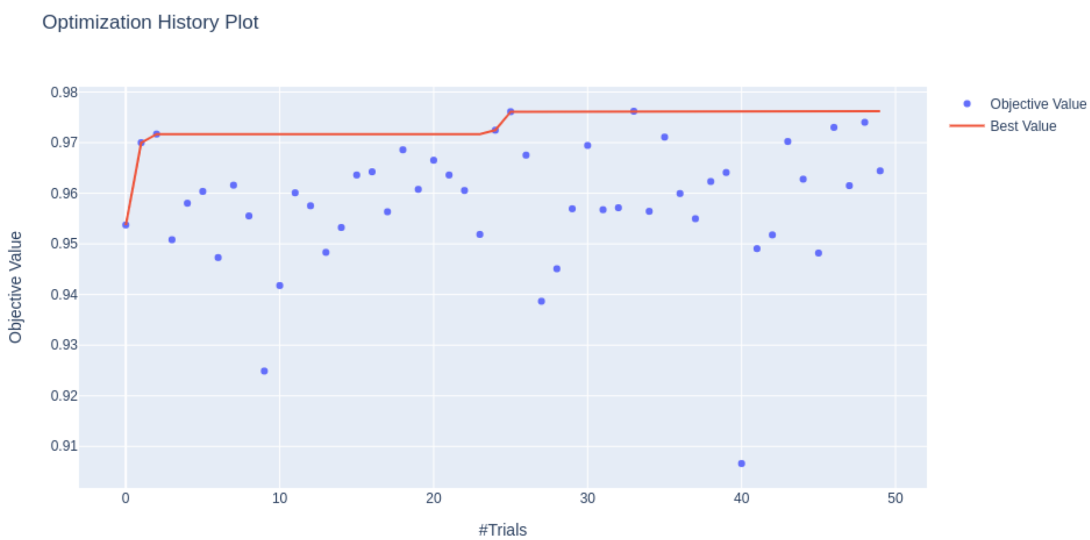
[37]:

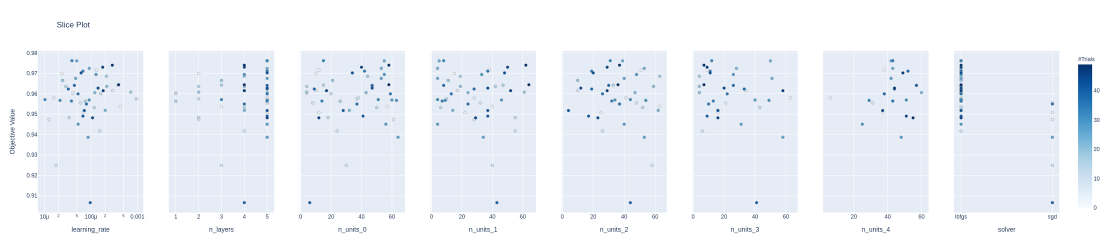
	number	value	params_learning_rate	params_n_layers	params_n_units_0 \
33	33	0.976210	0.000039	5	15
25	25	0.976119	0.000050	5	55
48	48	0.974018	0.000287	4	58
46	46	0.973014	0.000178	4	40
24	24	0.972466	0.000092	5	53

```
    params_n_units_1  params_n_units_2  params_n_units_3  params_n_units_4  \
33               8.0              31.0              12.0              43.0
25               5.0              39.0              50.0              42.0
48              62.0              37.0               7.0               NaN
46              50.0              29.0               9.0               NaN
24               4.0              53.0              28.0              43.0

   params_solver     state
33         lbfgs  COMPLETE
25         lbfgs  COMPLETE
48         lbfgs  COMPLETE
46         lbfgs  COMPLETE
24         lbfgs  COMPLETE
```

[38]:
```
optuna.visualization.plot_optimization_history(study)
```

[39]:
```
optuna.visualization.plot_slice(study)
```



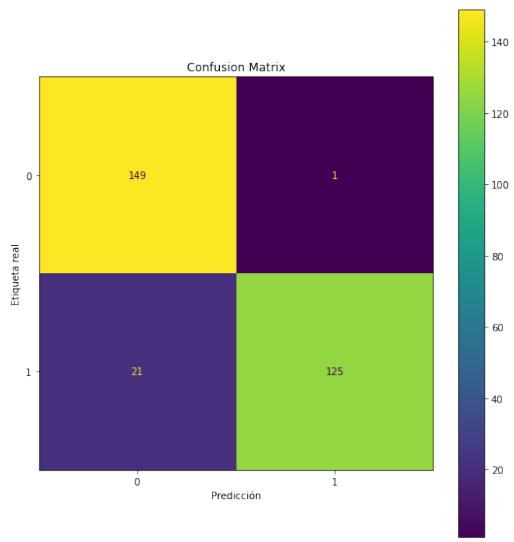


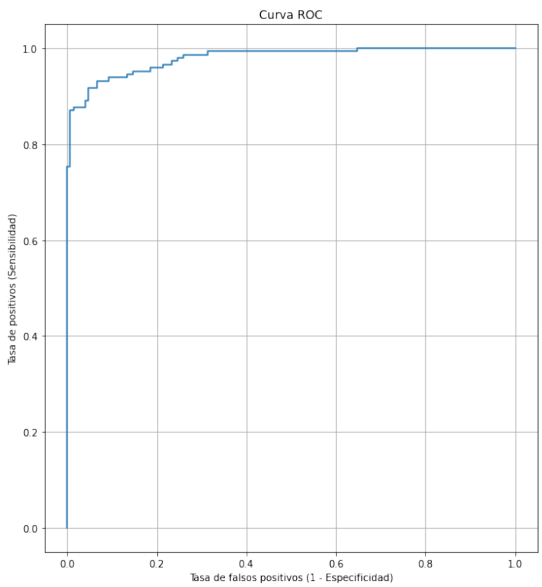
3.3.6 XGBoost

[ ]:
```
from xgboost import XGBClassifier
```

```
model_name = "xgboost"

def objective_func(trial):

    params = {
        "boosting": trial.suggest_categorical('boosting', ['gbtree',␣
↪'gblinear']),
        "tree_method": trial.suggest_categorical('tree_method',␣
↪['exact','approx','hist'] ),
        "max_depth": trial.suggest_int('max_depth', 2, 25),
        "reg_alpha": trial.suggest_int('reg_alpha', 0, 5),
        "reg_lambda": trial.suggest_int('reg_lambda', 0, 5),
        "min_child_weight": trial.suggest_int('min_child_weight', 0, 5),
        "gamma": trial.suggest_int('gamma', 0, 5),
        "learning_rate": trial.suggest_loguniform('learning_rate',0.005,0.5),
        "eval_metric": trial.suggest_categorical('eval_metric',  ['rmse']),
        "objective": trial.suggest_categorical('objective', ['reg:linear', 'reg:
↪gamma', 'reg:tweedie']),
        "colsample_bytree": trial.suggest_discrete_uniform('colsample_bytree',␣
↪0.1, 1, 0.01),
        "colsample_bynode": trial.suggest_discrete_uniform('colsample_bynode',␣
↪0.1, 1, 0.01),
        "colsample_bylevel": trial.
↪suggest_discrete_uniform('colsample_bylevel', 0.1, 1, 0.01),
        "subsample": trial.suggest_discrete_uniform('subsample', 0.5, 1, 0.05),
        "nthread": -1
    }

    # 1. Instanciar
    model = XGBClassifier(**params)

    # 2. Entrenar
    model = model.fit(X_train, y_train.values)

    # 3. Evaluar en test set
    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob[:,1])
    auc_score = metrics.auc(fpr, tpr)

    #trial.report()
    trial.set_user_attr(key="best", value=model)
    return auc_score

study = optuna.create_study(direction='maximize',sampler=TPESampler())
study.optimize(objective_func, n_trials=100,callbacks=[keep_best_model])
```

[46]:
```
best_model=study.user_attrs["best"]
y_pred = best_model.predict(X_test)
y_pred_prob = best_model.predict_proba(X_test)
model_metrics_list[model_name]=model_evaluation_report(best_model,y_test,y_pred,y_pred_prob,mo
save_model(study,model_name)
```






```
AUC: 0.9794520547945206
Accuracy: 0.9256756756756757
Precision: 0.9920634920634921
Recall: 0.8561643835616438
f1-score:  0.9191176470588235
Almacenado modelo:  /models/xgboost.pkl
Almacenados parámetros de modelo:  /models/xgboost.json
```

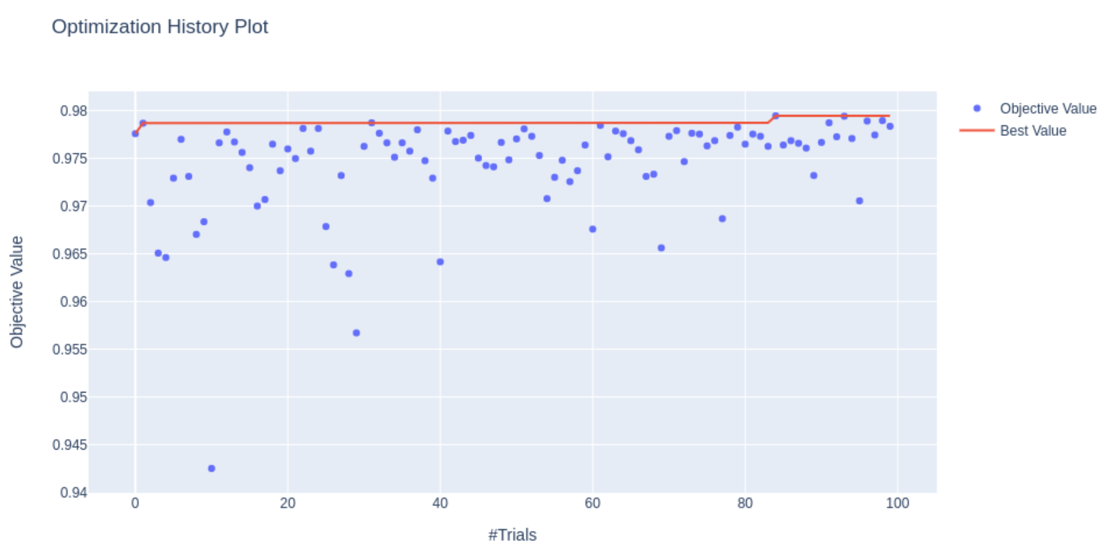
[47]:
```
trials_df = study.trials_dataframe(attrs=('number', 'value', 'params', 'state'))
trials_df.sort_values('value',ascending=False).head(5)
```

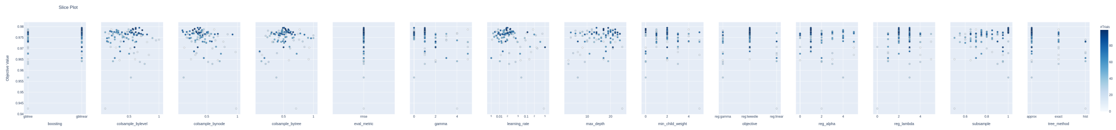
[47]:
```
    number     value params_boosting  params_colsample_bylevel  \
84      84  0.979452        gblinear                      0.67
93      93  0.979406        gblinear                      0.61
98      98  0.978950        gblinear                      0.56
96      96  0.978904        gblinear                      0.61
91      91  0.978721        gblinear                      0.57

    params_colsample_bynode  params_colsample_bytree params_eval_metric  \
84                     0.35                     0.51               rmse
93                     0.39                     0.49               rmse
98                     0.30                     0.53               rmse
```

```
96                     0.30                      0.53                   rmse
91                     0.37                      0.58                   rmse

    params_gamma  params_learning_rate  params_max_depth  \
84             0              0.020262                14
93             1              0.018596                19
98             1              0.025765                17
96             1              0.018101                17
91             1              0.025123                22

    params_min_child_weight params_objective  params_reg_alpha  \
84                        2      reg:tweedie                 1
93                        2      reg:tweedie                 1
98                        0      reg:tweedie                 1
96                        0      reg:tweedie                 1
91                        2      reg:tweedie                 1

    params_reg_lambda  params_subsample params_tree_method     state
84                  3               1.0             approx  COMPLETE
93                  2               1.0             approx  COMPLETE
98                  3               1.0             approx  COMPLETE
96                  2               1.0             approx  COMPLETE
91                  2               1.0             approx  COMPLETE
```

[48]:
```
optuna.visualization.plot_optimization_history(study)
```

[49]:
```
optuna.visualization.plot_slice(study)
```

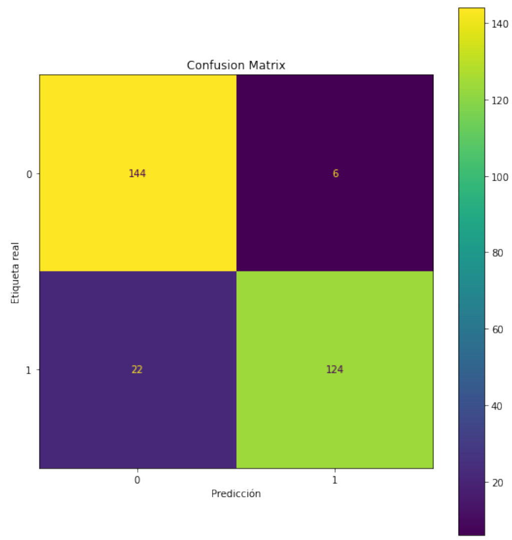


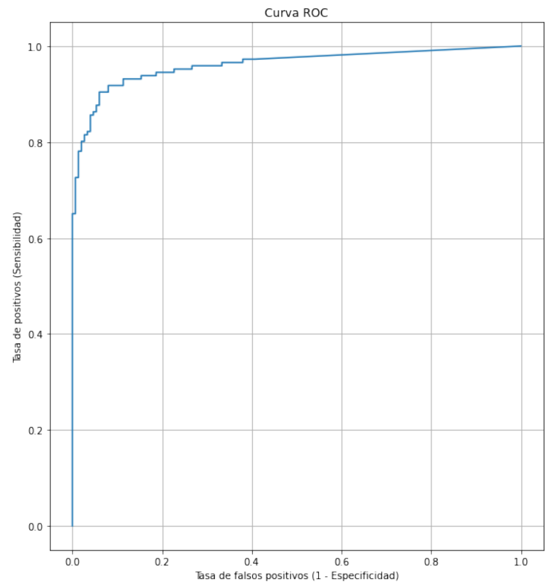


3.3.8 Nearest Neighbours

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier

model_name = "NearestNeighbors"

def objective_func(trial):
    # 1. Instanciar el modelo
    n_neighbors = trial.suggest_int('n_neighbors', 2, 10)
    weights = trial.suggest_categorical("weights",['uniform', 'distance'])
    algorithm  = trial.suggest_categorical( 'algorithm', ['ball_tree',␣
↪'kd_tree', 'brute'])
    leaf_size = trial.suggest_int('leaf_size', 10, 100)
    p = trial.suggest_int('p', 1, 2)

    model = KNeighborsClassifier( n_neighbors = n_neighbors, weights=weights,␣
↪algorithm = algorithm,
                              leaf_size = leaf_size, p = p )

    # 2. Entrenar
    model = model.fit(X_train,y_train)

    # 3. Evaluar en test set
    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob[:,1])
    auc_score = metrics.auc(fpr, tpr)

    #trial.report()
    trial.set_user_attr(key="best", value=model)
    return auc_score

study = optuna.create_study(direction='maximize',sampler=TPESampler())
study.optimize(objective_func, n_trials=100,callbacks=[keep_best_model])
```

[51]:
```
best_model=study.user_attrs["best"]
y_pred = best_model.predict(X_test)
y_pred_prob = best_model.predict_proba(X_test)
model_metrics_list[model_name]=model_evaluation_report(best_model,y_test,y_pred,y_pred_prob,mo
save_model(study,model_name)
```





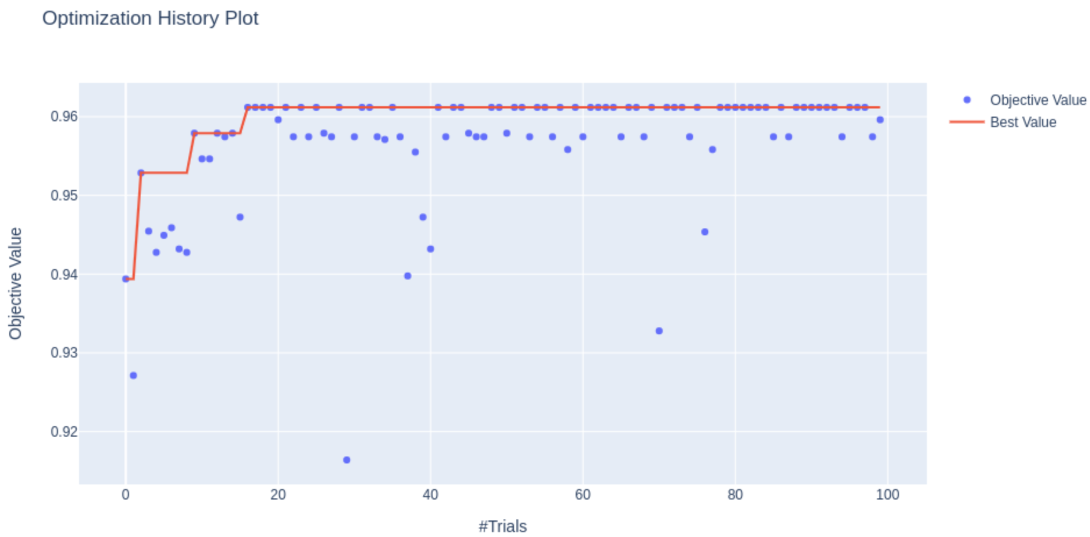

```
AUC: 0.9611872146118721
Accuracy: 0.9054054054054054
Precision: 0.9538461538461539
Recall: 0.8493150684931506
f1-score:  0.8985507246376813
Almacenado modelo:  /models/NearestNeighbors.pkl
Almacenados parámetros de modelo:  /models/NearestNeighbors.json
```

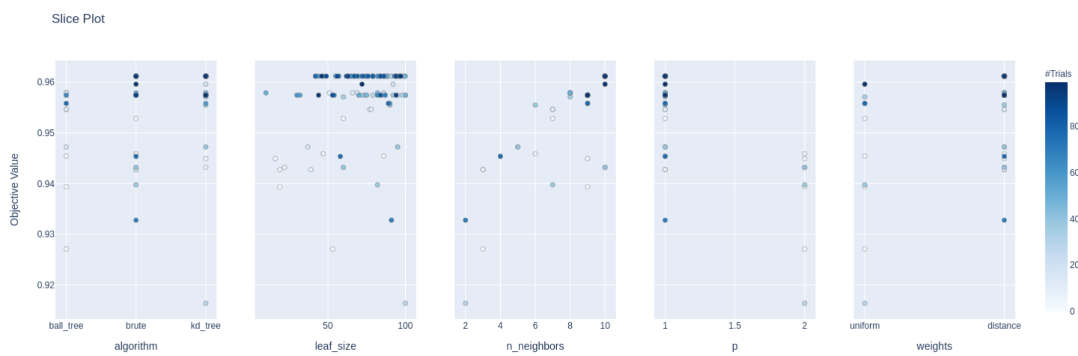
[52]:
```
trials_df = study.trials_dataframe(attrs=('number', 'value', 'params', 'state'))
trials_df.sort_values('value',ascending=False).head(5)
```

[52]:
```
    number     value params_algorithm  params_leaf_size  params_n_neighbors  \
78      78  0.961187          kd_tree                42                  10
52      52  0.961187            brute                67                  10
28      28  0.961187          kd_tree                89                  10
73      73  0.961187          kd_tree                63                  10
72      72  0.961187            brute                75                  10

    params_p params_weights     state
78         1       distance  COMPLETE
52         1       distance  COMPLETE
28         1       distance  COMPLETE
73         1       distance  COMPLETE
72         1       distance  COMPLETE
```

[53]:
```
optuna.visualization.plot_optimization_history(study)
```

[54]:
```
optuna.visualization.plot_slice(study)
```





1.3 3.3 Comparación de resultados y conclusiones

A continuación se presentan las métricas para el Test Set de los modelos entrenados:

[59]:
```
df = pd.DataFrame.from_dict(model_metrics_list,orient='index')
df.sort_values(by="auc", ascending=False)
```

[59]:

	accuracy	precision	recall	auc	f1-score	\
xgboost	0.925676	0.992063	0.856164	0.979452	0.919118	
RandomForest	0.925676	0.969697	0.876712	0.976438	0.920863	
MLP	0.905405	0.909722	0.897260	0.976210	0.903448	
SVM	0.922297	0.976744	0.863014	0.973333	0.916364	
DecisionTree	0.902027	0.887417	0.917808	0.965845	0.902357	
LogisticRegression	0.908784	0.921986	0.890411	0.964087	0.905923	
NearestNeighbors	0.905405	0.953846	0.849315	0.961187	0.898551	
base	0.493243	0.493243	1.000000	0.500000	0.660633	

	description
xgboost	xgboost

```
RandomForest                  RandomForest
MLP                                    MLP
SVM                                    SVM
DecisionTree                  DecisionTree
LogisticRegression  LogisticRegression
NearestNeighbors          NearestNeighbors
base                            Modelo base
```

[60]:
```
df.to_csv(MODELS_PATH+"model_summary.csv")
```

A modo de primer conclusión, comparando con los resultados del trabajo anterior de Machine Learning 1:

```
df.sort_values(by="auc", ascending=False).head(5)
```

	accuracy	precision	recall	auc	f1-score	description
xgboost	0.934010	0.978495	0.892157	0.978947	0.933333	XGBoost
svm1	0.903553	0.946237	0.862745	0.969763	0.902564	Support Vector Machine (kernel lineal)
rf_nest_30_mss_5_msl_3	0.908629	0.956522	0.862745	0.965067	0.907216	RF. N_est: 30. Part. samples: 5. Min samples. 3
rf_nest_200_mss_2_msl_4	0.903553	0.966292	0.843137	0.963777	0.900524	RF. N_est: 200. Part. samples: 2. Min samples. 4
rf_nest_200_mss_4_msl_8	0.913706	0.988506	0.843137	0.962539	0.910053	RF. N_est: 200. Part. samples: 4. Min samples. 8

se puede afirmar que la búsqueda Bayesiana de hiperparámetros con el algoritmo TPE tiene un efecto positivo, dado que todos los puntajes para todos los modelos han aumentado. En ambos casos el mejor puntaje es de XGBoost, y con la optimización se han obtenido mejores resultados para RandomForest y MLP que para SVM.

3.4 Trabajo futuro

- Agregar preprocesamiento a las opciones de hiperparámetros.
- Experimentar con Adaptative TPE.
- Automatizar el ciclo completo con TPO: preprocesamiento, evaluación de modelos con optimización debúsqueda de hiperparámetros y selección de modelo final.