Guia Optuna

Ronald Junior Pilco Nuñe

¿Qué es Optuna?

Optuna es una biblioteca de optimización automática de hiperparámetros de código abierto, desarrollada para simplificar y mejorar el ajuste de modelos predictivos en tareas de machine learning. Su enfoque se basa en la optimización bayesiana y técnicas avanzadas de búsqueda eficiente como la estimación de funciones objetivo.

Las principales características de Optuna incluyen: Github

- **Definición flexible de espacios de búsqueda**: permite manejar espacios de búsqueda complejos.
- **Optimización eficiente**: utiliza estrategias avanzadas como TPE (*Tree-structured Parzen Estimator*).
- **Integración directa con frameworks de machine learning**: compatible con scikit-learn, PyTorch, TensorFlow, entre otros.
- **Visualización de resultados**: proporciona herramientas para analizar los resultados de las optimizaciones.

Instalación

Para instalar Optuna, utiliza el siguiente comando en tu terminal:

```
pip install optuna
```

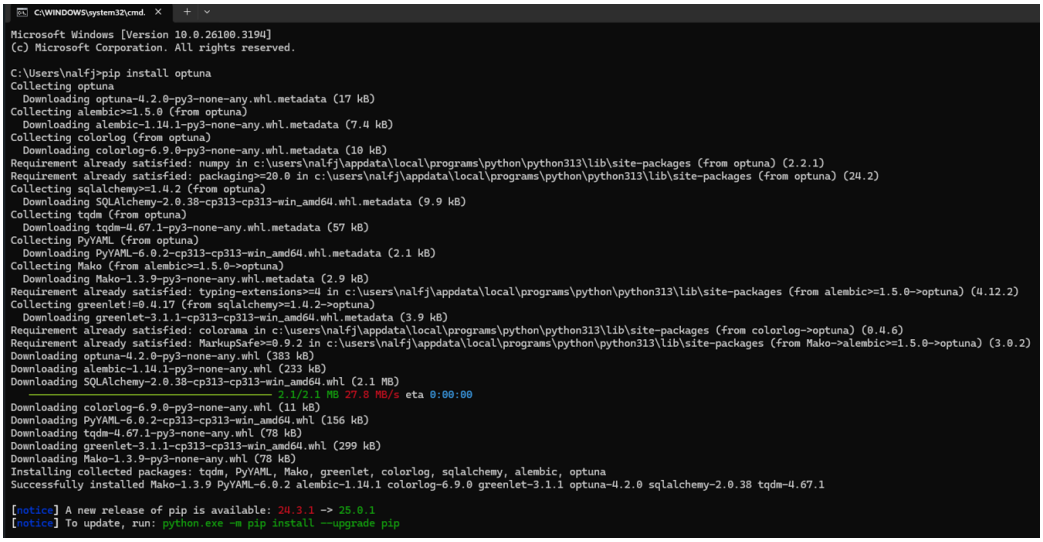


Figura 1: Instalacion

Cómo usar Optuna

A continuación se muestra un ejemplo básico de cómo usar Optuna para optimizar los hiperparámetros de una función:

Ejemplo básico

Supongamos que queremos encontrar el valor mínimo de la función $f(x) = (x-3)^2$.

```
import optuna

def objective(trial):
    x = trial.suggest_float("x", -10, 10)  # Espacio de
        b squeda
    return (x - 3) ** 2  # Funci n objetivo

# Crear un estudio y ejecutar la optimizaci n
study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=100)

# Mostrar los resultados
print(f"Mejor valor encontrado: {study.best_value}")
print(f"Mejores par metros: {study.best_params}")
```

Explicación del código

- `trial.suggest_float`: Define el rango del espacio de búsqueda para el parámetro x.
- `study.optimize`: Ejecuta el proceso de optimización durante `n_trials` iteraciones.
- `study.best_value` y `study.best_params`: Devuelven el mejor resultado y los parámetros asociados.

optimización de hiperparámetros

El código realiza los siguientes pasos principales:

1. **Carga del conjunto de datos Iris**: Utiliza `sklearn.datasets.load_iris()` para cargar el conjunto de datos Iris, que contiene características de tres especies de flores.

2. **Definición de modelos**: Se emplean dos modelos de clasificación:

   - `RandomForestClassifier`.
   - `SVC (Support Vector Classifier)`.

3. **Optimización de hiperparámetros**:

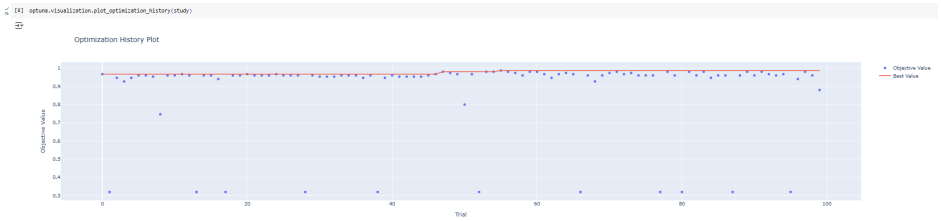
   - `RandomForestClassifier`: Optimiza los parámetros `n_estimators` (número de árboles) y `max_depth` (profundidad máxima).
   - `SVC`: Optimiza el parámetro `C` (regularización).

4. **Validación cruzada**: Calcula la precisión promedio de los modelos mediante validación cruzada (`cross_val_score`).

5. **Visualización de resultados**: Usa `optuna.visualization` para analizar el historial de optimización y las relaciones entre los hiperparámetros y el rendimiento.

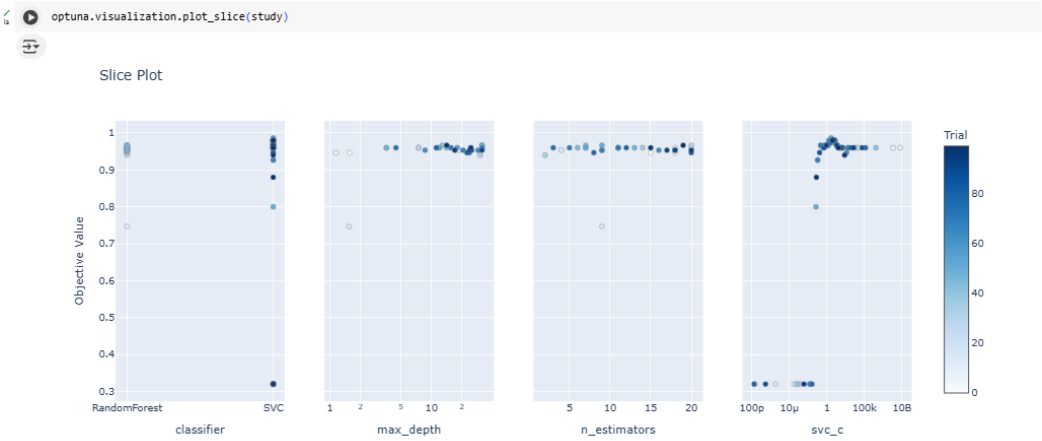
El siguiente fragmento ilustra la estructura principal de la optimización:

```
def objective(trial):
    iris = sklearn.datasets.load_iris()
    classifier = trial.suggest_categorical("classifier", ["
       RandomForest", "SVC"])

    if classifier == "RandomForest":
        n_estimators = trial.suggest_int("n_estimators", 2,
           20)
        max_depth = int(trial.suggest_float("max_depth", 1,
           32, log=True))
        clf = sklearn.ensemble.RandomForestClassifier(
           n_estimators=n_estimators, max_depth=max_depth)
    else:
        c = trial.suggest_float("svc_c", 1e-10, 1e10, log=
           True)
        clf = sklearn.svm.SVC(C=c, gamma="auto")

    return sklearn.model_selection.cross_val_score(clf, iris.
       data, iris.target, n_jobs=-1, cv=3).mean()
```



Historial de Optimización

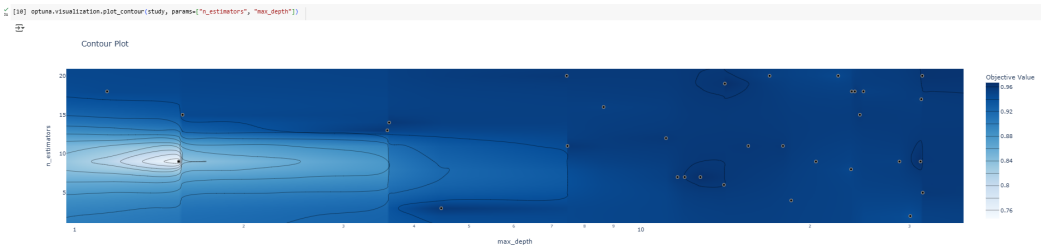
En la gráfica de historial de optimización (Optimization History Plot), se observa cómo el valor objetivo (MSE) converge hacia un mínimo a medida que avanzan los ensayos. Las primeras pruebas presentan un mayor error, lo cual es característico del proceso de exploración inicial. La línea roja indica el mejor valor encontrado, mostrando una estabilización en los últimos ensayos.



Impacto de los Hiperparámetros

El gráfico *Slice Plot* muestra la influencia de los hiperparámetros sobre el valor objetivo:

- El modelo `RandomForest` presenta un desempeño consistentemente mejor que `SVR`.
- En el caso de `RandomForest`, los hiperparámetros `max_depth` y `n_estimators` tienen una relación clara con la reducción del MSE. Valores intermedios de profundidad y un número adecuado de estimadores producen mejores resultados.
- Para `SVR`, los valores del hiperparámetro `C` muestran una mayor variabilidad en el MSE, indicando menor estabilidad.



Análisis de Contornos

El gráfico de contornos (*Contour Plot*) analiza el espacio de búsqueda para `RandomForest`. Los colores más oscuros indican combinaciones óptimas de `n_estimators` y `max_depth`, donde el MSE es mínimo. Se identificó una región específica donde el modelo alcanza su mejor desempeño.

Conclusiones Resultados

- El modelo `RandomForest` es más efectivo que `SVR` para el conjunto de datos evaluado.
- Las combinaciones óptimas de hiperparámetros pueden visualizarse claramente en los gráficos de contornos, lo que facilita la interpretación y selección.
- Optuna permitió una exploración eficiente del espacio de hiperparámetros, convergiendo rápidamente hacia soluciones óptimas.

Código

Modificación para trabajar con otras bases de datos

Para utilizar otras bases de datos, como un archivo CSV o una base de datos SQL, se deben realizar las siguientes modificaciones:

1. Cargar los datos

En lugar de usar `sklearn.datasets.load_iris()`, se puede cargar una base de datos personalizada. Ejemplo para un archivo CSV:

```
import pandas as pd
data = pd.read_csv("ruta_a_tu_archivo.csv")

X = data.drop("target", axis=1)  # Caracter sticas
y = data["target"]  # Etiqueta
```

Para una base de datos SQL:

```
import pandas as pd
from sqlalchemy import create_engine

engine = create_engine("sqlite:///ruta_a_tu_base_de_datos.db"
    )
data = pd.read_sql("SELECT␣*␣FROM␣tabla", engine)

X = data.drop("target", axis=1)
y = data["target"]
```

2. Adaptar el modelo a los datos

Si las dimensiones de las características (`X`) o las etiquetas (`y`) cambian, los modelos deben ajustarse automáticamente. Por ejemplo, al usar clasificación multiclase, es importante verificar que los modelos sean compatibles.

3. Cambiar el pipeline de evaluación

Si el conjunto de datos es grande, se puede usar una partición explícita de entrenamiento y prueba en lugar de validación cruzada:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)

def objective(trial):
```

```
    classifier = trial.suggest_categorical("classifier", ["
        RandomForest", "SVC"])

    if classifier == "RandomForest":
        n_estimators = trial.suggest_int("n_estimators", 2,
            20)
        max_depth = int(trial.suggest_float("max_depth", 1,
            32, log=True))
        clf = sklearn.ensemble.RandomForestClassifier(
            n_estimators=n_estimators, max_depth=max_depth)
    else:
        c = trial.suggest_float("svc_c", 1e-10, 1e10, log=
            True)
        clf = sklearn.svm.SVC(C=c, gamma="auto")

    clf.fit(X_train, y_train)
    return clf.score(X_test, y_test)
```

4. Visualización de resultados

Si trabajas con datos complejos, puedes incluir visualizaciones personalizadas para interpretar los resultados en el contexto de tu problema.

Conclusión

Optuna es una herramienta poderosa para la optimización automática de hiperparámetros, especialmente en proyectos de machine learning donde encontrar los mejores parámetros puede ser un desafío. Con su simplicidad y eficiencia, Optuna permite a los investigadores y desarrolladores ahorrar tiempo y obtener mejores resultados.

Para más información, visita Optuna.