

Universidad Nacional del Altiplano

Facultad de Ingeniería Estadística e Informática

Docente: Fred Torres Cruz

Autor : Ronald Junior Pilco Nuñez

Manual Optuna

[Github](#)

Índice

1. Introducción a Optuna	2
2. Conceptos Básicos	2
2.1. Estudio (Study)	2
2.2. Prueba (Trial)	2
2.3. Espacio de Búsqueda	2
3. Instalación	3
4. Ejemplos	3
4.1. Optimización de Hiperparámetros	3
4.2. Condicionamiento de Hiperparámetros	4
4.3. Optimización de Modelos de IA con Optuna y Datos de Uso de IA (OptimizacionModelosUso _{IA} .py)	5
5. Optimización Avanzada	6
5.1. Samplers	6
5.2. Pruning	7
6. Visualización	8
6.1. Gráfico de Optimización	8
6.2. Gráfico de Importancia de Hiperparámetros	8
6.3. Gráfico de Dependencia	8
6.4. Gráfico de Contorno	9

1. Introducción a Optuna

Optuna es un framework de optimización de hiperparámetros diseñado para ser fácil de usar y eficiente. Es especialmente útil para la optimización de modelos de machine learning, pero también puede ser utilizado en otros contextos donde se necesite optimizar parámetros.

2. Conceptos Básicos

2.1. Estudio (Study)

Un **estudio** en Optuna es un conjunto de ejecuciones de optimización con el objetivo de maximizar o minimizar una métrica de rendimiento. Se crea usando `optuna.create_study()` y se ejecuta con `study.optimize()`.

```
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=10)
```

2.2. Prueba (Trial)

Cada ejecución en un estudio es una **prueba**. Cada prueba evalúa una combinación de hiperparámetros definida por el objeto `trial`. Los resultados de cada prueba guían la búsqueda hacia la mejor configuración.

```
def objective(trial):
    n_estimators = trial.suggest_int("n_estimators", 10, 100)
    max_depth = trial.suggest_int("max_depth", 1, 10)
    clf = RandomForestClassifier(n_estimators=n_estimators, max_depth=
                               max_depth)
    return cross_val_score(clf, X_train, y_train).mean()
```

2.3. Espacio de Búsqueda

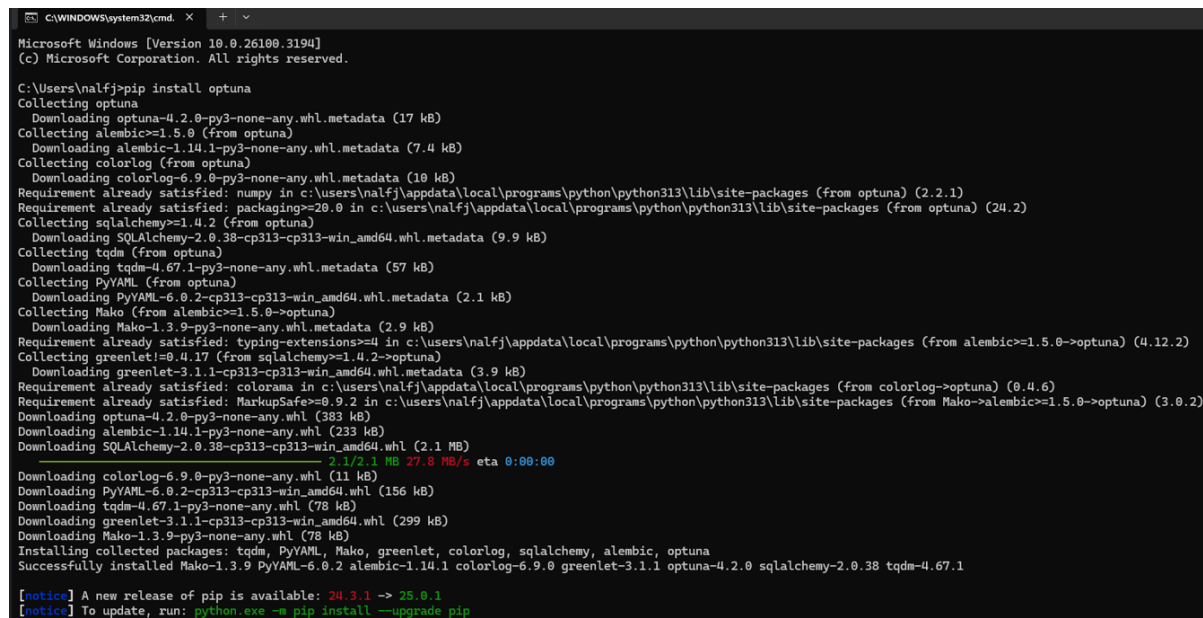
El **espacio de búsqueda** define los rangos y distribuciones de los hiperparámetros. Se puede usar `suggest_int`, `suggest_float` o `suggest_categorical` para definir las opciones a explorar.

```
n_estimators = trial.suggest_int("n_estimators", 10, 100)
max_depth = trial.suggest_int("max_depth", 1, 10)
classifier = trial.suggest_categorical("classifier", ["RandomForest", "
SVC"])
```

3. Instalación

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

```
pip install optuna
```



```
C:\WINDOWS\system32\cmd. X + v
Microsoft Windows [Version 10.0.26100.3194]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalfj>pip install optuna
Collecting optuna
  Downloading optuna-4.2.0-py3-none-any.whl.metadata (17 kB)
Collecting alembic<=1.5.0 (from optuna)
  Downloading alembic-1.14.1-py3-none-any.whl.metadata (7.4 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in c:\users\nalfj\appdata\local\programs\python\python313\lib\site-packages (from optuna) (2.2.1)
Requirement already satisfied: packaging<=20.0 in c:\users\nalfj\appdata\local\programs\python\python313\lib\site-packages (from optuna) (24.2)
Collecting sqlalchemy<=1.4.2 (from optuna)
  Downloading SQLAlchemy-2.0.38-cp313-cp313-win_amd64.whl.metadata (9.9 kB)
Collecting tqdm (from optuna)
  Downloading tqdm-4.67.1-py3-none-any.whl.metadata (57 kB)
Collecting PyYAML (from optuna)
  Downloading PyYAML-6.0.2-cp313-cp313-win_amd64.whl.metadata (2.1 kB)
Collecting Mako (from alembic<=1.5.0->optuna)
  Downloading Mako-1.3.9-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: typing-extensions<=4 in c:\users\nalfj\appdata\local\programs\python\python313\lib\site-packages (from alembic<=1.5.0->optuna) (4.12.2)
Collecting greenlet!=0.4.17 (from sqlalchemy<=1.4.2->optuna)
  Downloading greenlet-3.1.1-cp313-cp313-win_amd64.whl.metadata (3.9 kB)
Requirement already satisfied: colorama in c:\users\nalfj\appdata\local\programs\python\python313\lib\site-packages (from colorlog->optuna) (0.4.6)
Requirement already satisfied: MarkupSafe<=0.9.2 in c:\users\nalfj\appdata\local\programs\python\python313\lib\site-packages (from Mako->alembic<=1.5.0->optuna) (3.0.2)
Downloading optuna-4.2.0-py3-none-any.whl (383 kB)
Downloading alembic-1.14.1-py3-none-any.whl (233 kB)
Downloading SQLAlchemy-2.0.38-cp313-cp313-win_amd64.whl (2.1 MB)
----- 2.1/2.1 MB 27.8 MB/s eta 0:00:00
Downloading colorlog-6.9.0-py3-none-any.whl (11 kB)
Downloading PyYAML-6.0.2-cp313-cp313-win_amd64.whl (156 kB)
Downloading tqdm-4.67.1-py3-none-any.whl (78 kB)
Downloading greenlet-3.1.1-cp313-cp313-win_amd64.whl (299 kB)
Downloading Mako-1.3.9-py3-none-any.whl (78 kB)
Installing collected packages: tqdm, PyYAML, Mako, greenlet, colorlog, sqlalchemy, alembic, optuna
Successfully installed Mako-1.3.9 PyYAML-6.0.2 alembic-1.14.1 colorlog-6.9.0 greenlet-3.1.1 optuna-4.2.0 sqlalchemy-2.0.38 tqdm-4.67.1

[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

4. Ejemplos

Optimización de Hiperparámetros de Modelos de Clasificación con Optuna

Requerimientos para los ejemplos

```
pip install optuna
pip install scikit-learn
pip install plotly
pip install matplotlib
```

4.1. Optimización de Hiperparámetros

El siguiente código muestra cómo optimizar los hiperparámetros de un modelo de Random Forest utilizando Optuna.

```
import optuna

def objective(trial):
    iris = sklearn.datasets.load_iris()

    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)

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

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=10)

trial = study.best_trial

print("Accuracy: {}".format(trial.value))
print("Best hyperparameters: {}".format(trial.params))
```

4.2. Condicionamiento de Hiperparámetros

Es posible condicionar hiperparámetros utilizando sentencias 'if' en Python. En este ejemplo, se incluyen dos clasificadores: Random Forest y Support Vector Machine (SVM).

```
import sklearn.svm

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()

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=10)

trial = study.best_trial

print("Accuracy: {}".format(trial.value))
print("Best hyperparameters: {}".format(trial.params))
```

4.3. Optimización de Modelos de IA con Optuna y Datos de Uso de IA (OptimizacionModelosUso_{IA}.py)

```

import optuna
import pandas as pd
import sklearn.ensemble
import sklearn.model_selection
import sklearn.svm
import optuna.visualization
from sklearn.preprocessing import LabelEncoder

def load_data(csv_path):
    df = pd.read_csv(csv_path)
    X = df.iloc[:, :-1].values # Todas las columnas excepto la ltima
    y = df.iloc[:, -1].values # ltima columna como variable objetivo

    # Si la variable objetivo es categorica, la convertimos a valores
    # num rcos
    if y.dtype == 'object':
        y = LabelEncoder().fit_transform(y)

    return X, y

def objective(trial):
    X, y = load_data("uso_ia_data.csv") # Usar la base de datos de uso
    # de IA

    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, X, y, n_jobs=-1, cv=3
    ).mean()

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=100)

trial = study.best_trial

print("Accuracy: {}".format(trial.value))
print("Best hyperparameters: {}".format(trial.params))

# Graficar los resultados de la optimización
optuna.visualization.plot_optimization_history(study).show()

```

```
optuna.visualization.plot_slice(study).show()
optuna.visualization.plot_contour(study, params=["n_estimators", "
    max_depth"]).show()
```

Este programa utiliza **Optuna** para optimizar hiperparámetros de modelos de clasificación (**RandomForestClassifier** y **SVC**) basados en datos sobre uso de IA. Se entrena y evalúa mediante validación cruzada, maximizando la precisión del modelo.

Resultado:

```
Accuracy: 0.4571078431372549
Best hyperparameters: {'classifier': 'RandomForest', 'n_estimators': 10, 'max_depth': 22.041516843146912}
```

Interpretación del resultado:

El mejor modelo encontrado es un **RandomForestClassifier** con **10** árboles y una profundidad máxima de aproximadamente **22**. Sin embargo, la precisión alcanzada (**0.457**) es baja, lo que sugiere que el modelo no está generalizando bien. Posibles mejoras incluyen el preprocesamiento de datos, selección de características o ajuste de hiperparámetros adicionales.

5. Optimización Avanzada

5.1. Samplers

En Optuna, el muestreo de hiperparámetros es un proceso clave para la búsqueda de la mejor combinación. Optuna ofrece varios algoritmos de muestreo, cada uno con ventajas en diferentes situaciones. Los tres samplers más comunes son:

- **TPE (Tree-structured Parzen Estimator)**: Este algoritmo probabilístico modela la distribución de los hiperparámetros a través de un árbol estructurado, el cual se ajusta de manera eficiente a partir de la información obtenida durante la optimización. El TPE es especialmente útil cuando el espacio de búsqueda es complejo y no necesariamente convexo.
- **CMA-ES (Covariance Matrix Adaptation Evolution Strategy)**: Es un algoritmo de optimización evolutiva que se utiliza principalmente en espacios de búsqueda continuos y multidimensionales. El CMA-ES ajusta la matriz de covarianza de una distribución normal

para generar muestras más efectivas y explorar el espacio de manera más dinámica.

- **Random Sampler:** Este es el algoritmo más simple, ya que selecciona combinaciones de hiperparámetros de manera completamente aleatoria dentro del espacio de búsqueda. Aunque no es tan eficiente como los métodos probabilísticos, puede ser útil en exploraciones iniciales o cuando no se tiene una idea clara del espacio de búsqueda.

```
# Usar TPE como sampler en el estudio
study = optuna.create_study(sampler=optuna.samplers.TPESampler(),
                             direction="maximize")
study.optimize(objective, n_trials=100)
```

5.2. Pruning

El **pruning** (o poda) es una técnica que permite interrumpir pruebas que no están ofreciendo buenos resultados, con el objetivo de ahorrar tiempo y recursos computacionales. Esto es especialmente útil cuando se trabaja con modelos costosos de entrenar, como redes neuronales o modelos complejos.

Optuna tiene implementado un conjunto de *pruners* que determinan cuándo interrumpir una prueba. Algunos de los *pruners* más comunes son:

- **MedianPruner:** Interrumpe las pruebas que están por debajo de la mediana de las pruebas anteriores en el estudio. Esto puede ser útil para identificar y eliminar combinaciones de hiperparámetros no prometedoras rápidamente.
- **PercentilePruner:** Similar al *MedianPruner*, pero interrumpe las pruebas que están por debajo de un percentil específico de las mejores pruebas realizadas hasta el momento.
- **SuccessiveHalvingPruner:** Divide el número de recursos (por ejemplo, número de épocas de entrenamiento) entre las pruebas, priorizando las más prometedoras en cada etapa.

A continuación, se muestra cómo aplicar un *pruner* en un estudio:

```
import optuna
from optuna.pruners import MedianPruner

study = optuna.create_study(direction='maximize', pruner=MedianPruner())
study.optimize(objective, n_trials=100)
```

En este ejemplo, el *MedianPruner* interrumpe las pruebas que no están mejorando y que se encuentran por debajo de la mediana de los resultados previos.

6. Visualización

Optuna ofrece potentes herramientas de visualización para analizar y comprender el proceso de optimización, así como los resultados obtenidos. Estas visualizaciones permiten evaluar tanto la evolución de la optimización como la importancia de cada hiperparámetro.

6.1. Gráfico de Optimización

El **gráfico de optimización** muestra cómo ha evolucionado el valor objetivo durante las pruebas del estudio. Este gráfico es útil para observar el progreso general de la optimización, así como identificar posibles estancamientos o la efectividad de la estrategia de optimización utilizada.

```
optuna.visualization.plot_optimization_history(study)
```

Este gráfico ilustra la evolución de la función objetivo a lo largo de las pruebas, lo que ayuda a evaluar si la optimización está convergiendo correctamente.

6.2. Gráfico de Importancia de Hiperparámetros

El **gráfico de importancia de hiperparámetros** muestra qué tan importante es cada hiperparámetro en el rendimiento del modelo. Este análisis es crucial para entender qué hiperparámetros tienen un mayor impacto en el modelo y orientar mejor las futuras iteraciones de optimización.

```
optuna.visualization.plot_param_importances(study)
```

Este gráfico permite identificar de manera visual qué parámetros son más relevantes para la función objetivo, lo que puede guiar la selección de los hiperparámetros a optimizar en futuras ejecuciones.

6.3. Gráfico de Dependencia

El **gráfico de dependencia** visualiza cómo el valor de un hiperparámetro afecta a la función objetivo, teniendo en cuenta los valores de otros hiperparámetros. Este tipo de gráfico ayuda a identificar interacciones entre los hiperparámetros que pueden ser cruciales para la optimización.


```
optuna.visualization.plot_parallel_coordinate(study)
```

Este gráfico es especialmente útil para observar cómo varían varias combinaciones de hiperparámetros y su impacto en el rendimiento del modelo.

6.4. Gráfico de Contorno

El **gráfico de contorno** muestra cómo cambian los valores de la función objetivo en función de dos hiperparámetros específicos. Es útil cuando se desea entender cómo interactúan dos hiperparámetros y cómo afectan el rendimiento del modelo.

```
optuna.visualization.plot_contour(study, params=["n_estimators", "max_depth"])
```

Este gráfico proporciona una representación visual de la superficie de precisión en función de los hiperparámetros seleccionados, lo que permite una exploración más detallada del espacio de búsqueda.

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.org](https://optuna.org).