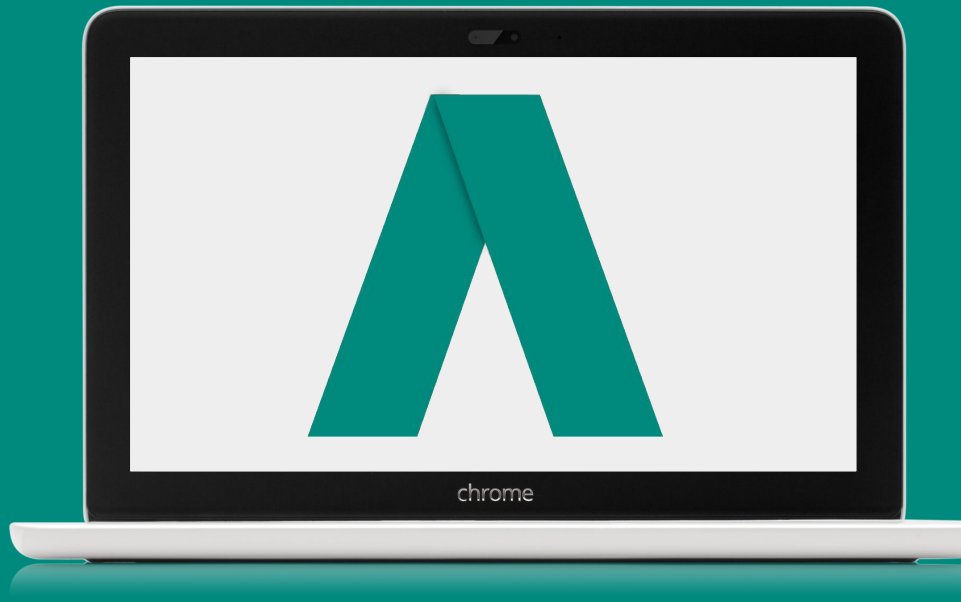
Hyperparameter Tuning



But first: Parameters.

Parameters are components of the model learned during the modeling process. You do not set these parameters manually, in fact, you cannot set them manually. They are not the same as hyperparameters.

Parameters:

- are necessary for the model to make predictions.
- values determine the model's ability to solve your problem.
- are calculated or figured out using data.
- are frequently saved as a part of the learned model.
- are not set by the developer.

https://www.datacamp.com/tutorial/parameter-optimization-machine-learning-models

Hyperparameters.

A hyperparameter, on the other hand, is a configuration that is external to the model and whose value cannot be estimated from data

- They control the model's behavior and learning process.
- They are not learned during training but are set by the data scientist or engineer before the model training begins.
- Hyperparameter tuning involves finding the optimal set of hyperparameter values that result in the best model performance.

https://www.datacamp.com/tutorial/parameter-optimization-machine-learning-models

Examples of hyperparameters.

Key hyperparameters for **logistic regression** include:

- Regularization type
- Regularization strength (C)
- Solver
- Class weight

Random forests have several hyperparameters that control the construction and behavior of the ensemble of decision trees:

- n_estimators
- max_depth
- min_samples_split
- min_samples_leaf
- max_features

K-Nearest Neighbors (KNN) is a simple and widely used machine learning algorithm for classification and regression tasks:

- n_neighbors
- weights
- algorithm
- leaf_size

Read more about hyperparameters for Logistic Regression
Read more about hyperparameters for RandomForestClassifier
Read more about hyperparameters for K-Nearest Neighbors

Methods for hyperparameter tuning.

GridSearchCV and **RandomizedSearchCV** are popular methods for hyperparameter tuning provided by the scikit-learn library in Python. They perform an exhaustive search over a specified hyperparameter space and use cross-validation to estimate the model's performance.

GridSearchCV

GridSearchCV is a method that performs an exhaustive search over a specified hyperparameter grid, meaning it tries out every possible combination of hyperparameters. For each combination, it evaluates the model using cross-validation and computes a performance score based on a specified evaluation metric.

Advantages:

- It is guaranteed to find the best hyperparameter combination in the specified grid.

Disadvantages:

- It can be computationally expensive, especially when the search space is large or the model takes a long time to train.
- It does not scale well with the number of hyperparameters.

To use GridSearchCV, you need to:

1. Define a model (estimator) for which you want to tune the hyperparameters.
2. Specify a dictionary containing the hyperparameters and their possible values.
3. Choose a cross-validation strategy and an evaluation metric.
4. Fit the GridSearchCV object to your data.

RandomizedSearchCV

RandomizedSearchCV is an alternative to GridSearchCV that samples a fixed number of hyperparameter combinations from the specified search space. Instead of trying every possible combination, it randomly selects a subset, which can save time and computational resources.

Advantages:

- It is more efficient than GridSearchCV, especially for large search spaces or computationally expensive models.
- It allows specifying a number of iterations it will explore

Disadvantages:

- It is not guaranteed to find the best combination in the specified search space, as it samples only a subset of the possible combinations.

To use RandomizedSearchCV, you need to:

1. Define a model (estimator) for which you want to tune the hyperparameters.
2. Specify a dictionary containing the hyperparameters and their possible values or distributions.
3. Choose a cross-validation strategy, an evaluation metric, and the number of iterations.
4. Fit the RandomizedSearchCV object to your data.

GridSearchCV sample code

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Define the model
logreg_classifier = LogisticRegression()

# Define the hyperparameter grid
param_grid = {
        'penalty': ['l1', 'l2', 'elasticnet', 'none'],
        'C': [0.001, 0.01, 0.1, 1, 10, 100],
        'fit_intercept': [True, False],
        'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
'class_weight': [None, 'balanced']
}
```

```
# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=logreg_classifier, param_grid=param_grid, scoring='accuracy', cv=5)

# Fit the object to the data
grid_search.fit(X_train, y_train)

# Get the best hyperparameter combination
best_params = grid_search.best_params_
```

Read more about scoring

RandomizedSearchCV sample code

```
from sklearn.model_selection import
RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from scipy.stats import uniform, randint

# Define the model
logreg_classifier = LogisticRegression()

# Define the hyperparameter search space
param_dist = {
        'penalty': ['l1', 'l2', 'elasticnet', 'none'],
        'C': uniform(loc=0.001, scale=100),
        'fit_intercept': [True, False],
        'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
        'class_weight': [None, 'balanced']
}
```

```
# Create the RandomizedSearchCV object
random_search =
RandomizedSearchCV(estimator=logreg_classifier,
param_distributions=param_dist, n_iter=50,
scoring='accuracy', cv=5)

# Fit the object to the data
random_search.fit(X_train, y_train)

# Get the best hyperparameter combination
best_params = random_search.best_params_
```