Optimization Techniques for Machine Learning

Alassane KONE
(ISE 2019 – Senior Data Scientist)

February 2023

Chapter 5

Hyperparameters Optimization

Overview

Machine learning models have **hyperparameters** that you must set in order to customize the model to your dataset.

Often the general effects of hyperparameters on a model are known, but how to best set a hyperparameter and combinations of interacting hyperparameters for a given dataset is challenging.

A better approach is to objectively search different values for model hyperparameters and choose a subset that results in a model that achieves the best performance on a given dataset. This is called **hyperparameter optimization or hyperparameter tuning**.

The result of a hyperparameter optimization is a single set of well-performing hyperparameters that you can use to configure your model.

1. Classification Problem

Hyperparameter Optimization in a Classification

Initialization

```
In this section, we will use hyperparameter optimization to discover a well-performing model
configuration for the sonar dataset.

The sonar dataset is a standard machine learning dataset comprising 208 rows of data with 60 numerical
input variables and a target variable with two class values, e.g. binary classification.

# 1. Read the dataset data_0004.csv and splits it into input and output elements and print their shape

# 2. Initialize a LogisticRegression model into a variable called model

# 3. Into a variable called cv, initialize an instance of ... RepeatedStratifiedKFold(n_splits=10,
n_repeats=3, random_state=1)

# 4. Create an empty dictionnary called space

# 5. Explain what we are doing in the code below
space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
space['C'] = loguniform(1e-5, 100)
```

Hyperparameter Optimization in a Classification

Random Search

```
# 6. from sklearn.model_selection import RandomizedSearchCV

# 7. Into a variable search, initialize an instance of RandomizedSearchCV(model, space, n_iter=500,
scoring='accuracy', n_jobs=-1, cv=cv, random_state=1)

# 8. Fit search on (X,y) and put the output into a variable called result

# 9. Run the code below:
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)
```

Hyperparameter Optimization in a Classification

Grid Search

```
# 10. Create another empty dict called space2

# 11. Run the code below and explain what it does:
space2['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
space2['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
space2['C'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]

# 12. Into a variable search2, initialize an instance of search = GridSearchCV(model, space2,
scoring='accuracy', n_jobs=-1, cv=cv)

# 13. Fit search2 on (X,y) and put the output into a variable called result2

# 14. Summarize the output as done previously
```

Hyperparameter Optimization in a Classification

Application – On a Gradient Boosting Classifier

```
We are going to replicate, on an Gradient Boosting Classifier, everything we did previously on
LogisticRegression.

# 1. Initialize a variable model = GradientBoostingClassifier()

# 2. For the grid search, create a dict as below:
  n_estimators = [10, 100, 1000]
  learning_rate = [0.001, 0.01, 0.1]
  subsample = [0.5, 0.7, 1.0]
  max_depth = [3, 7, 9]
  grid = dict(learning_rate=learning_rate, n_estimators=n_estimators, subsample=subsample,
max_depth=max_depth)

# 3. For the random search, create a dict below:
from scipy.stats import uniform, randint, truncnorm, truncexpon
  n_estimators = randint(10, 1000)
  learning_rate = uniform(0.001, 0.1)
  subsample = uniform(0.3, 1)
  max_depth = randint(3, 10)
  random_grid = dict(learning_rate=learning_rate, n_estimators=n_estimators, subsample=subsample,
max_depth=max_depth)
```

2. Regression Problem

Hyperparameter Optimization in a Regression

Initialization

```
In this section we will use hyper optimization to discover a top-performing model configuration for the
auto insurance dataset.

The auto insurance dataset is a standard machine learning dataset comprising 63 rows of data with 1
numerical input variable and a numerical target variable.

A naive model can achieve a mean absolute error (MAE) of about 66.

The dataset involves predicting the total amount in claims (thousands of Swedish Kronor) given the
number of claims for different geographical regions.

# 1. Load the dataset data_0005.csv also check the description file

# 2. Split into input and output elements (X and y) and print the shape of both

# 3. Initialize in a variable called model a Ridge model (you will need to import it from
sklearn.linear_model)

# 4. Create a variable cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
```

Hyperparameter Optimization in a Regression

Random Search

```
# 5. Run the code below and explain it

    # define search space
    space = dict()
    space['solver'] = ['svd', 'cholesky', 'lsqr', 'sag']
    space['alpha'] = loguniform(1e-5, 100)
    space['fit_intercept'] = [True, False]
    space['normalize'] = [True, False]
    # define search
    search = RandomizedSearchCV(model, space, n_iter=500, scoring='neg_mean_absolute_error', n_jobs=-1,
             cv=cv, random_state=1)
    # execute search
    result = search.fit(X, y)
    # summarize result
    print('Best Score: %s' % result.best_score_)
    print('Best Hyperparameters: %s' % result.best_params_)
```