IT496: Introduction to Data Mining

Lecture 12

Optimization - I

[Search Methods]

Arpit Rana

29th August 2023

Components of Supervised Learning

Representation ✔

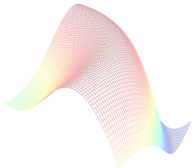
choosing the set of functions (*hypotheses space* or *the model class*) that can be learned.

$$h_\beta(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_m X_m = \sum_{i=1}^{m} \beta_i X_i$$

Evaluation ✔

An evaluation function (also called *objective function* or *scoring function*) is needed to distinguish good hypotheses from bad ones.

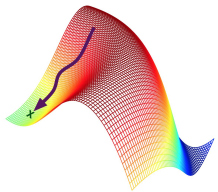
$$J(\beta) = \sum_{i=1}^{m} (h_\beta(X_i) - y_i)^2$$



Optimization

We need a method to search the hypothesis space for the highest-scoring one.

$$\arg min \; J(\beta)$$



Hyperparameters

Hyperparameters are parameters of the model class, not of the individual model.

- We define them before training to control the learning process.
- The *learner* algorithm does not learn these (can't be estimated from data).

Example:

```
from sklearn.ensemble import RandomForestClassifier

# Instantiate the model
rf_model = RandomForestClassifier()

# Print hyperparameters
rf_model.get_params
```

```
RandomForestClassifier(
bootstrap=True,
ccp_alpha=0.0, class_weight=None,
criterion='gini',
max_depth=None,
max_features='auto',
max_leaf_nodes=None,
max_samples=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
n_estimators=100,
n_jobs=None, oob_score=False,
random_state=None,
verbose=0, warm_start=False)
```

Hyperparameters

Hyperparameters are parameters of the model class, not of the individual model.

- We define them before training to control the learning process.
- The *learner* algorithm does not learn these (can't be estimated from data).

Example:

```
from sklearn.ensemble import RandomForestClassifier

# Instantiate the model
rf_model = RandomForestClassifier()

# Print hyperparameters
rf_model.get_params
```

Some hyperparameters are more important than others.

```
RandomForestClassifier(
bootstrap=True,
ccp_alpha=0.0, class_weight=None,
criterion='gini',
max_depth=None,
max_features='auto',
max_leaf_nodes=None,
max_samples=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
n_estimators=100,
n_jobs=None, oob_score=False,
random_state=None,
verbose=0, warm_start=False)
```

Parameters

(Model) Parameters are components of the model whose value can be estimated from data.

- We do not set these manually (we can't in fact!)
- The algorithm will discover these for us (learned during the training).

Example:

```
from sklearn import LogisticRegression

# Instantiate the model
log_reg_clf = LogisticRegression()

# Train the model
log_reg_clf.fit(X_train, y_train)

print(log_reg_clf.coef_)
```

```
array([[-2.88651273e-06,
-8.23168511e-03,
7.50857018e-04,
3.94375060e-04,
3.79423562e-04,
4.34612046e-04,
4.37561467e-04,
4.12107102e-04,
-6.41089138e-06,
-4.39364494e-06, cont... ]])
```

- For decision tree or random forest, *split column* (the attribute chosen for split) and *split column value* (the value of that attribute chosen for split) are examples of parameters that are learned while training.

Hyperparameter Tuning

Hyperparameter Tuning is choosing the best combination of hyperparameters. But, they can't be estimated from the data.

The following methods are commonly used for tuning the hyperparameters.

- Manual Search
- Grid Search
- Random Search
- Coarse to Fine Search
- Bayesian Search
- Genetic Algorithm

Manual Search/Hand Tuning

In this search, the user himself manually tweak the hyperparameter combinations until the model gets the optimal performance.

- Guess some parameter values based on past experience,
- Train a model, measure its performance on the validation data,
- Analyze the results, and use your intuition to suggest new parameter values.
- Repeat until you have satisfactory performance (or you run out of time, computing budget, or patience).

Pros

- For a skilled practitioner, this can help to reduce computational time

Cons

- Hard to guess even though you really understand the algorithm
- Time-consuming

Grid Search

If there are only a few hyperparameters, each with a small number of possible values, then a more systematic approach called ***grid search*** is appropriate.

- Try all combinations of values and see which performs best on the validation data.
- Different combinations can be run in parallel on different machines, so if you have sufficient computing resources, this need not be slow.
- Although in some cases model selection has been known to suck up resources on thousand-computer clusters for days at a time.
- if two hyperparameters are independent of each other, they can be optimized separately.

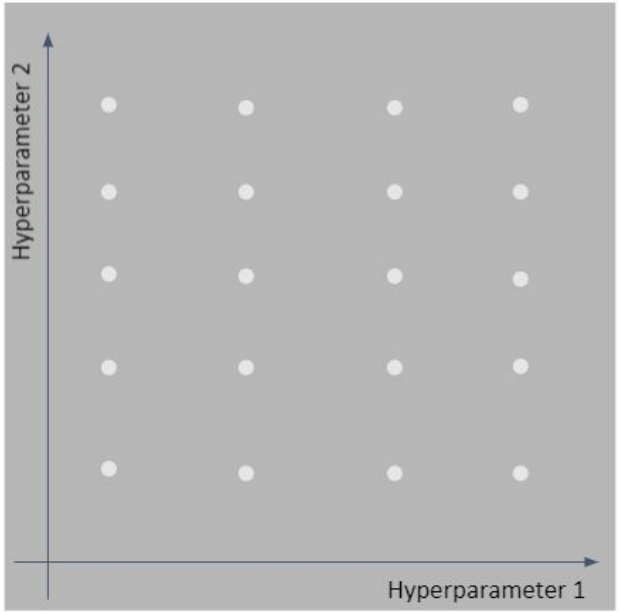


Image Source: A Medium Blog by Louis Owen

Grid Search

We simply run a random forest classifier with default values and get the predictions for the test set.

Example:

```
# Instantiate and fit random forest classifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Predict on the test set and call accuracy
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(accuracy)
```

0.81

Grid Search

Grid Search starts with defining a *search space grid*.

The grid consists of selected hyperparameter names and values, and grid search exhaustively searches the best combination of these given values.

Example:

```
# Define the parameter grid
param_grid = {
'n_estimators': [50, 100, 200, 300],
'min_samples_leaf': [1, 5, 10],
'max_depth': [2, 4, 6, 8, 10],
'max_features': ['auto', 'sqrt'],
'bootstrap': [True, False]}

# Instantiate GridSearchCV
model_gridsearch = GridSearchCV(
      estimator=rf_model,
      param_grid=param_grid,
      scoring='accuracy',
      n_jobs=4,
      cv=5,
      refit=True,
      return_train_score=True)
```

Grid search will have to run and compare 240 models (=4*5*3*2*2).

For 5-fold cross-validation, grid search will have to evaluate 1200 (=240*5) model performances.

Grid Search

```
# Record the current time
start = time()

# Fit the selected model
model_gridsearch.fit(X_train, y_train)

# Print the time spend and number of models ran
print("GridSearchCV took %.2f seconds for %d candidate parameter settings." % ((time() -
start), len(model_gridsearch.cv_results_['params'])))

# Predict on the test set and call accuracy
y_pred_grid = model_gridsearch.predict(X_test)
accuracy_grid = accuracy_score(y_test, y_pred_grid)
```

GridSearchCV took 247.79 seconds for 240 candidate parameter settings.

0.88

Random Search

In random search, we define distributions for each hyperparameter which can be defined *uniformly* or with a *sampling method*.

For example, if there are 500 values in the distribution and if we input `n_iter=50` then random search will randomly sample 50 values to test.

Since random search does not try every hyperparameter combination, it does not necessarily return the best performing values, but it returns a relatively good performing model in a significantly shorter time.

Random Search

In random search, we define distributions for each hyperparameter which can be defined *uniformly* or with a *sampling method.*

```
# specify distributions to sample from
param_dist = {
      'n_estimators': list(range(50, 300, 10)),
      'min_samples_leaf': list(range(1, 50)),
      'max_depth': list(range(2, 20)),
      'max_features': ['auto', 'sqrt'],
      'bootstrap': [True, False]}

# specify number of search iterations
n_iter = 50

# Instantiate RandomSearchCV
model_random_search = RandomizedSearchCV(
      estimator=rf_model,
      param_distributions=param_dist,
      n_iter=n_iter)
```

Random Search

```
# Record the current time
start = time()

# Fit the selected model
model_random_search.fit(X_train, y_train)

# Print the time spend and number of models ran
print("RandomizedSearchCV took %.2f seconds for %d candidate parameter settings." %
((time() - start), len(model_random_search.cv_results_['params'])))

# Predict on the test set and call accuracy
y_pred_random = model_random_search.predict(X_test)
accuracy_random = accuracy_score(y_test, y_pred_random)
```

RandomizedSearchCV took 64.17 seconds for 50 candidate parameter settings.

0.86

Coarse to Fine Search

For Grid Search, an increased number of hyperparameters easily becomes a bottleneck. To prevent this inefficiency, we can combine grid search with random search.

- In coarse-to-fine tuning, we start with a random search to find the promising value ranges for each hyperparameter.
- After getting focus area for each hyperparameter using random search, we can define the grid accordingly for grid search to find the best values amongst them.
- For example, if the random search returns high performance for n_estimators between 150 and 200, this is the range we want grid search to focus on.

Coarse to Fine Search

For Grid Search, an increased number of hyperparameters easily becomes a bottleneck. To prevent this inefficiency, we can combine grid search with random search.

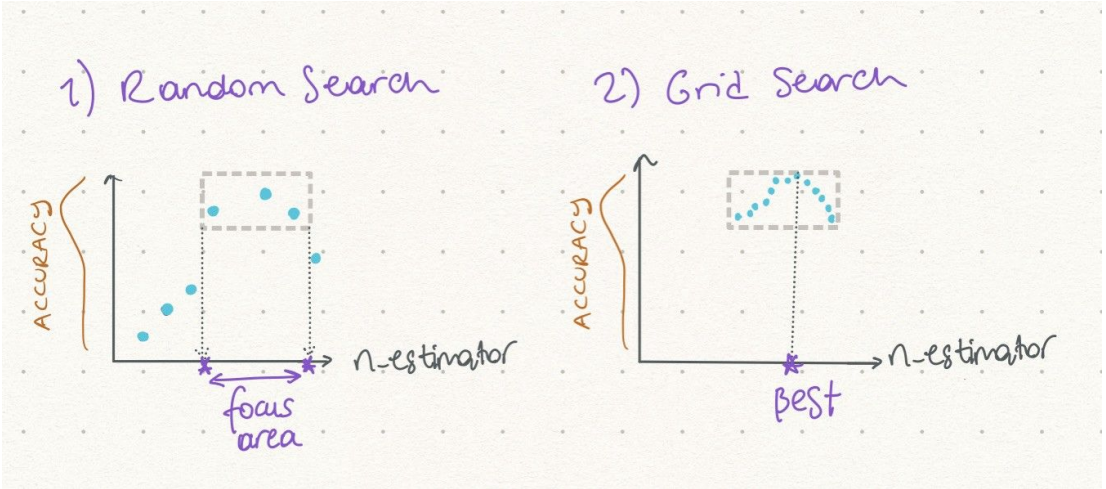


Image Source: A Medium Blog by Idil Ismiguzel

When to Use What

When we are training a deep neural network with huge hyperparameter space, it is preferable to use a Manual Search or Random Search method rather than using the Grid Search method.

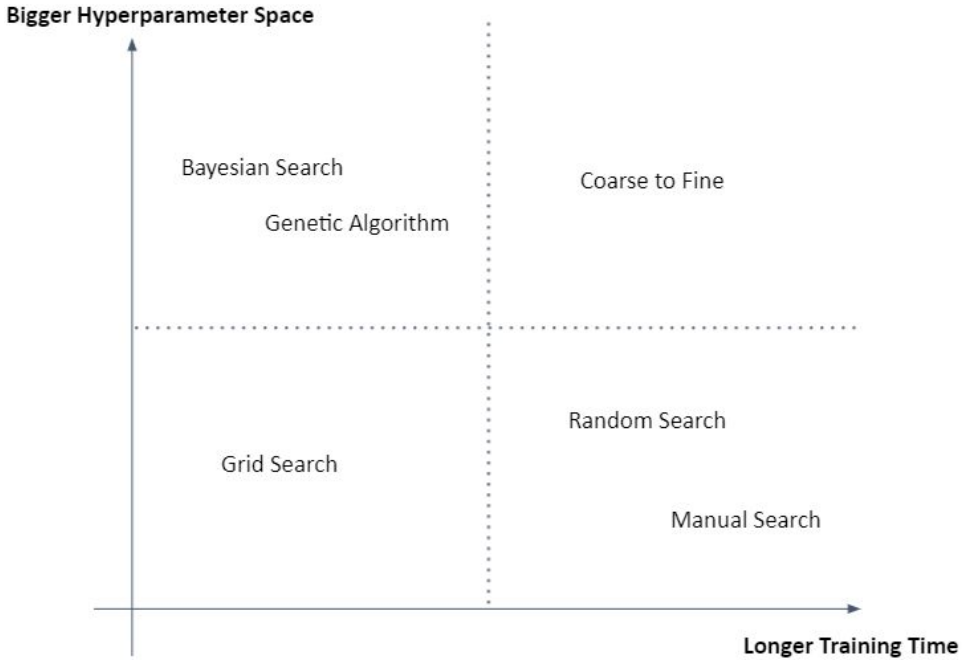


Image Source: A Medium Blog by Louis Owen

Next lecture

Optimization - II

11th September 2023