FACULTY OF INFORMATION TECHNOLOGY

Machine Learning

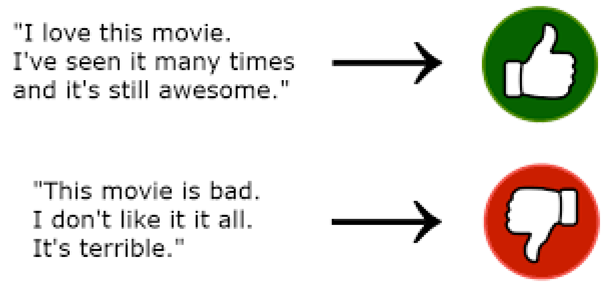
(Học Máy)

Lab 7

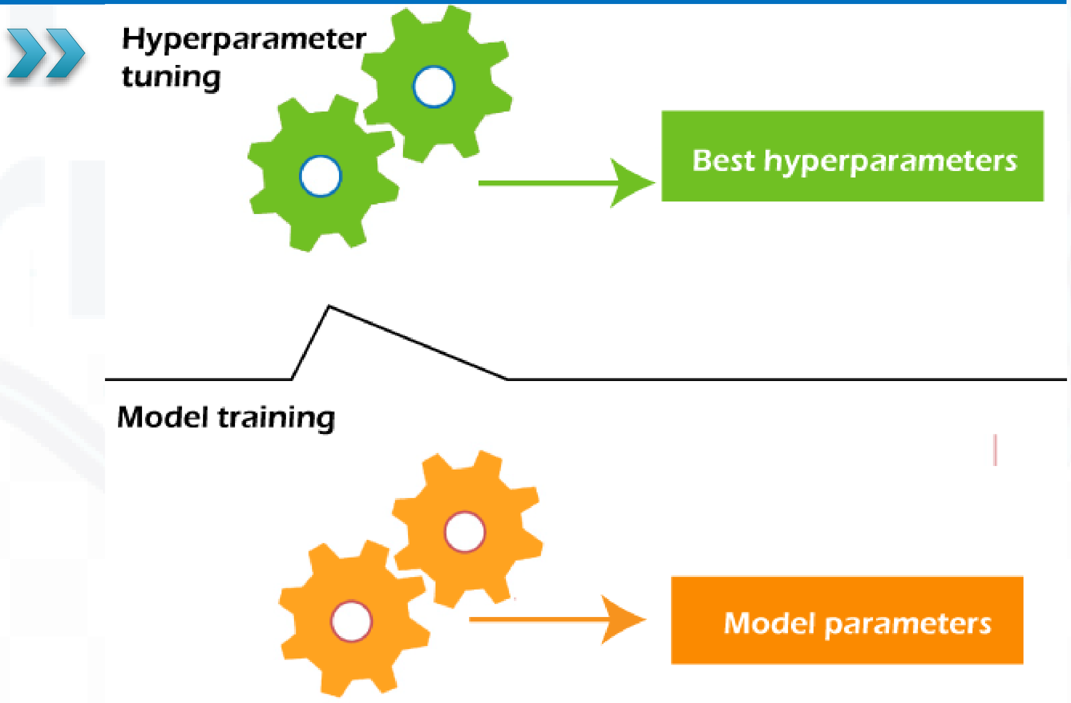
Semester 2, 2023/2024

Content

- What is a parameter?
- What is a hyperparameter?
- Tuning hyperparameters
  - GridSearchCV object
  - Steps in a grid search
  - GridSearchCV usage
- Movie reviews sentiment
  - Dataset
  - Vectorization
  - Building classification models



8.1 Parameters



What are parameters?

- Components of the model learned during the modeling process.

- You do not set these manually (you can't in fact!)

- The algorithm will discover these for you

Parameters in Logistic Regression

- A simple logistic regression model:

```
log_reg_clf = LogisticRegression()
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... ]])
```

Parameters in Random forest

- What about tree-based algorithms?
- Random forest has no coefficients, but node decisions (what feature and what value to split on).

```
# A simple random forest estimator
rf_clf = RandomForestClassifier(max_depth=2)
rf_clf.fit(X_train, y_train)

# Pull out one tree from the forest
chosen_tree = rf_clf.estimators_[7]
```

Where to find parameters?

- To find parameters we need:
  - To know a bit about the algorithm
  - Consult the Scikit Learn documentation
- Parameters will be found under the '**Attributes**' section, not the '**Parameters**' section.

Where to find parameters?

- Parameters will be found under the '**Attributes**' section, not the '**Parameters**' section.

Parameters:

n_estimators : *int, default=100*

The number of trees in the forest.

> *Changed in version 0.22:* The default value of `n_estimators` changed from 10 to 100 in 0.22.

criterion : *{"gini", "entropy", "log_loss"}, default="gini"*

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy" both for the Shannon information gain, see Mathematical formulation. Note: This parameter is tree-specific.

max_depth : *int, default=None*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*

The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Where to find parameters?

- Parameters will be found under the '**Attributes**' section, not the '**Parameters**' section.

Attributes:

estimator_ : *DecisionTreeClassifier*

The child estimator template used to create the collection of fitted sub-estimators.

New in version 1.2: `base_estimator_` was renamed to `estimator_`.

`base_estimator_` : *DecisionTreeClassifier*

Estimator used to grow the ensemble.

estimators_ : *list of DecisionTreeClassifier*

The collection of fitted sub-estimators.

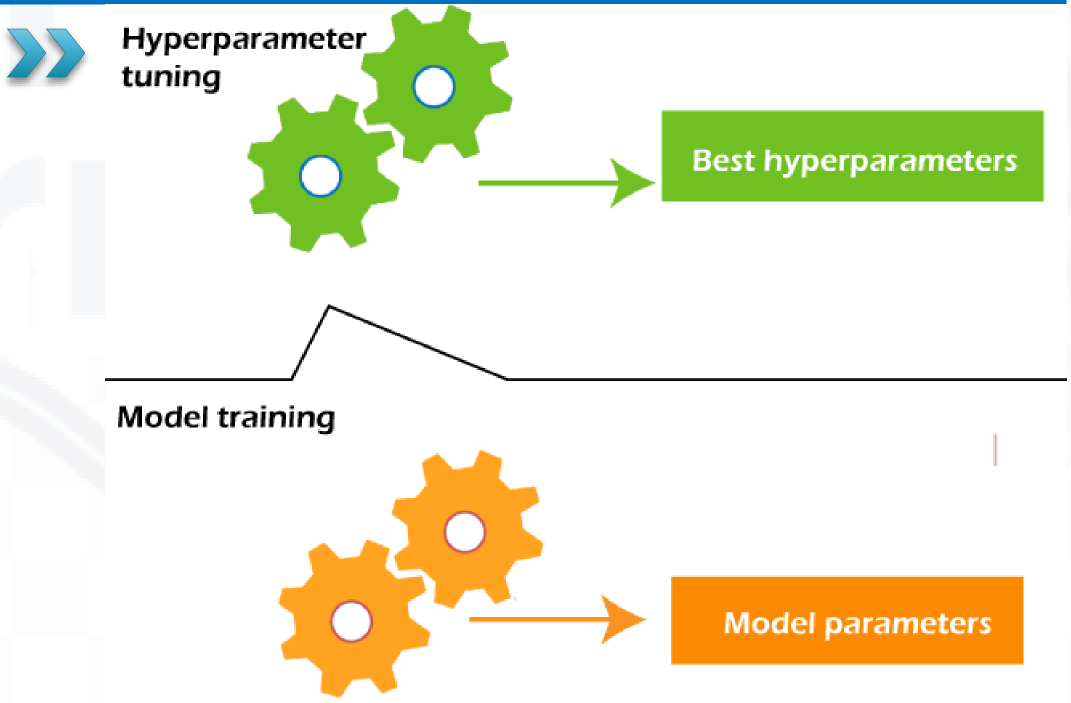
classes_ : *ndarray of shape (n_classes,) or a list of such arrays*

The classes labels (single output problem), or a list of arrays of class labels (multi-output problem).

n_classes_ : *int or list*

The number of classes (single output problem), or a list containing the number of classes for each output (multi-output problem).

8.2 Hyper-parameters



What are hyperparameters?

- Something you set before the modeling process (like knobs on an old radio)
- You also '**tune**' your hyperparameters!
- The algorithm does not learn these

How to find hyperparameters that matter?

- Some resources for learning this:
  - Academic papers
  - Blogs and tutorials from trusted sources
  - The Scikit Learn module documentation
  - Experience

Hyperparameter Importance

- Some hyperparameters are more important than others.
- Some will not help model performance (For the random forest classifier: **n_jobs**, **random_state**, **verbose**)
- Not all hyperparameters make sense to **'train'**

Random Forest: Important Hyperparameters

- Some important hyperparameters:
  - n_estimators (high value)
  - max_features (try different values)
  - max_depth & min_sample_leaf (important for overfitting)
  - criterion (maybe)

Hyperparameter Values

- Some hyperparameters are more important than others to begin tuning.

- But which values to try for hyperparameters?

  - Specific to each algorithm & hyperparameter

  - Some best practice guidelines & tips do exist

Conflicting Hyperparameter Choices

- Be aware of **conflicting hyperparameter** choices.
  - **LogisticRegression**() conflicting parameter options of solver & penalty that conflict
  - The 'newton-cg', 'sag' and 'lbfgs' solvers support only L2 penalties.

Silly Hyperparameter Values

Be aware of setting 'silly' values for different algorithms:

- Random forest with low number of trees
  - Would you consider it a 'forest' with only 2 trees?
- 1 Neighbor in KNN algorithm
  - Averaging the 'votes' of one person doesn't sound very robust!
- Increasing a hyperparameter by a very small amount

Spending time documenting sensible values for hyperparameters is a valuable activity

Manual Hyperparameter Choice

- In the previous labs, we built models as:

```
knn_5 =  KNeighborsClassifier(n_neighbors=5)
knn_10 = KNeighborsClassifier(n_neighbors=10)
knn_20  = KNeighborsClassifier(n_neighbors=20)
```

- This is quite inefficient. Can we do better?

Automating Hyperparameter Tuning

- Try a for loop to iterate through options:

```
neighbors_list = [3,5,10,20,50,75]

for test_number in neighbors_list:
    model = KNeighborsClassifier(n_neighbors=test_number)
    predictions = model.fit(X_train, y_train).predict(X_test)

    accuracy = accuracy_score(y_test, predictions)
    accuracy_list.append(accuracy)
```

Automating Hyperparameter Tuning

- We can store the results in a DataFrame to view:

```
results_df = pd.DataFrame({'neighbors':neighbors_list, 'accuracy':accuracy_list})
print(results_df)
```

Neighbors	3	5	10	20	50	75
Accuracy	0.71	0.7125	0.765	0.7825	0.7825	0.7825

8.3 Tunning Hyperparameters

GridSearchCV Object

- Introducing a **GridSearchCV** object:

```
sklearn.model_selection.GridSearchCV(
    estimator,
    param_grid, scoring=None, fit_params=None,
    n_jobs=None, iid='warn', refit=True, cv='warn',
    verbose=0, pre_dispatch='2*n_jobs',
    error_score='raise-deprecating',
    return_train_score='warn')
```

Steps in a Grid Search

- An algorithm to tune the hyperparameters. (Sometimes called an 'estimator')
- Defing which hyperparameters we will tune
- Define a range of values for each hyperparameter
- Setting a cross-validation scheme; and
- Define a score function so we can decide which square on our grid was 'the best'.
- Include extra useful information or functions

HYPERPARAMETER TUNING IN PYTHON
GridSearchCV Object Inputs

The important inputs are:

- Estimator:
  - Essentially our algorithm
  - Algorithm can be kNN, Decision Tree, SVM, …
- Remember:
  - Only one estimator per **GridSearchCV** object

HYPERPARAMETER TUNING IN PYTHON
GridSearchCV Object Inputs

The important inputs are:

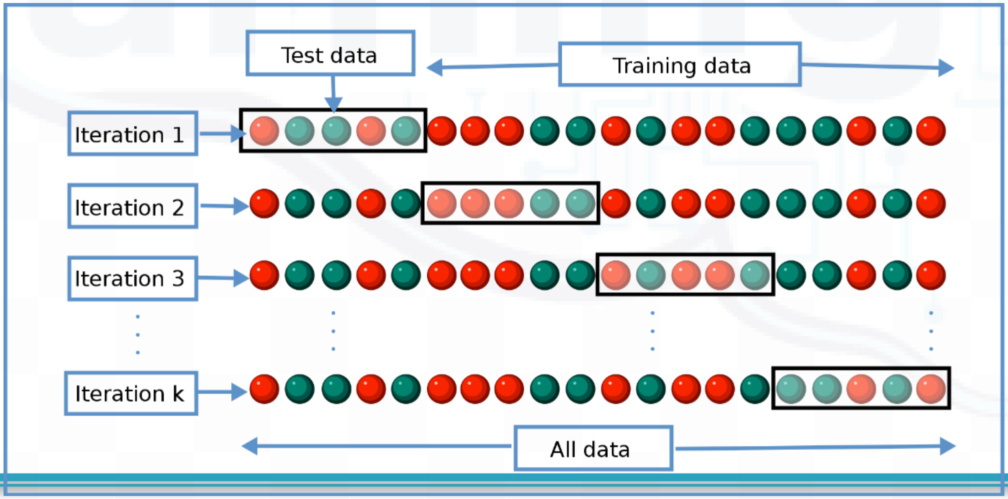
- **param_grid**:
  - Setting which hyperparameters and values to test
  - The keys in your param_grid dictionary must be valid hyperparameters.

```
param_grid = {'max_depth': [2, 4, 6, 8],
              'min_samples_leaf': [1, 2, 4, 6]}
```

HYPERPARAMETER TUNING IN PYTHON
GridSearchCV Object Inputs

The important inputs are:

- **cv**:
  - Choice of how to undertake cross-validation
  - Using an integer undertakes k-fold cross validation where 5 or 10 is usually standard



HYPERPARAMETER TUNING IN PYTHON
GridSearchCV Object Inputs

The important inputs are:

- scoring:
  - Which score to use to choose the best grid square (model)
  - Use your own or Scikit Learn's metrics module
- We can check all the built in scoring functions this way:

```
from sklearn import metrics
sorted(metrics.SCORERS.keys())
```

HYPERPARAMETER TUNING IN PYTHON
GridSearchCV Object Inputs

The important inputs are:

- refit:
  - Fits the best hyperparameters to the training data
  - Allows the **GridSearchCV** object to be used as an estimator (for prediction)
  - A very handy option!

HYPERPARAMETER TUNING IN PYTHON
GridSearchCV Object Inputs

The important inputs are:

- n_jobs:
  - Assists with parallel execution
  - Allows multiple models to be created at the same time, rather than one after the other
- Some handy code:

```
import os
print(os.cpu_count())
```

Careful using all your cores for modelling if you want to do other work!

HYPERPARAMETER TUNING IN PYTHON
GridSearchCV Object Inputs

The important inputs are:

- return_train_score:
  - Logs statistics about the training runs that were undertaken
  - Useful for analyzing bias-variance trade-off but adds computational expense.
  - Does not assist in picking the best model, only for analysis purposes

Building a GridSearchCV object

- Building our own GridSearchCV Object:

```
# Create the grid
param_grid = {'max_depth': [2, 4, 6, 8], 'min_samples_leaf': [1, 2, 4, 6]}


#Get a base classifier with some set parameters.
rf_class = RandomForestClassifier(criterion='entropy', max_features='auto')
```

Building a GridSearchCV object

- Putting the pieces together:

```
grid_rf_class = GridSearchCV(
    estimator = rf_class,
    param_grid = parameter_grid,
    scoring='accuracy',
    n_jobs=4,
    cv = 10,
    refit=True,
    return_train_score=True)
```

Using a GridSearchCV Object

- Since we set refit to True we can directly use the object:

```
#Fit the object to our data
grid_rf_class.fit(X_train, y_train)

# Make predictions
grid_rf_class.predict(X_test)
```

Analyzing the output

- Three different groups for the GridSearchCV properties

- A results log
  - `cv_results_`
- The best results
  - `best_index_` , `best_params_` & `best_index_`
- 'Extra information'
  - `scorer_` , `n_splits_` & `refit_time_`

Accessing object properties

- Properties are accessed using the dot notation.

- For example:

grid_search_object.property

- Where property is the actual property we want to retrieve

The '.cv_results_' property

- Read the property into a DataFrame to print and analyze

```
cv_results_df = pd.DataFrame(grid_rf_class.cv_results_)

print(cv_results_df.shape)
```

```
(12, 23)
```

- The 12 rows for the 12 squares in our grid or 12 models we ran

The best grid square

- Information on the best grid square is neatly summarized in the following three properties:
  - best_params_ , the dictionary of parameters that gave the best score.
  - best_score_ , the actual best score.
  - best_index , the row in our cv_results_.rank_test_score that was the best.

The `best_estimator_` property

- The best_estimator_ property is an estimator built using the best parameters from the grid search

```
type(grid_rf_class.best_estimator_)
```

```
sklearn.ensemble.forest.RandomForestClassifier
```

```
print(grid_rf_class.best_estimator_)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=10, max_features='auto', max_leaf_nodes=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=300, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

Extra information

- Some extra information is available in the following properties:
- scorer_
  - What scorer function was used on the held-out data.
- n_splits_
  - How many cross-validation splits.
- refit_time_
  - The number of seconds used for refitting the best model on the whole dataset

Grid Search Pros & Cons

- Advantages:
  - You don’t have to write thousands of lines of code
  - Finds the best model within the grid
  - Easy to explain

- Disadvantages:
  - Computationally expensive! Remember how quickly we made 6,000+ models?
  - It is 'uninformed’. The results of one model don't help create the next model.

8.4 Movie reviews sentiment

The movie reviews dataset

- The dataset consists of 2000 user-created movie reviews archived on the IMDb (Internet Movie Database)
- The reviews are equally partitioned into a **positive set** and a **negative set** (1000+1000).
- Each review consists of a plain text file (.txt) and a class label representing the overall user opinion.
- The class attribute has only two values: pos (positive) or neg (negative).

Importing libraries

- Necessary libraries for sentiment movie reviews dataset:

```
import nltk, random
nltk.download('movie_reviews')#download movie reviews dataset
from nltk.corpus import movie_reviews
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.model_selection import train_test_split
```

Getting movie reviews information

- Movie reviews information:
  - Fileids
  - Categories
  - Words
  - ...

```
print(len(movie_reviews.fileids()))
print(movie_reviews.categories())
print(movie_reviews.words()[:100])
print(movie_reviews.fileids()[:10])
```

```
2000
['neg', 'pos']
['plot', ':', 'two', 'teen', 'couples', 'go', 'to', ...]
['neg/cv000_29416.txt', 'neg/cv001_19502.txt', 'neg/cv002_17424.txt',
```

Create dataset from movie reviews

- Load words and categories into the **document**:

```
[ ] documents = [(list(movie_reviews.words(fileid)), category)
                 for category in movie_reviews.categories()
                 for fileid in movie_reviews.fileids(category)]
    random.seed(123)
    random.shuffle(documents)
```

```
print('Number of Reviews/Documents: {}'.format(len(documents)))
print('Corpus Size (words): {}'.format(np.sum([len(d) for (d,l) in documents])))
print('Sample Text of Doc 1:')
print('-'*30)
print(' '.join(documents[0][0][:50])) # first 50 words of the first document
```

```
Number of Reviews/Documents: 2000
Corpus Size (words): 1583820
Sample Text of Doc 1:
------------------------------
most movies seem to release a third movie just so it can be called a trilogy .
```

Train-Test Split

- Split the document into train test split:

```
[ ]  train, test = train_test_split(documents, test_size = 0.33, random_state=42)
```

```
[ ]  ## Sentiment Distrubtion for Train and Test
     print(Counter([label for (words, label) in train]))
     print(Counter([label for (words, label) in test]))
```

```
Counter({'neg': 674, 'pos': 666})
Counter({'pos': 334, 'neg': 326})
```

Determine X_train, X_test, ...

- Split our training data into **X_train** and **y_train** as the features (X) and labels (y) in training.

- Likewise, we split our testing data into **X_test** and **y_test** as the features (X) and labels (y) in testing.

```
[ ]  X_train = [' '.join(words) for (words, label) in train]
     X_test = [' '.join(words) for (words, label) in test]
     y_train = [label for (words, label) in train]
     y_test = [label for (words, label) in test]
```

Text Vectorization

- In feature-based machine learning, we need to vectorize texts into feature sets (i.e., feature engineering on texts).

```
[ ]  from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

     tfidf_vec = TfidfVectorizer(min_df = 10, token_pattern = r'[a-zA-Z]+')
     X_train_bow = tfidf_vec.fit_transform(X_train) # fit train
     X_test_bow = tfidf_vec.transform(X_test) # transform test
```

Model Selection

- Apply classification algorithms to sentiment movie reviews
  - SVM, Decision Tree, Naive Bayes, Logistic Regression

shutterstock.com · 2042284517

FACULTY OF INFORMATION TECHNOLOGY