SVM Hyperparameter Tuning using GridSearchCV | ML

A Machine Learning model is defined as a [illegible] parameters that need to be learned from the data. However, there are some parameters, [illegible] cannot be directly learned. They are commonly chosen by human based on some intuition or hit and trial before the actual training begins. These parameters exhibits their importance by improving performance of the model such as its complexity or its learning rate. Models can have many hyper-parameters and finding the best combination of parameters can be treated as a search problem.

SVM also has some hyper-parameters (like what C or gamma values to use) and finding optimal hyper-parameter is a very hard task to solve. But it can be found by just trying all combinations and see what parameters work best. The main idea behind it is to create a grid of hyper-parameters and just try all of their combinations (hence, this method is called **Gridsearch**, But don't worry! we don't have to do it manually because Scikit-learn has this functionality built-in with GridSearchCV.

`GridSearchCV` takes a dictionary that describes the parameters that could be tried on a model to train it. The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

This article demonstrates how to use **GridSearchCV** searching method to find optimal hyper-parameters and hence improve the accuracy/prediction results

Import necessary libraries and get the Data –

We'll use the built-in breast cancer dataset from Scikit Learn. We can get with the load function:

```
import pandas as pd
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC

cancer = load_breast_cancer()

# The data set is presented in a dictionary form:
print(cancer.keys())
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

Now we will extract all features into the new dataframe and our target features into separate dataframe.

```
df_target = pd.DataFrame(cancer['target'],
                         columns =['Cancer'])

print("Feature Variables: ")
print(df_feat.info())
```

```
Feature Variables:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
mean radius                 569 non-null float64
mean texture                569 non-null float64
mean perimeter              569 non-null float64
mean area                   569 non-null float64
mean smoothness             569 non-null float64
mean compactness            569 non-null float64
mean concavity              569 non-null float64
mean concave points         569 non-null float64
mean symmetry               569 non-null float64
mean fractal dimension      569 non-null float64
radius error                569 non-null float64
texture error               569 non-null float64
perimeter error             569 non-null float64
area error                  569 non-null float64
smoothness error            569 non-null float64
compactness error           569 non-null float64
concavity error             569 non-null float64
concave points error        569 non-null float64
symmetry error              569 non-null float64
fractal dimension error     569 non-null float64
worst radius                569 non-null float64
worst texture               569 non-null float64
worst perimeter             569 non-null float64
worst area                  569 non-null float64
worst smoothness            569 non-null float64
worst compactness           569 non-null float64
worst concavity             569 non-null float64
worst concave points        569 non-null float64
worst symmetry              569 non-null float64
worst fractal dimension     569 non-null float64
dtypes: float64(30)
memory usage: 133.4 KB
None
```

```
print("Dataframe looks like : ")
print(df_feat.head())
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	wors smoothnes
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.162
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.123
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.144
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.209
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.137

5 rows × 30 columns

Train Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
                        df_feat, np.ravel(df_target),
                test_size = 0.30, random_state = 101)
```

Train the Support Vector Classifier without Hyper-parameter Tuning –

First, we will train our model by calling standard SVC() function without doing Hyper-parameter Tuning and see its classification and confusion matrix.

```
# train the model on train set
model = SVC()
model.fit(X_train, y_train)

# print prediction results
predictions = model.predict(X_test)
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        66
           1       0.61      1.00      0.76       105

    accuracy                           0.61       171
   macro avg       0.31      0.50      0.38       171
weighted avg       0.38      0.61      0.47       171
```

We got 61 % accuracy but did you notice something strange ?
Notice that recall and precision for class 0 are always 0. It means that classifier is always classifying everything into a single class i.e class 1! This means our model needs to have its parameters tuned.

Here is when the usefulness of GridSearch comes into picture. We can search for parameters using a GridSearch!

Use GridsearchCV

One of the great things about GridSearchCV is that it is a meta-estimator. It takes an estimator like SVC, and creates a new estimator, that behaves exactly the same – in this case, like a classifier. You should add refit=True and choose verbose to whatever number you want, higher the number, the more verbose (verbose just means the text output describing the process).

```
from sklearn.model_selection import GridSearchCV
```

```
grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(X_train, y_train)
```

What **fit** does is a bit more involved then usual. First, it runs the same loop with cross-validation, to find the best parameter combination. Once it has the best combination, it runs fit again on all data passed to fit (without cross-validation), to built a single new model using the best parameter setting.

You can inspect the best parameters found by GridSearchCV in the best_params_ attribute, and the best estimator in the best_estimator_ attribute:

```
# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
```

```
{'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.0001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Then you can re-run predictions and see classification report on this grid object just like you would with a normal model.

```
grid_predictions = grid.predict(X_test)

# print classification report
print(classification_report(y_test, grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.91      0.93        66
           1       0.94      0.97      0.96       105

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171
```

We have got almost **95 % prediction** result.

tyagikartik4282
Check out this Author's contributed articles.

If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please Improve this article if you find anything incorrect by clicking on the "Improve Article" button below.

Article Tags : Machine Learning Python

Practice Tags : Machine Learning


1

0

To-do Done

No votes yet.

Feedback/ Suggest Improvement | Add Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe Add Disqus to your siteAdd DisqusAdd