Hyperparameter tuning using GridSearchCV and KerasClassifier

Hyperparameter tuning is done to increase the efficiency of a model by tuning the parameters of the neural network. Some scikit-learn APIs like GridSearchCV and RandomizedSearchCV are used to perform hyper parameter tuning.

In this article, you'll learn how to use GridSearchCV to tune Keras Neural Networks hyper parameters.

Approach:

1. We will wrap Keras models for use in scikit-learn using KerasClassifier which is a wrapper.
2. We will use cross validation using KerasClassifier and GridSearchCV
3. Tune hyperparameters like number of epochs, number of neurons and batch size.

Implementation of the scikit-learn classifier API for Keras:

```
tf.keras.wrappers.scikit_learn.KerasClassifier(

  build_fn=None, **sk_params

)
```

Code:

```
import  tensorflow as tf

import  pandas as pd

from  sklearn.compose  import  ColumnTransformer

from  sklearn.preprocessing  import  OneHotEncoder

from  keras.wrappers.scikit_learn  import  KerasClassifier

from  sklearn.model_selection  import  GridSearchCV

from  sklearn.preprocessing  import  LabelEncoder

from  sklearn.preprocessing  import  StandardScaler
```

Import the dataset using which we'll predict if a customer stays or leave.

Code:

```
dataset = pd.read_csv(``'Churn_Modelling.csv'``)

X = dataset.iloc[:, 3``:``-``1``].values

y = dataset.iloc[:, -``1``].values
```

Code: Preprocess the data

```
le = LabelEncoder()

X[:, 2``] = le.fit_transform(X[:, 2``])

ct = ColumnTransformer(transformers``=``[(``'encoder'``, OneHotEncoder(), [``1``])],
remainder``=``'passthrough'``)

X = np.array(ct.fit_transform(X))

sc = StandardScaler()

X = sc.fit_transform(X)
```

To use the KerasClassifier wrapper, we will need to build our model in a function which needs to be passed to the *build_fn* argument in the KerasClassifier constructor.

Code:

```
def build_clf(unit):

  ann = tf.keras.models.Sequential()

  ann.add(tf.keras.layers.Dense(units``=``unit, activation``=``'relu'``))

  ann.add(tf.keras.layers.Dense(units``=``unit, activation``=``'relu'``))

  ann.add(tf.keras.layers.Dense(units``=``1``, activation``=``'sigmoid'``))

  ann.``compile``(optimizer = 'adam'``, loss = 'binary_crossentropy'``, metrics =
[``'accuracy'``])

  return ann
```

Code: create the object of KerasClassifier class

```
model``=``KerasClassifier(build_fn``=``build_clf)
```

Now we will create the dictionary of the parameters we want to tune and pass as an argument in GridSearchCV.

Code:

```
params``=``{``'batch_size'``:[``100``,  20``,  50``,  25``,  32``],

     'nb_epoch'``:[``200``,  100``,  300``,  400``],

     'unit'``:[``5``,``6``,  10``,  11``,  12``,  15``],

     }

gs``=``GridSearchCV(estimator``=``model, param_grid``=``params, cv``=``10``)

gs  =  gs.fit(X, y)
```

The *best_score_* member gives the best score observed during the optimization procedure and the *best_params_* describes the combination of parameters that achieved the best results.

Code:

```
best_params``=``gs.best_params_

accuracy``=``gs.best_score_
```

Output:

> Accuracy: 0.80325
>
> Best Params: {'batch_size': 20, 'nb_epoch': 200, 'unit': 15}