How to use TPOT to discover top-performing models for classification tasks.

TPOT for Classification

In this section, we will use TPOT to discover a model for the iris dataset.The iris dataset is a standard machine learning dataset comprising 150 rows of data with 4 numerical input variables and a target variable with three class values e.g. ['setosa', 'versicolor', 'virginica']

Using a test harness of repeated stratified 8-fold cross-validation with three repeats, a naive model can achieve an accuracy of about 78.4 percent. A top-performing model can achieve accuracy on this same test harness of about 98.3 percent. This provides the bounds of expected performance on this dataset.

Load the dataset and summarize like below:

```
from sklearn.datasets import load_iris

data=load_iris()

x=data.data

y=data.target

data.feature_names
```

Output :

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
data.target_names
```

Output :

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
data.DESCR
```

Output :

```
'.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n
:Number of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 numeric, predic
tive attributes and the class\n    :Attribute Information:\n        - sepal length in cm\n        - s
epal width in cm\n        - petal length in cm\n        - petal width in cm\n        - class:\n
- Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virginica\n
```

First, let's use naive_bayes model for the iris dataset.

```
Import required libraries:

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

xtest,xtrain,ytest,ytrain=train_test_split(x,y,test_size=0.20,r
andom_state=32)

model.fit(xtest,ytest)

y_predicted=model.predict(xtrain)

accuracy_score(ytrain,y_predicted)*100
```

Output :

78.4

Next, let's use TPOT to find a good model for the iris dataset.We will use a population size of 100 for 8 generations for the search and use all cores on the system by setting "*n_jobs*" to -1,scoring to 'accuracy and verbosity to 2.

```
from tpot import TPOTClassifier

# define the search

model = TPOTClassifier(generations=8,
population_size=100,scoring='accuracy', verbosity=2,n_jobs=-1)

Finally, we can start the search and ensure that the
best-performing model is saved at the end of the execution.

# perform the search

model.fit(xtest,ytest)

# export the best model

model.export('iris_best_classifier.py')
```

The accuracy of top-performing models will be reported along the way.

```
Generation 1 - Current best internal CV score: 0.975

Generation 2 - Current best internal CV score: 0.9833333333333334

Generation 3 - Current best internal CV score: 0.9833333333333334

Generation 4 - Current best internal CV score: 0.9833333333333334

Generation 5 - Current best internal CV score: 0.9833333333333334

Generation 6 - Current best internal CV score: 0.9833333333333334

Generation 7 - Current best internal CV score: 0.9833333333333334

Generation 8 - Current best internal CV score: 0.9833333333333334

Best pipeline: ExtraTreesClassifier(Nystroem(input_matrix, gamma=0.7000000000000001, kernel=additive_chi2, n_components=4), boo
tstrap=False, criterion=entropy, max_features=0.6500000000000001, min_samples_leaf=7, min_samples_split=12, n_estimators=100)

: TPOTClassifier(generations=8, n_jobs=-1, scoring='accuracy', verbosity=2)
```

Running the model may take a few minutes, and you will see a progress bar.

In this case, we can see that the top-performing pipeline achieved the mean accuracy of about 98.3 percent. This is a best performing model.The top-performing model saved to a file named “iris_best_classifier.py“.

By opening this file, you can see that there is some generic code for loading a dataset and fitting the pipeline.

```
import numpy as np

import pandas as pd

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.kernel_approximation import Nystroem

from sklearn.model_selection import train_test_split

from sklearn.pipeline import make_pipeline



# NOTE: Make sure that the outcome column is labeled 'target'
in the data file

tpot_data = pd.read_csv('PATH/TO/DATA/FILE',
sep='COLUMN_SEPARATOR', dtype=np.float64)

features = tpot_data.drop('target', axis=1)

training_features, testing_features, training_target,
testing_target = \

            train_test_split(features, tpot_data['target'],
random_state=None)



# Average CV score on the training set was: 0.9833333333333334

exported_pipeline = make_pipeline(
Nystroem(gamma=0.7000000000000001, kernel="additive_chi2",
n_components=4),  ExtraTreesClassifier(bootstrap=False,
```

```
criterion="entropy", max_features=0.6500000000000001,
min_samples_leaf=7, min_samples_split=12, n_estimators=100))

exported_pipeline.fit(training_features, training_target)

results = exported_pipeline.predict(testing_features)
```

We can use this code to fit a final model on all available data and make a prediction for new data.