

Week 3 – HomeWork 2

Paritosh Rai



7335 Machine Learning 2



HW 2

This is called DeathToGridSearch because with this example you will never have to think about how to manage a large number of classifiers etc simultaneously.  You will now be able to run and collect results in a very straightforward manner.  #LongLongLiveGridSearch!

PS C:\Users\parit> conda activate ML1

PS C:\Users\parit> & C:/Users/parit/Anaconda/envs/ML1/python.exe

Python 3.7.1 (default, Oct 28 2018, 08:39:03) [MSC v.1912 64 bit (AMD64)] :: Anaconda, Inc. on win32

Type "help", "copyright", "credits" or "license" for more information.

# Homework 2

import numpy as np

from sklearn.metrics import accuracy\_score # other metrics too pls!

from sklearn.ensemble import RandomForestClassifier # more!

from sklearn.model\_selection import KFold

# adapt this code below to run your analysis

**1. Write a function to take a list or dictionary of clfs and hypers (i.e., use logistic regression), each with 3** different sets of hyper parameters for each.

Details in enclosed .py file. Followed following steps:

Step 1: Created a parameter dictionary for KNeighborsClassifier, DecisionTreeClassifier and Logistic Regression with more than three parameters with different set of parameters values.

Step 2: Build the function to pull 1st three parameters.

Step 3: Run the function with three classifiers.

**2. Expand to include larger number of classifiers and hyperparameter settings.**

# create two more classifier

# create dictionary for SVC

Details in enclosed .py file. Followed following steps:

Step1: Created two more Classifier SVC and RidgeClassifier with more than 3 parameters.

Step2: Created list containing five (5) classifiers abbreviated as 'DT\_parm', 'LR\_parm', 'KNN\_parm', 'SV\_parm','RC\_Parm'.

Step3: Created a clfs function to pull the classifier dictionary using clfs directory.

**3. Find some simple data**

Step 1: Selected iris data from sklearn.datasets

Step 2: Listed keys:

>>> iris.keys()

dict\_keys(['data', 'target', 'target\_names', 'DESCR', 'feature\_names', 'filename'])

>>>

Step 3: Listed targets:

>>> print (iris.target)

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2]

>>>

Step 4: Listed Target values:

>>> print (iris.target\_names)

['setosa' 'versicolor' 'virginica']

>>>

Step 5: Data shape (number of samples and number of features)

>>>

>>> n\_samples, n\_features = iris.data.shape

>>> print (n\_samples, n\_features)

150 4

>>>

150 rows and four columns.

Step 6: Print sample data:

>>> print (iris.data)

[[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

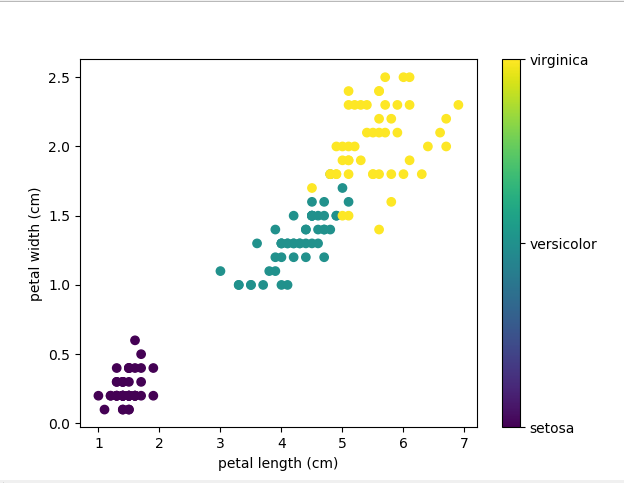
[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

**4. generate matplotlib plots that will assist in identifying the optimal clf and parampters settings**

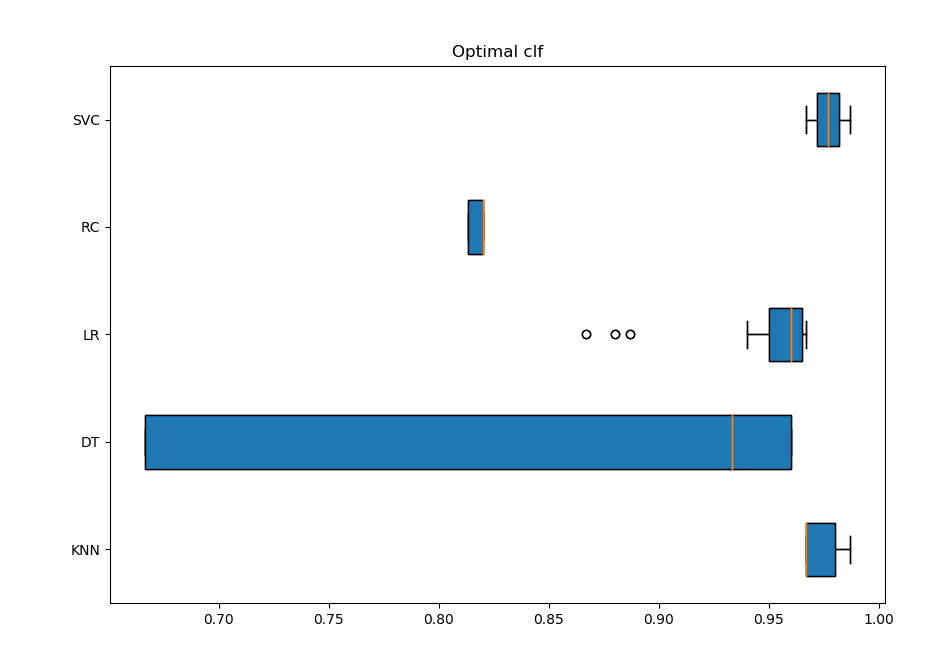
**Plot the iris data.**



Using for loop created accuracy score leveraging CV (Cross Validation) for all five classifiers (KNeighborsClassifier, DecisionTree, LogisticRegression, RidgeClassifier and SVC). Calculated min, max nd average for each classifier. KNN and SVC were very close, but SVC did little better. Although their max and min accuracy was same.

|  |  |  |  |
| --- | --- | --- | --- |
| Classifier | Min | Max | Average |
| KNeighborsClassifier | 0.96667 | 0.98667 | 0.97267 |
| DecisionTreeClassifier | 0.64667 | 0.95333 | 0.82 |
| LogisticRegression | 0.86667 | 0.96667 | 0.9463 |
| RidgeClassifier | 0.81333 | 0.82 | 0.81778 |
| SVC | 0.96667 | 0. 98667 | 0.97778 |

**Created the Box plot:** Comparison of five classifier Accuracy with different parameter set:



Created box plot comparing the matrix of classifier:

**Best Parameters for Accuracy**

Selected the best parameter set for each classifier considering max accuracy as key measure:

KNeighborsClassifier:

KNN \_Parm: {'algorithm': 'auto', 'n\_neighbors': 10, 'weights': 'distance'} parmeter set gave higest accuracy of 0.9866666666666667.

DecisionTreeClassifier:

DT \_Parm: {'criterion': 'gini', 'max\_depth': 3, 'min\_samples\_split': 4} parmeter set gave higest accuracy of 0.9533333333333335.

LogisticRegression:

LR \_Parm: {'C': 0.1, 'penalty': 'none', 'solver': 'sag'} parmeter set gave higest accuracy of

0.9666666666666668.

RidgeClassifier:

RC \_Parm: {'alpha': 1.0, 'fit\_intercept': 'False', 'normalize': 'False'} parmeter set gave higest accuracy of 0.8200000000000001.

SVC:

SV \_Parm: {'C': 0.5, 'class\_weight': 'balanced', 'kernel': 'linear'} parmeter set gave higest

accuracy of 0.9866666666666667.

**# 5. Please set up your code to be run and save the results to the directory that its executed from**

fig = plt.figure(figsize =(10, 7))

# Creating plot

box\_plot\_data=[KNN, DT, LR, RC,SV]

box=plt.boxplot(box\_plot\_data,vert=0,patch\_artist=True,labels=['KNN', 'DT', 'LR','RC', 'SVC'],)

plt.boxplot(box\_plot\_data,vert=0, patch\_artist=True,labels=['KNN', 'DT', 'LR', 'RC', 'SVC'],)

colors = ['cyan', 'lightblue', 'lightgreen','tan','red']

for patch, color in zip(box['boxes'], colors):

  patch.set\_facecolor(color)

plt.title('Optimal clf')

#Save plot file to specific location on my computer

# Response to Q5S

plt.savefig("C:\\Paritosh\\SMU\\7335 Machine Learning 2\\VS Code\\clf\_plot.jpg")

plt.savefig("C:\\Paritosh\\SMU\\7335 Machine Learning 2\\VS Code\\clf\_plot.jpg")

**# 6. Investigate grid search function**

Grid search is an optimization algorithm that selects the best parameters from the parameters' list (provided by users). It leverages the automated 'trial-and-error' method. However, it can be applied to many optimization problems, most popularly used in machine learning to obtain the parameters at which the model gives the best accuracy.

If we had 10 different input parameters, we wanted to try out 5 possible values for each parameter. It would require manual input from our side each time we wish to change a parameter value, rerun the code, and keep track of the results for all the combinations of parameters. Grid Search automates this process. It takes the possible values for each parameter, runs the code with all possible combinations, outputs the result for each variety, and helps identify the best possible accuracy.

To determine which classifier determines the best value with multiple parameter combinations with multiple values, users have to run a grid search for each classifier and then compare their accuracy.

In this exercise, code build will determine the best possible classifier with the best possible parameter combinations to deliver the highest accuracy. Code takes multiple classifiers, builds an optimal parameter set for each classifier. Compares the best accuracy of each classifier and determines the best classifier with the best parameter set.

**Conclusion:** Grid search helps pick the best parameter combination for the given classifier. The code used above selects the best classifier with the best parameter set.