Hasan Al-Qudah

Compare classifiers in scikit-learn library

Introduction:

In this project we utilize scikit-learn in python which contains a robust library you can use to implement different machine learning. Scikit-learn provide a range of supervised and unsupervised learning algorithms via a consistent interface in Python. In this project we implement

1. perception
2. Support Vector Machine (linear and non-linear using Radial Basis Function (RBF) kernel).
3. decision tree,
4. K-nearest neighbour.
5. Logistic regression.

We test classifiers on two data set, digits dataset and Activity Recognition respectively. The first dataset has loaded from Scikit-learn data library. This dataset contains 1797 8x8 images is of a hand-written digit. The data contains 7494 training set and 3498 and 10 clases. The other dataset is a heterogeneity Dataset for Human Activity Recognition from Smartphone and Smart watch sensors consists of two datasets devised to investigate sensor heterogeneities' impacts on human activity recognition algorithms. This data contains 179037 instance and 119 attributes divided between 32 classes.

Data pre-processing:

To prepare data for training we divide data into two different set (training 70%, testing 30%) by utilizing train_test_split from sklearn pre-processing method. Then we apply the Standardization technique which useful is to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. It is most suitable for techniques that assume a Gaussian distribution in the input variables and work better with rescaled data, such as linear regression, logistic regression and linear discriminate analysis. From sklearn.preprocessing we import StandardScaler figure 1 shows the pre-processing code.

Compare classifiers in scikit-learn library

```
    test_size=0.3
    randome_state=0
    X_train, X_test, y_train, y_test = train_test_split(X,
Y,test_size=test_size,random_state=randome_state)
     #standarized the data
    sc=StandardScaler()
    sc.fit(X_train)
    X_train_std=sc.transform(X_train)
    X_test_std=sc.transform(X_test)
#X_train_std=X_train_std[:,[2,3]]
    X_train_std=X_train_std[:,:]
    X_test_std=X_test_std[:,:]
```

figure1: Standard Scalar

The usage of standardization technique data increase the accuracy of all classifiers and make the running data faster especially in the SVM and regression classifiers. For example, it takes a long time to run the SVM linear and none Liner. So, I show the result for standardizing data only**.**

Results

We classify the digit dataset on the five classifiers we mentioned earlier and we calculate the accuracy score for each classifier. We also recorded the running time for each classifier in order to compare the different performance for these algorithms. Table1 shows the classifiers with their accuracy score and running time in second.

classifier	Accuracy	Running Time(second)
perception	92.9 %	0.12191343307495117
Support vector machine (linear)	97 %	0.07695484161376953
Support vector machine (non linear)	98.5%	0.18689274787902832
decision tree	85.9%	0.02698826789855957
K-nearest neighbour	97.0%	0.13192391395568848
Logistic regression	98.8%	0.2348799705505371

Table1: classifiers with their accuracy score and running time in second on digit dataset

Hasan Al-Qudah

Compare classifiers in scikit-learn library

table 2 shows the result of five classifiers on Activity Recognition datasets

classifier	Accuracy	Running Time(second)
perception	94.6%	51.98766469955444
Support vector machine (linear)	99.3%	182.89206790924072
Support vector machine (non linear)	97.6%	412.2090344429016
decision tree	96.83%	35.15687370300293
K-nearest neighbour	98.16%	453.88919830322266
Logistic regression	98.8%	1058.9294638633728

Algorithm behaviour

1. Support vector machines (SVC linear, SVC RBF) provide a robust mechanism for classification, regression and outlier detection, scaling well even with a large-dimensional feature vector. The choice of the parameter C is important to obtain a appropriately trained SVM. In our project on both problem concluded the best settings are C = 0.025 on linear kernel and C = 1 for SVC RBF.

2. Knn the only parameter that can be tuned is K . Hence, in the test I choose the default parameter of K=5. Changing of K to be more than 5 was dropped the accuracy.

3. In logistic regression classifier the main effective parameter was the number of iteration. When I choose a small number of iteration the data are not converge. In order to get the desired input for both datasets I increase the number of iteration to be 5000. Another important parameter was the solver in our classifier i use 'lbfgs' the which is solver for weight optimization and consider one of quasi-Newton methods

4. Decision tree implemented using a default setting for algorithm by using Gini impurity as a node splitting measure. I noticed with experiments with decision trees, the results were not competitive when compared to other classifiers. I provide a parameter to pre-prune by specifying parameters like max depth which I set to 35

Hasan Al-Qudah

Compare classifiers in scikit-learn library

Strategies for implementing pruning in Decision tree

The goal of decision tree-based models is to split the tree for each leaf node corresponds to the prediction of a single class. However, this can lead to the tree radically overfitting the data in pre-pruning, the algorithm stop the decision tree growth before it fits the training data.

- The first Pre-pruning strategy is implemented in lines of code: 278-293 which is min_impurity_decrease OR min_impurity_split.

- The other strategy of Pre-pruning strategy is max_depth parameter tuning which can be found in Line of code: 171-172 .

link to code:
• https://github.com/scikit-learn/scikit-learn/blob/bac89c2/sklearn/tree/tree.py#L518