CENG 3521 DATA MINING ASSIGMENT 3

Gizem PESEN
pesengizem@gmail.com

Tuesday 15th December, 2020

1 Contents

- Declaration of Honor Code
- Ensemble Learning
- Different training sets
- Bagging Pasting
- Boosting
- Different learning algorithm
- Different parameter setting
- k-Nearest Neighbors Classifier

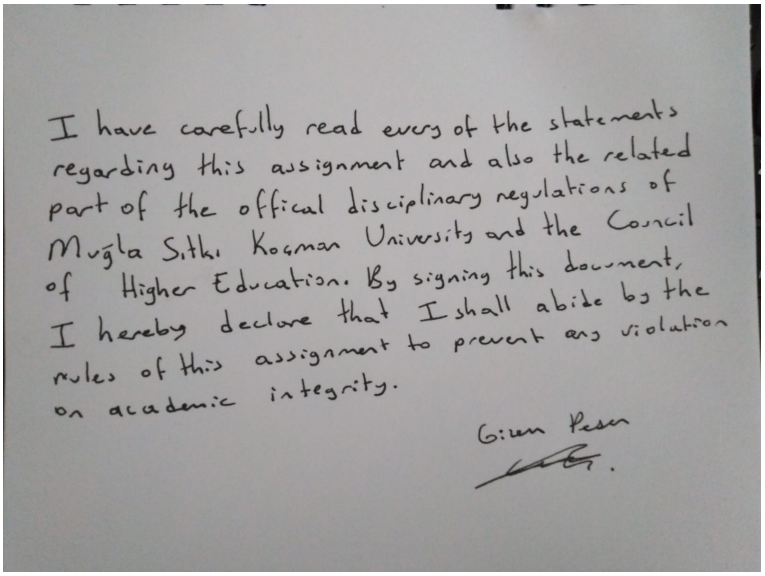
2 Declaration of Honor Code

Student ID :170709050
Name Surname: Gizem Pesen

In the course of Data Mining (CENG 3521), I take academic integrity very seriously and ask you to do as well. That's why, this page is dedicated to some clear statements that defines the policies of this assignment, and hence, will be in force. Before reading this assignment booklet, please first read the following rules to avoid any possible violation on academic integrity.

- This assignment must be done individually unless stated otherwise.
- You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, you cannot copy code (in whole or in part) of someone else, cannot share your code (in whole or in part) with someone else either.
- The previous rule also holds for the material found on the web as everything on the web has been written by someone else. • You must not look at solution sets o

- You must not look at solution sets or program code from other years.
- You cannot share or leave your code (in whole or in part) in publicly accessible areas.
- You have to be prepared to explain the idea behind the solution of this assignment you submit.
- Finally, you must make a copy of your solution of this assignment and keep it until the end of this semester.



3 Ensemble Learning

3.0.1 Different training sets

3.0.2 Bagging Pasting

Listing 1: digit dataset

```
#1. Load digit dataset (D).
  digits = datasets.load_digits()
  X = digits.data
  y = digits.target

  #or we can write X,y = load_digits(return_X_y=True)
```

.

Listing 2: train and test

```
#2. 70% tuples are used for training while 30% tuples are used for testing.
  X_train,X_test,y_train,y_test = model_selection.train_test_split(X,y, test_size=0.3,
      train_size= 0.7, random_state=42)
  scaler = StandardScaler()
  scaler.fit(X_train)
  X_train = scaler.transform(X_train)# Now apply the transformations to the data
  X_test = scaler.transform(X_test)
```

Listing 3: mlp

```
#3. Create an instance of multi-layer perceptron network
  mlp = MLPClassifier(hidden_layer_sizes=(16,8,4,2),max_iter=1001) #(four hidden layers
      with 16, 8, 4,and 2 neurons in order)
  mlp.fit(X_train,y_train)
```

Listing 4: BaggingClassifier

```
#4. Apply bagging classifier with eight base classifiers created at the previous step.
  clf = BaggingClassifier(mlp,n_estimators=8 )
  clf.fit(X_train, y_train)
  clf.score(X_test,y_test)
```

In the assignment , It says "**Calculate number of correctly classified test instance**". So I used this logic

Listing 5: $\text{predicted}_{i}nstances_{p}er_{c}lass$

```
  predicted_instances_per_class = cm[np.eye(len(clf.classes_)).astype("bool")]
```

and the output was this values ;

```
47  out of 540 instances are correctly classified by learner
39  out of 540 instances are correctly classified by learner
44  out of 540 instances are correctly classified by learner
36  out of 540 instances are correctly classified by learner
60  out of 540 instances are correctly classified by learner
53  out of 540 instances are correctly classified by learner
49  out of 540 instances are correctly classified by learner
51  out of 540 instances are correctly classified by learner
34  out of 540 instances are correctly classified by learner
48  out of 540 instances are correctly classified by learner
```

To have inforations I find Confusion Matrix ;

```
Confusion Matrix:
 [[ 0  0 53  0  0  0  0  0  0  0]
 [ 0  0 42  0  0  6  0  1  1  0]
 [ 0  0 40  1  0  3  0  1  0  2]
 [ 0  0  1 48  0  0  0  2  3  0]
 [ 0  0 56  0  0  4  0  0  0  0]
 [ 0  0  3  0  0 61  2  0  0  0]
 [ 0  0  0  0  0  3 50  0  0  0]
 [ 0  0  1  0  0  1  0 52  0  1]
 [ 0  0  0  1  0  0  1  1 40  0]
 [ 0  0 23  2  0  1  0 22  3  8]]
```

3.0.3 Boosting

Listing 6: moon dataset

```
#1. Load moon dataset D with a tuple size greater than 100 .
  X,y = make_moons(n_samples=100, noise=noise)
```

Listing 7: moon noise

```
#2.Give Gaussian noise to D with a deviation value of 0.2.
  noise = 0.2
```

Listing 8: train and test

```
#3. 70% tuples are used for training while 30% tuples are used for testing.
  X_train,X_test,y_train,y_test = model_selection.train_test_split(X,y,test_size =0.3,
      train_size = 0.7)
```

Listing 9: logistic regression

```
#4. Create an instance of logistic regression algorithm with SGD solver
  clf_1 = LogisticRegression(random_state=0)
  clf_1.fit(X,y)
  reg = linear_model.SGDClassifier(loss='log',max_iter=10000) #(with log loss function).
  reg.fit(X,y)
```

Listing 10: AdaBoost classifier

```
  #5. Apply AdaBoost classifier with four base classifiers created at the previous step
  clf_2 = AdaBoostClassifier(clf,n_estimators=100, random_state=0)
  clf_2.fit(X, y)
```

3.0.4 Different learning algorithm

Listing 11: dataset

```
  #1. Load breast-cancer dataset D
  data = load_breast_cancer()
    X = data.data
    y = data.target
    X_train,X_test,y_train,y_test = model_selection.train_test_split(X,y,
        test_size=0.3, train_size= 0.7, random_state=42)
```

.

Listing 12: three classification

```
#2. Create instances of three classification algorithms (feel free to choose estimator
    algorithms).
  log_clf = LogisticRegression()
  rnd_clf = RandomForestClassifier()
  svm_clf = SVC()
  voting_clf = VotingClassifier(
    estimators=[('lr', log_clf),('rf',rnd_clf),('svc',svm_clf)],voting='hard')
  voting_clf.fit(X_train, y_train)
```

Listing 13: accuracy

```
#4. Calculate accuracy of every base classifier and ensemble classifier
  for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print("Accuracy obtained by learner",clf.__class__.__name__ + " is:")
    print(accuracy_score(y_test, y_pred))
```

The output that I find is like;

```
Accuracy obtained by learner LogisticRegression is:
0.9707602339181286
Accuracy obtained by learner RandomForestClassifier is:
0.9649122807017544
Accuracy obtained by learner SVC is:
0.935672514619883
Accuracy obtained by learner VotingClassifier is:
0.9707602339181286
```

3.0.5 Different parameter setting

Listing 14: dataset

```
#1. Load breast-cancer dataset (D).
  data = load_breast_cancer()
  X = data.data
  y = data.target
```

Listing 15: train and test

```
#2. 70% tuples are used for training while 30% tuples are used for testing.
  X_train,X_test,y_train,y_test = model_selection.train_test_split(data,data.target,
      test_size=0.3,train_size= 0.7, random_state=42)
```

Listing 16: multi-layer perceptron

```
#3. Create 10 instances of multi-layer perceptron network with different hidden layer
    and neuron size.
  hidden_size = 10
  loss_train = np.zeros(hidden_size)
  loss_test = np.zeros(hidden_size)
```

.

Listing 17: 2**n

```
#4
  for i in range(1, hidden_size +1):
    a = tuple() #hidden layer size
    for t in range(1, i+1):
      a = (2 ** (t),) + a #until 2**n
    mlp = MLPClassifier(a, activation="relo", solver = "sgd", shuffle =True)
    mlp.fit(X_train, y_train)
    loss_train[i - 1] = mlp.score(X_train,y_train) #with score we calculate loss value
    loss_test[i - 1] = mlp.score[X_test, y_test]
```

3.0.6 3 k-Nearest Neighbors Classifier

Listing 18: dataset

```
#1. Load moon dataset D with a tuple size greater than 100 .
X,y = make_moons(n_samples=100, noise=n)
```

I used this get neighbors function to make easier my job ;

Listing 19: a function

```
#5. Through a 14-axis figure. For each axis, visualize training samples
def get_neighbors(xs, sample, k=5):
  neighbors = [(x, np.sum(np.abs(x - sample))) for x in xs]
  neighbors = sorted(neighbors, key=lambda x: x[1])
  return np.array([x for x, _ in neighbors[:k]])
```

Listing 20: visualize

```
_, ax = plt.subplots(nrows=1, ncols=4, figsize=(15, 5))
for i in range(4):
  sample = X_test[i]
  neighbors = get_neighbors(X_train, sample, k=5)
  ax[i].scatter(X_train[:, 0], X_train[:, 1], c="skyblue")
  ax[i].scatter(neighbors[:, 0], neighbors[:, 1], edgecolor="green")
  ax[i].scatter(sample[0], sample[1], marker="+", c="red", s=100)
  ax[i].set(xlim=(-2, 2), ylim=(-2, 2))

  plt.tight_layout()
```

Output :