Evaluation

November 15, 2021

Evaluation

Sample	Attributes				Buy new
	Age	Income	Student	Credit rating	Computer
S01	Youth	High	No	Good	No
S02	Youth	High	No	Excellent	No
S03	Middle-aged	High	Yes	Good	Yes
S04	Senior	Medium	Yes	Good	Yes
S05	Senior	Low	Yes	Good	Yes
S06	Senior	Low	Yes	Excellent	No
S07	Middle-aged	Low	Yes	Excellent	Yes
S08	Youth	Medium	No	Good	No
S09	Youth	Low	Yes	Good	Yes
S10	Senior	Medium	Yes	Good	Yes
S11	Youth	High	Yes	Excellent	Yes
S12	Middle-aged	Medium	No	Excellent	Yes
S13	Senior	Medium	No	Excellent	No
S14	Middle-aged	High	Yes	Good	Yes

Evaluation

1. True Positive (TP): The actual positive class is predicted positive.
2. True Negative (TN): The actual negative class is predicted negative.
3. False Positive (FP): The actual class is negative but predicted as Positive.
4. False Negative (FN): The actual class is positive but predicted as negative.
5. Accuracy Score = (TP + TN)/ (TP + FN + TN + FP)
6. Precision = TP/(TP + FP)
7. Recall = TP/(TP + FN)
8. Specificity = TN/(TN + FP)
9. F1-score is the Harmonic mean of the Precision and Recall F1-score =2* (Precision * Recall)/(Precision + Recall)

Evaluation: Split data

```
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

pd.set_option('display.max_colwidth', None)
computer = pd.read_csv('/Users/catherine/Desktop/NLP/MachineLearning/MachineLearning2021/computer.csv')

le = preprocessing.LabelEncoder()
x_train = computer[["Age", "Income", "Student", "Credit rating"]]
x_train = pd.DataFrame(columns=x_train.columns, data=le.fit_transform(x_train.values.flatten()).reshape(x_train.shape))

y_train = le.fit(computer["Buy new Computer"])
y_train = le.transform(computer["Buy new Computer"])#converts to 0 and 1

Xd_train, Xd_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.35)
print(Xd_train_knn)
print("y_train = ",y_train, "\n")
print(Xd_test_knn, "\n")
```

```
    Age  Income  Student  Credit rating
0     9       2        6              1
10    9       2        8              0
11    5       4        6              0
12    7       4        6              0
4     7       3        8              1
6     5       3        8              0
8     9       3        8              1
9     7       4        8              1
7     9       4        6              1
y_train =  [1 1 0 1 0 1 0 0 1]

    Age  Income  Student  Credit rating
3     7       4        8              1
5     7       3        8              0
2     5       2        8              1
1     9       2        6              0
13    5       2        8              1
```

Evaluation: Calculate

KNeighborsClassifier Results
y_test = [0 1 1 1 1]
y_pred = [0 0 1 1 1]

TP	3
FP	0
TN	1
FN	1
Acc= (TP + TN)/ (TP + FN + TN + FP)	(3+1)/(3+1+1+0) = 4/5 = 0.8
Precision = TP/(TP + FP)	3/(3+0) = 1
Recall = TP/(TP + FN)	3/(3+1) = 0.75
Sensitivity = TP/(TP + FN)	3/(3+1) = 0.75
Specificity = TN/(TN + FP)	1/(1+0) = 1
F1-score = 2* (Precision * Recall)/(Precision + Recall)	2*(1*0.75)/(1+0.75) = 0.85714285714

Evaluation: KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier

knnclassifier = KNeighborsClassifier(n_neighbors = 3, metric='cosine')
knnclassifier.fit(Xd_train, y_train)

y_pred = knnclassifier.predict(Xd_test)

KNN_Accuracy = accuracy_score(y_test, y_pred)

print( "KNeighborsClassifier Results")
print( "y_test = ",y_test)
print("y_pred = ",y_pred,"\n")

# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %f' % accuracy)

# precision tp / (tp + fp)
precision = precision_score(y_test, y_pred)
print('Precision: %f' % precision)

# recall: tp / (tp + fn)
recall = recall_score(y_test, y_pred)
print('Recall: %f' % recall)

# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, y_pred)
print('F1 score: %f' % f1, "\n")

print("confusion_matrix \n", confusion_matrix(y_test, y_pred))

report = classification_report(y_test, y_pred)
print(report)
```

```
KNeighborsClassifier Results
y_test =  [0 1 1 1 1]
y_pred =  [0 0 1 1 1]

Accuracy: 0.800000
Precision: 1.000000
Recall: 0.750000
F1 score: 0.857143

confusion_matrix
 [[1 0]
 [1 3]]
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       1.00      0.75      0.86         4

    accuracy                           0.80         5
   macro avg       0.75      0.88      0.76         5
weighted avg       0.90      0.80      0.82         5
```

Evaluation: GaussianNB

```
from sklearn.naive_bayes import GaussianNB

clf_nb=GaussianNB()

clf_nb.fit(Xd_train, y_train)

y_pred = clf_nb.predict(Xd_test)

NB_Accuracy = accuracy_score(y_test, y_pred)

print( "GaussianNB Results","\n")
print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")


# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(y_test, y_pred)
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_test, y_pred)
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, y_pred)
print('F1 score: %f' % f1, "\n")

print("confusion_matrix \n", confusion_matrix(y_test, y_pred), "\n")
report = classification_report(y_test, y_pred)
print(report)
```

```
GaussianNB Results

y_test =  [0 1 1 1 1]
y_pred =  [0 0 1 1 1]

Accuracy: 0.800000
Precision: 1.000000
Recall: 0.750000
F1 score: 0.857143

confusion_matrix
 [[1 0]
 [1 3]]

              precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       1.00      0.75      0.86         4

    accuracy                           0.80         5
   macro avg       0.75      0.88      0.76         5
weighted avg       0.90      0.80      0.82         5
```

Evaluation: DecisionTreeClassifier

```
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier

DT_classifier = DecisionTreeClassifier()

DT_classifier.fit(Xd_train, y_train)

y_pred = DT_classifier.predict(Xd_test)

print( "Decision Tree Results","\n")
print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")


# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(y_test, y_pred)
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_test, y_pred)
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, y_pred)
print('F1 score: %f' % f1, "\n")

print("confusion_matrix \n", confusion_matrix(y_test, y_pred), "\n")
report = classification_report(y_test, y_pred)
print(report)
```

```
Decision Tree Results

y_test =  [0 1 1 1 1]
y_pred =  [0 0 1 0 1]

Accuracy: 0.600000
Precision: 1.000000
Recall: 0.500000
F1 score: 0.666667

confusion_matrix
 [[1 0]
 [2 2]]

              precision    recall  f1-score   support

           0       0.33      1.00      0.50         1
           1       1.00      0.50      0.67         4

    accuracy                           0.60         5
   macro avg       0.67      0.75      0.58         5
weighted avg       0.87      0.60      0.63         5
```

Implementing the K-Fold Cross-Validation

The dataset is split into ‘k’ number of subsets, k-1 subsets then are used to train the model and the last subset is kept as a validation set to test the model. Then the score of the model on each fold is averaged to evaluate the performance of the model.

5 Fold Cross Validation

Evaluation: KFold with DecisionTreeClassifier()

```
from sklearn.model_selection import KFold
computer = pd.read_csv('/Users/catherine/Desktop/NLP/MachineLearning/MachineLearning2021/computer.csv')
le = preprocessing.LabelEncoder()
x_train = computer[["Age", "Income", "Student", "Credit rating"]]
#converts to 0 and 1
x_train = pd.DataFrame(columns=x_train.columns, data=le.fit_transform(x_train.values.flatten()).reshape(x_train.shape))
y_train = le.fit(computer["Buy new Computer"])
y_train = le.transform(computer["Buy new Computer"])#converts to 0 and 1

#Implementing cross validation with DecisionTreeClassifier()
k = 5
kf = KFold(n_splits=k, random_state=None) ## random_state=1, shuffle=True
model = DecisionTreeClassifier()

acc_score = []

for train_index , test_index in kf.split(x_train):
    print(train_index, test_index)
    X_train , X_test = x_train.iloc[train_index,:],x_train.iloc[test_index,:]

    Y_train , Y_test = y_train[train_index] , y_train[test_index]

    model.fit(X_train,Y_train)
    pred_values = model.predict(X_test)

    acc = accuracy_score(pred_values , Y_test)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/k

print("\n",'accuracy of each fold - {}'.format(acc_score), "\n")
print('Avg accuracy : {}'.format(avg_acc_score))
```

```
[ 3  4  5  6  7  8  9 10 11 12 13] [0 1 2]
[ 0  1  2  6  7  8  9 10 11 12 13] [3 4 5]
[ 0  1  2  3  4  5  9 10 11 12 13] [6 7 8]
[ 0  1  2  3  4  5  6  7  8 12 13] [ 9 10 11]
[ 0  1  2  3  4  5  6  7  8  9 10 11] [12 13]

 accuracy of each fold - [1.0, 0.6666666666666666, 0.6666666666666666, 0.3333333333333333, 0.5]

Avg accuracy : 0.6333333333333333
```

Evaluation KFold with GaussianNB()

```
#Implementing cross validation with GaussianNB()
k = 5
kf = KFold(n_splits=k, random_state=None) ## random_state=1, shuffle=True
model = GaussianNB()

acc_score = []

for train_index , test_index in kf.split(x_train):
    print(train_index, test_index)
    X_train , X_test = x_train.iloc[train_index,:],x_train.iloc[test_index,:]

    Y_train , Y_test = y_train[train_index] , y_train[test_index]

    model.fit(X_train,Y_train)
    pred_values = model.predict(X_test)

    acc = accuracy_score(pred_values , Y_test)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/k

print("\n",'accuracy of each fold - {}'.format(acc_score), "\n")
print('Avg accuracy : {}'.format(avg_acc_score))
```

```
[ 3  4  5  6  7  8  9 10 11 12 13] [0 1 2]
[ 0  1  2  6  7  8  9 10 11 12 13] [3 4 5]
[ 0  1  2  3  4  5  9 10 11 12 13] [6 7 8]
[ 0  1  2  3  4  5  6  7  8 12 13] [ 9 10 11]
[ 0  1  2  3  4  5  6  7  8  9 10 11] [12 13]

 accuracy of each fold - [0.3333333333333333, 0.6666666666666666, 1.0, 0.6666666666666666, 1.0]

Avg accuracy : 0.7333333333333333
```

Evaluation KFold with KNeighborsClassifier()

```
#Implementing cross validation with KNeighborsClassifier()
k = 5
kf = KFold(n_splits=k, random_state=None) ## random_state=1, shuffle=True
model = KNeighborsClassifier()

acc_score = []

for train_index , test_index in kf.split(x_train):
    print(train_index, test_index)
    X_train , X_test = x_train.iloc[train_index,:],x_train.iloc[test_index,:]

    Y_train , Y_test = y_train[train_index] , y_train[test_index]

    model.fit(X_train,Y_train)
    pred_values = model.predict(X_test)

    acc = accuracy_score(pred_values , Y_test)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/k

print("\n",'accuracy of each fold - {}'.format(acc_score), "\n")
print('Avg accuracy : {}'.format(avg_acc_score))
```

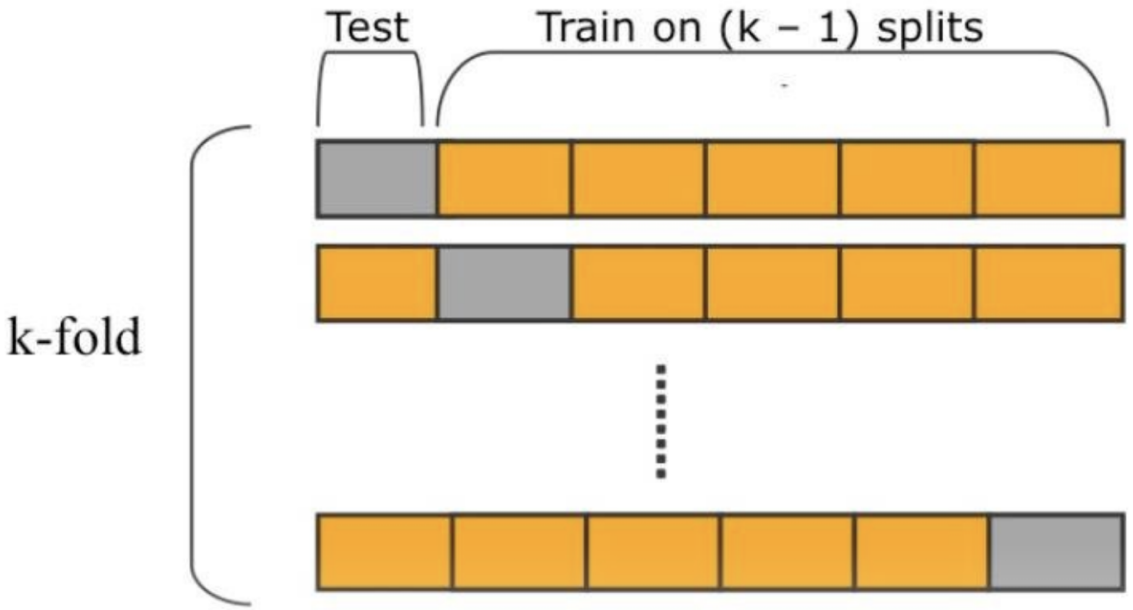
```
[ 3  4  5  6  7  8  9 10 11 12 13] [0 1 2]
[ 0  1  2  6  7  8  9 10 11 12 13] [3 4 5]
[ 0  1  2  3  4  5  9 10 11 12 13] [6 7 8]
[ 0  1  2  3  4  5  6  7  8 12 13] [ 9 10 11]
[ 0  1  2  3  4  5  6  7  8  9 10 11] [12 13]

 accuracy of each fold - [0.6666666666666666, 0.6666666666666666, 1.0, 0.6666666666666666, 0.5]

Avg accuracy : 0.7
```

Evaluation: Leave-One-Out Cross-Validation

1. Split a dataset into a training set and a testing set, using all but one observation as part of the training set.
2. Build a model using only data from the training set.
3. Use the model to predict the response value of the one observation left out of the model
4. Repeat this process n times until you have tested all data points



Evaluation LeaveOneOut with KNeighborsClassifier()

```
from sklearn.model_selection import LeaveOneOut
model = KNeighborsClassifier()

computer = pd.read_csv('/Users/catherine/Desktop/NLP/MachineLearning/MachineLearning2021/computer.csv')
le = preprocessing.LabelEncoder()
x_train = computer[["Age", "Income", "Student", "Credit rating"]]
#converts to 0 and 1
x_train = pd.DataFrame(columns=x_train.columns, data=le.fit_transform(x_train.values.flatten()).reshape(x_train.shape))
y_train = le.fit(computer["Buy new Computer"])
y_train = le.transform(computer["Buy new Computer"])#converts to 0 and 1

acc_score = []

# create loocv procedure
cv = LeaveOneOut()
# enumerate splits
for train_index , test_index in cv.split(x_train):
    print(train_index, test_index)
    X_train , X_test = x_train.iloc[train_index,:],x_train.iloc[test_index,:]

    Y_train , Y_test = y_train[train_index] , y_train[test_index]

    model.fit(X_train,Y_train)
    pred_values = model.predict(X_test)

    acc = accuracy_score(pred_values , Y_test)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/13

print("\n",'accuracy of each loocv - {}'.format(acc_score), "\n")
print('Avg accuracy : {}'.format(avg_acc_score))
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13] [0]
[ 0  2  3  4  5  6  7  8  9 10 11 12 13] [1]
[ 0  1  3  4  5  6  7  8  9 10 11 12 13] [2]
[ 0  1  2  4  5  6  7  8  9 10 11 12 13] [3]
[ 0  1  2  3  5  6  7  8  9 10 11 12 13] [4]
[ 0  1  2  3  4  6  7  8  9 10 11 12 13] [5]
[ 0  1  2  3  4  5  7  8  9 10 11 12 13] [6]
[ 0  1  2  3  4  5  6  8  9 10 11 12 13] [7]
[ 0  1  2  3  4  5  6  7  9 10 11 12 13] [8]
[ 0  1  2  3  4  5  6  7  8 10 11 12 13] [9]
[ 0  1  2  3  4  5  6  7  8  9 11 12 13] [10]
[ 0  1  2  3  4  5  6  7  8  9 10 12 13] [11]
[ 0  1  2  3  4  5  6  7  8  9 10 11 13] [12]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12] [13]

 accuracy of each loocv - [0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0]

Avg accuracy : 0.7692307692307693
```

Evaluation LeaveOneOut with GaussianNB()

```
from sklearn.model_selection import LeaveOneOut

model = GaussianNB()

computer = pd.read_csv('/Users/catherine/Desktop/NLP/MachineLearning/MachineLearning2021/computer.csv')
le = preprocessing.LabelEncoder()
x_train = computer[["Age", "Income", "Student", "Credit rating"]]
#converts to 0 and 1
x_train = pd.DataFrame(columns=x_train.columns, data=le.fit_transform(x_train.values.flatten()).reshape(x_train.shape))
y_train = le.fit(computer["Buy new Computer"])
y_train = le.transform(computer["Buy new Computer"])#converts to 0 and 1

acc_score = []

# create loocv procedure
cv = LeaveOneOut()
# enumerate splits
for train_index , test_index in cv.split(x_train):
    print(train_index, test_index)
    X_train , X_test = x_train.iloc[train_index,:],x_train.iloc[test_index,:]

    Y_train , Y_test = y_train[train_index] , y_train[test_index]

    model.fit(X_train,Y_train)
    pred_values = model.predict(X_test)

    acc = accuracy_score(pred_values , Y_test)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/13

print("\n",'accuracy of each loocv - {}'.format(acc_score), "\n")
print('Avg accuracy : {}'.format(avg_acc_score))
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13] [0]
[ 0  2  3  4  5  6  7  8  9 10 11 12 13] [1]
[ 0  1  3  4  5  6  7  8  9 10 11 12 13] [2]
[ 0  1  2  4  5  6  7  8  9 10 11 12 13] [3]
[ 0  1  2  3  5  6  7  8  9 10 11 12 13] [4]
[ 0  1  2  3  4  6  7  8  9 10 11 12 13] [5]
[ 0  1  2  3  4  5  7  8  9 10 11 12 13] [6]
[ 0  1  2  3  4  5  6  8  9 10 11 12 13] [7]
[ 0  1  2  3  4  5  6  7  9 10 11 12 13] [8]
[ 0  1  2  3  4  5  6  7  8 10 11 12 13] [9]
[ 0  1  2  3  4  5  6  7  8  9 11 12 13] [10]
[ 0  1  2  3  4  5  6  7  8  9 10 12 13] [11]
[ 0  1  2  3  4  5  6  7  8  9 10 11 13] [12]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12] [13]

 accuracy of each loocv - [1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0]

Avg accuracy : 0.8461538461538461
```

Evaluation LeaveOneOut with DecisionTreeClassifier()

```
from sklearn.model_selection import LeaveOneOut

model = DecisionTreeClassifier()

computer = pd.read_csv('/Users/catherine/Desktop/NLP/MachineLearning/MachineLearning2021/computer.csv')
le = preprocessing.LabelEncoder()
x_train = computer[["Age", "Income", "Student", "Credit rating"]]
#converts to 0 and 1
x_train = pd.DataFrame(columns=x_train.columns, data=le.fit_transform(x_train.values.flatten()).reshape(x_train.shape))
y_train = le.fit(computer["Buy new Computer"])
y_train = le.transform(computer["Buy new Computer"])#converts to 0 and 1

acc_score = []

# create loocv procedure
cv = LeaveOneOut()
# enumerate splits
for train_index , test_index in cv.split(x_train):
    print(train_index, test_index)
    X_train , X_test = x_train.iloc[train_index,:],x_train.iloc[test_index,:]

    Y_train , Y_test = y_train[train_index] , y_train[test_index]

    model.fit(X_train,Y_train)
    pred_values = model.predict(X_test)

    acc = accuracy_score(pred_values , Y_test)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/13

print("\n",'accuracy of each loocv - {}'.format(acc_score), "\n")
print('Avg accuracy : {}'.format(avg_acc_score))
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13] [0]
[ 0  2  3  4  5  6  7  8  9 10 11 12 13] [1]
[ 0  1  3  4  5  6  7  8  9 10 11 12 13] [2]
[ 0  1  2  4  5  6  7  8  9 10 11 12 13] [3]
[ 0  1  2  3  5  6  7  8  9 10 11 12 13] [4]
[ 0  1  2  3  4  6  7  8  9 10 11 12 13] [5]
[ 0  1  2  3  4  5  7  8  9 10 11 12 13] [6]
[ 0  1  2  3  4  5  6  8  9 10 11 12 13] [7]
[ 0  1  2  3  4  5  6  7  9 10 11 12 13] [8]
[ 0  1  2  3  4  5  6  7  8 10 11 12 13] [9]
[ 0  1  2  3  4  5  6  7  8  9 11 12 13] [10]
[ 0  1  2  3  4  5  6  7  8  9 10 12 13] [11]
[ 0  1  2  3  4  5  6  7  8  9 10 11 13] [12]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12] [13]

 accuracy of each loocv - [1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0]

Avg accuracy : 0.6923076923076923
```