Machine Learning Homework 2

Apr. 10, 2021

** Please note that all homework should be your own work. You should also not copy answers from other person's, books or internet resources.*
** I didn't proofread the questions. If you find any typos/errors, let me know.*

* Find **ONE** dataset which includes **at least one categorical** attribute. Everyone should use different data and, as you have done in HW 1, drop a line in Q&A saying "OOO uses OOO data for HW2".

Preprocessing

0. Read File
Read your csv file into a 2-dimension list (**a_list**).

1. Label Encoding
scikit library (like most of ML packages) handles ***numbers* only**. That means: if some attributes contain symbols/strings (categorical attributes), you must change them to integer numbers. For example, suppose an attribute 'temperature' contains 3 values: 'high', 'medium', 'low', you have to change them to 0, 1, 2, respectively. (the integer value ought to begin with 0, and, for now, the order of the value doesn't matter) Label encoding **MUST** be done for **EVERY categorical attributes** (including target attributes).

1) scikit learn provides a built-in function 'LabelEncoder' for this purpose. 'LabelEncoder' changes symbols to integers from 0 to (no_of_values – 1). Refer to the following example and implement Label Encoding using 'LabelEncoder'

```
>from sklearn.preprocessing import LabelEncoder
# change symbols to integer
>le = LabelEncoder()
>le.fit(["paris", "paris", "tokyo", "amsterdam"])
# shows unique values
>le.classes_
array(['amsterdam', 'paris', 'tokyo'], dtype='|S9')
# convert symbols to sequential integers
>le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1])
# you can also convert integers into seqential integers (begining 0)
>le.fit([1, 2, 2, 6])
>le.classes_
array([1, 2, 6])
>le.transform([1, 1, 2, 6])
array([0, 0, 1, 2])
```

* After label Encoding for every categorical attribute, 'a_list' file is now renamed to **'a_list_enc'**

2. Normalize
Using 'a_list_enc' above, we are going to normalize the values of each attributes.

***** DON'T normalize target(class) attribute**

1) For every value x of a certain attribute, change it to **|x – min|/|max-min|** where 'min'/'max' means the minimum/maximum value within the attribute, respectively. scikit learn has a function 'MinMaxScaler' which does the same functionality.

```
import numpy as np
data = [44.645, 44.055, 44.54, 44.04, 43.975, 43.49, 42.04, 42.6, 42.46, 41.405]
min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled = min_max_scaler.fit_transform([np.float32(data)])
print data_scaled
```

Normalize every attribute (except target attribute) using MinMaxScaler.

2) Instead of 'MinMaxScaler', now you use 'StandardScaler'.

```
scaler = StandardScaler()
# Fit only to the training data
scaler.fit(X_train)
# Now apply the transformations to the data:
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Normalize every attribute (except target attribute) using StandardScaler.

* **Choose the resulting file from either 1) or 2),** and now 'a_list_enc' file is renamed to **'a_list_enc_norm'**

3. Divide_train_test
From a file 'a_list', 'X_data' is the attribute values except target attribute and 'Y_data' is the list of target values.
For example, suppose a_list=[[1,0,2,3,1], [0,1,1,2,0], [0,1,0,1,1], [0,0,2,3,1]],
X_data = [[1,0,2,3], [0,1,1,2], [0,1,0,1], [0,0,2,3]] and Y_data = [1,0,1,1]

1) write a program which splits 'a_list_enc_norm' file into 'X_data' and 'Y_data'.
2) split 'X_data' and 'Y_data' into X_train, X_test, Y_train, Y_test as follows:
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data, test_size=0.2, random_state=33)

4. Running Neural Network

* refer to the following page.
http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Now you are ready to run scikit program! Refer to the following Python program segment to run neural networks.

```
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,confusion_matrix

...
prepare your X_train, X_test, Y_train, Y_test files.
```

```
# build neural network model
clf = MLPClassifier(hidden_layer_sizes=(20,))

# train your NN using X_train & Y_train data
clf.fit(X_train, Y_train)

# Now training is done
# predict results for X_test
predictions = clf.predict(X_test)

# Compare the 'predictions' with Y_test
# show the accuracy or confusion matrix or etc
... your code ....
```

1) different hidden layer/node
hidden_layer_sizes=(20,) means one hidden layer with 20 hidden nodes
hidden_layer_sizes=(15,15,15) means 3 hidden layers with 15 hidden nodes each.
run neural network with
  a. one hidden layer with 3 different hidden nodes (on your own choice)
  b. two hidden layers with 3 different hidden node configurations. (on your own choice)
2) based on the result of 1), plot the relationship between accuracy and hidden layer/nodes.
3) run neural network by changing 'activation function' {'identity', 'logistic', 'tanh', 'relu'}, Compare the results of these activation functions by plotting
4) run neural network with different 'momentum' values of (0, 0.2, 0.4, 0.6, 0.8). Compare the results of these parameter values by plotting.
5) run neural network with 4 different learning rate (on your choice).
  You have to change 'learning_rate' parameter and (sometimes) 'learning_rate_init' parameter. Compare the results of these parameter values by plotting.

5. Discretization
For every numeric(continuous) attribute, we are going to discretize it, meaning numeric attribute is converted in a number of intervals (bins). In this exercise, we use equal distance discretization.

You can also use scikit library to discretize values.

Referring to the following example code,

```
from sklearn.preprocessing import KBinsDiscretizer
disc = KBinsDiscretizer(n_bins=3, encode='uniform', strategy='uniform')
disc.fit_transform(X)
```

Using **'a_list_enc'** from Q. 1 above, do the following.

1) For every numeric attribute in 'a_list_enc', discretize the attribute with N=4 intervals. (you can change N value if you want)

* After discretizing every numeric attribute, 'a_list_enc' file is now renamed to **'a_list_enc_disc'**

6. Running Decision Tree

FYI, refer to the following page.
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

1) Using Q. 3 method, split **'a_list_enc_disc'** into X_train, X_test, Y_train, Y_test
From Q. 4 (neural network), replace the following line
  clf = MLPClassifier(hidden_layer_sizes=(20,))
  by
  clf = DecisionTreeClassifier()

Run this decision tree **TWICE** by changing criterion='gini' or 'entropy'.
Compare the results. For your dataset, which model is better ? gini or entropy ? (Maybe your 'max_depth' value should be 5 or higher.)
2) show the diagram of one decision tree using graphviz (refer to lab class)
3) run decision tree TEN TIMES by changing 'max_depth' values. What if the max_depth value is very large or very small? You need to compare the two values(very small and very large values).
4) Plot the graph of Q 3) and find the optimal 'max_depth'.
5) Explain why the 'max_depth' is the optimal value in relation to the Q 4) graph.

7. Running IBL (K nearest neighbor)

```
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsClassifier

prepare your X_train, X_test, Y_train, Y_test files

# build IBL model
clf = KNeighborsClassifier(n_neighbors=7)
clf.fit(X_train, Y_train)

# Predict the class labels for the provided data
predictions = clf.predict(X_test)

# calculate the accuracies by comparing predictions and Y_test
... show accuracies using graph

# you may use the following
clf.score(X_test, Y_test)                OR
accuracy_score(Y_test, predictions)

# you can also get the probability estimates for the test data X.
pred_prob = clf.predict_proba(X_test)
```

O refer to the following url
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

1) run the algorithm by changing the value of *n_neighbors* (default=5) to 1, 5, 9, and 13, respectively. Draw a graph showing the relationship between accuracy and

n_neighbors.

2) run the algorithm by changing the value of weights to ‘uniform’ and ‘distance’, respectively. Compare the results by plotting bar graph and explain the meaning of the results.

3) run the algorithm by changing the value of p to 1 (Manhattan) and 2 (Euclidean). Compare the results by plotting bar graph and explain the meaning of the results.

Hand In

Upload the following files at e-class.
i) report file, ii) program source file, iii) datafile file

Due: 4/24(Sat)