CISC 6930 - Data Mining - Assignment 3 - Question 3-4

November 1, 2018

1 CISC 6930 - Data Mining - Assignment 3

Extend KNN classifier from previous assignment to include automated feature selection.

Feature selection is used to remove irrelevant or correlated features in order to improve classification performance. Perform feature selection on a variance of the UCI vehicle dataset in the file `veh-prime.arff`. Compare two different feature selection methods: the Filter method which does not make use of cross-validation performance and the Wrapper method which does.

Note: Fix the KNN parameter to be $k = 7$ for all runs of LOOCV (Leave One Out Cross Validation).

In [1]:
```
# Import packages
from scipy.io import arff
import numpy as np
import pandas as pd
import time
```

In [2]:
```
# Load data
data, metadata = arff.loadarff("veh-prime.arff")

# Create dataframe of the data
x = pd.DataFrame(data)
```

1.1 Question 3: Filter Method

Make the class labels numeric (set "noncar" = 0 and "car" = 1) and calculate the Pearson Correlation Coefficient (PCC) of each feature with the numeric class level. The PCC value is commonly referred to as *r*.

In [3]:
```
#Reclassify the class labels
x['CLASS'] = np.where(x['CLASS'] == b'noncar', 0, 1)
```

In [4]:
```
# A function to calculate the Pearson Correlation Coefficient
# between two variables
def pearson(x,y):

    n = len(x)
    sum_sq_x = 0
```

```python
    sum_sq_y = 0
    sum_coproduct = 0
    mean_x = 0
    mean_y = 0

    for i in range(n):

        sum_sq_x += x[i]**2
        sum_sq_y += y[i]**2
        sum_coproduct += x[i] * y[i]
        mean_x += x[i]
        mean_y += y[i]

    mean_x = mean_x / n
    mean_y = mean_y / n
    pop_sd_x = np.sqrt((sum_sq_x / n) - (mean_x**2))
    pop_sd_y = np.sqrt((sum_sq_y / n) - (mean_y**2))
    cov_x_y = (sum_coproduct / n) - (mean_x * mean_y)
    correlation = cov_x_y / (pop_sd_x * pop_sd_y)

    return correlation
```

(a) List the features from highest $|r|$ to lowest, along with their $|r|$ values. Why would one be interested in the absolute value of r rather than the raw value?

In [5]:
```python
# Separate the features from the label
X = x.iloc[:,:-1]
Y = x.iloc[:,-1]

# Create list to store PCC values
r = list()

# For each feature, calculate its |r| values with respect to the class label
# and append it to the above list
for i in range(36):
    r.append(np.abs(pearson(x.iloc[:,i], Y)))

# Sort the list of PCCs in descending order
r_sorted = pd.DataFrame(sorted(r, reverse = True))

# Obtain the appropriate feature numbers whose |r| values are
# in descending order
features_rank = pd.DataFrame(np.argsort(r)[::-1])

# Create a dataframe that shows the feature number and its respective
# |r| value
df = pd.concat((features_rank, r_sorted),axis = 1)
```

```python
# Rename columns
df.columns = ["Feature Number", "|r| score"]

# Keep the list of sorted feature numbers in terms of |r| score for
# later use
high_low_r = df.iloc[:,0]

# Show the dataframe
df
```

Out[5]:

	Feature Number	|r| score
0	4	0.436922
1	13	0.368269
2	14	0.368224
3	16	0.366025
4	7	0.352141
5	22	0.351350
6	26	0.341043
7	1	0.308811
8	20	0.299049
9	31	0.290783
10	34	0.266093
11	2	0.195732
12	28	0.156904
13	25	0.153096
14	19	0.137636
15	17	0.113945
16	32	0.093174
17	8	0.087773
18	0	0.069795
19	10	0.056876
20	21	0.056605
21	11	0.042117
22	33	0.038810
23	6	0.035295
24	15	0.031478
25	35	0.030855
26	29	0.020829
27	18	0.017931
28	27	0.015606
29	9	0.013005
30	3	0.009214
31	30	0.008955
32	24	0.007780
33	23	0.005508
34	12	0.002179
35	5	0.000098

Answer: The parity of the Pearson correlation coefficient tells whether the data is positively or negatively correlated. In our case, we are not looking at if the correlation is positive or negative but rather just how much two column of data are correlated to each other from 0 being no correlation to 1 being a perfect correlation.

(b) Select the features that have the highest m values of $|r|$ and run LOOCV on the dataset restricted to only those m features. Which value of m gives the highest LOOCV classification accuracy and what is the value of this optimal accuracy?

In [6]:
```
# A function to run LOOCV on a dataset, with features and class labels
# where features to use is given
def KNN_LOOCV(df, feats, k = 7):

    # Make the dataset an array and store its dimensions in variables
    #df = np.array(df)
    rows = df.shape[0]
    cols = df.shape[1]

    # Instantiate a matrix to store distances for each data value
    # its n - 1 neighbors
    dist = np.zeros((rows, rows - 1))

    # Create an empty list for keeping track of LOOCV
    # classification accuracies
    accuracies = list()

    # Iterate over all features given
    for i in feats:

        # Initialize number of correct classifications to 0
        count = 0

        for j in range(rows):

            test_index = df.index.isin([j])
            train = np.array(df[~test_index])
            test = np.array(df[test_index])

            # Create Xtrain and Ytrain from the training set
            Xtrain, Ytrain = np.split(train, [cols - 1], axis = 1)
            Xtrain = Xtrain[:,i]

            # Create Xtest and Ytest from the test set
            Xtest, Ytest = np.split(test, [cols - 1], axis = 1)
            Xtest = Xtest[:,i]

            # Calculate the Euclidean distance between Xtrain and Xtest
            dist[j] = np.add(dist[j], np.sqrt((Xtrain - Xtest)**2))
```

```
                    # Obtain the closest k neighbors by sorting the distances
                    # and getting the top indexes
                    indices = np.argsort(dist[j])[:k]

                    # Get the majority vote of the k neighbors
                    mode = 1 if np.mean(Ytrain[indices]) > 0.5 else 0

                    # If the majority vote is accurate, correct
                    # classification increases by 1
                    if mode == Ytest:
                        count = count + 1

                # Calculate the accuracy of adding the feature
                accuracy = 100 * count / rows
                print("Accuracy of adding feature", i, "is:", accuracy, '%')

                # Store accuracy
                accuracies.append((100 * count) / rows)

            # Return the list of accuracies
            return accuracies
```

```
In [7]: start = time.time()
        # Run the KNN algorithm with LOOCV on the dataset using the
        # ranked feature based on |r|
        accuracies = KNN_LOOCV(x, high_low_r)
        end = time.time()
        print("Time to run Filter Method:", end - start)
```

```
Accuracy of adding feature 4 is: 69.38534278959811 %
Accuracy of adding feature 13 is: 80.02364066193853 %
Accuracy of adding feature 14 is: 83.096926713948 %
Accuracy of adding feature 16 is: 84.39716312056737 %
Accuracy of adding feature 7 is: 83.33333333333333 %
Accuracy of adding feature 22 is: 83.33333333333333 %
Accuracy of adding feature 26 is: 86.7612293144208 %
Accuracy of adding feature 1 is: 88.0614657210402 %
Accuracy of adding feature 20 is: 89.47990543735224 %
Accuracy of adding feature 31 is: 88.53427895981088 %
Accuracy of adding feature 34 is: 89.24349881796691 %
Accuracy of adding feature 2 is: 90.78014184397163 %
Accuracy of adding feature 28 is: 90.89834515366431 %
Accuracy of adding feature 25 is: 91.13475177304964 %
Accuracy of adding feature 19 is: 91.84397163120568 %
Accuracy of adding feature 17 is: 92.55319148936171 %
Accuracy of adding feature 32 is: 93.26241134751773 %
Accuracy of adding feature 8 is: 94.56264775413712 %
Accuracy of adding feature 0 is: 94.32624113475177 %
```

```
Accuracy of adding feature 10 is: 95.15366430260048 %
Accuracy of adding feature 21 is: 95.0354609929078 %
Accuracy of adding feature 11 is: 95.0354609929078 %
Accuracy of adding feature 33 is: 94.91725768321513 %
Accuracy of adding feature 6 is: 94.79905437352245 %
Accuracy of adding feature 15 is: 94.91725768321513 %
Accuracy of adding feature 35 is: 95.15366430260048 %
Accuracy of adding feature 29 is: 94.56264775413712 %
Accuracy of adding feature 18 is: 93.97163120567376 %
Accuracy of adding feature 27 is: 94.44444444444444 %
Accuracy of adding feature 9 is: 94.68085106382979 %
Accuracy of adding feature 3 is: 94.08983451536643 %
Accuracy of adding feature 30 is: 94.56264775413712 %
Accuracy of adding feature 24 is: 94.08983451536643 %
Accuracy of adding feature 23 is: 94.68085106382979 %
Accuracy of adding feature 12 is: 94.56264775413712 %
Accuracy of adding feature 5 is: 94.20803782505911 %
Time to run Filter Method: 19.648001670837402
```

In [8]:

```
# Create dataframe to show m, feature number added and aggregating accuracy
df = pd.DataFrame({"m": list(range(1, len(high_low_r) + 1)),
                   "Feature # Added": high_low_r,
                   "Aggregating Accuracy": accuracies})

# Show the above dataframe
df[["m", "Feature # Added", "Aggregating Accuracy"]]
```

Out[8]:

	m	Feature # Added	Aggregating Accuracy
0	1	4	69.385343
1	2	13	80.023641
2	3	14	83.096927
3	4	16	84.397163
4	5	7	83.333333
5	6	22	83.333333
6	7	26	86.761229
7	8	1	88.061466
8	9	20	89.479905
9	10	31	88.534279
10	11	34	89.243499
11	12	2	90.780142
12	13	28	90.898345
13	14	25	91.134752
14	15	19	91.843972
15	16	17	92.553191
16	17	32	93.262411
17	18	8	94.562648
18	19	0	94.326241

19	20	10	95.153664
20	21	21	95.035461
21	22	11	95.035461
22	23	33	94.917258
23	24	6	94.799054
24	25	15	94.917258
25	26	35	95.153664
26	27	29	94.562648
27	28	18	93.971631
28	29	27	94.444444
29	30	9	94.680851
30	31	3	94.089835
31	32	30	94.562648
32	33	24	94.089835
33	34	23	94.680851
34	35	12	94.562648
35	36	5	94.208038

Answer: The highest LOOCV classification accuracy is when $m = 20$ and the LOOCV accuracy is 95.153664%

1.2 Question 4: Wrapper Method

Starting with the empty set of features, use a greedy approach to add the single feature that improves performance by the largest amount when added to the feature set. This is called Sequential Forward Selection. Define perfomance as the LOOCV classification accuracy of the KNN classifier using only the features in the selection set (including the candidate feature). Stop adding features only when there is no candidate that when added to the selection set increases the LOOCV accuracy.

In [9]:
```
# A function to run LOOCV on a dataset, with features and class labels
# where features to use and calculated Euclidean distances is given
def KNN_LOOCV_wrapper(df, feat, prior_distances, k = 7):

    # Make the dataset an array and store its dimensions in variables
    rows = df.shape[0]
    cols = df.shape[1]

    # Instantiate a matrix to store distances for each
    # data value and its (n - 1) neighbors
    distances = np.zeros((rows, rows - 1))

    # Create an empty list for keeping track of LOOCV
    # classification accuracies
    accuracies = list()

    # Initialize number of correct classifications to 0
    count = 0
```

```python
        for j in range(rows):

            test_index = df.index.isin([j])
            train = np.array(df[~test_index])
            test = np.array(df[test_index])

            # Create Xtrain and Ytrain from the training set
            Xtrain, Ytrain = np.split(train, [cols - 1], axis = 1)
            Xtrain = Xtrain[:,feat]

            # Create Xtest and Ytest from the test set
            Xtest, Ytest = np.split(test, [cols - 1], axis = 1)
            Xtest = Xtest[:,feat]

            # Calculate the Euclidean distance between Xtrain and Xtest
            distances[j] = np.add(prior_distances[j],
                                  np.sqrt((Xtrain - Xtest)**2))

            # Obtain the closest k neighbors by sorting the distances
            # and getting the top indexes
            indices = np.argsort(distances[j])[:k]

            # Get the majority vote of the k neighbors
            mode = 1 if np.mean(Ytrain[indices]) > 0.5 else 0

            # If the majority vote is accurate, correct
            # classification increases by 1
            if mode == Ytest:
                count = count + 1

            # Calculate the accuracy of adding the feature
            accuracies.append((100 * count) / rows)

        # Return the last accuracy and distances
        return accuracies[-1], distances
```

In [10]:
```python
# A function to execute sequential forward selection on a dataset
def sfs(df):

    dist = np.zeros((df.shape[0], df.shape[0] - 1))

    # Store number of features in a variable
    num_features = df.shape[1] - 1

    # Store original list of features to be worked with
    features = list(range(0, num_features))
```

```python
# Instantiate a vector to store the features that improve
# performance by the greatest amount
best_feat = []

# Instantiate a vector to store aggregating LOOCV
# classification accuracies
max_accuracy = [0]

# Create a variable to store previous accuracy value for testing
prev_acc = 0

# Run KNN LOOCV over the range of possible features
for i in range(num_features):

    # Print the iteration number and the
    # current feature selection set
    print("Iteration", i, "'s Best Feature List:", best_feat)

    # Instantiate an empty vector to store LOOCV accuracy,
    # feature number and distance array for all features added to
    # the best feature list
    acc_feat_dist = []

    # Run KNN with LOOCV over all possible features that can be added
    for j in features:
        a, temp_dist = KNN_LOOCV_wrapper(df, j, dist)
        acc_feat_dist.append((a, j, temp_dist))

    # Store the maximum accuracy obtained and its feature number
    new_acc = max(acc_feat_dist)[0]
    best_feat_num = max(acc_feat_dist)[1]
    aggregate_dist = max(acc_feat_dist)[2]

    # If the maximum accuracy is not greater than the previous accuracy
    # value, stop the sequential forward selection process,
    # else add the maximum accuracy and best feature number to its
    # respective lists. Remove the new feature from the list of features
    # to test on, update the distances and store the new acccuracy value
    # in the previous variable
    if new_acc <= prev_acc:
        break
    else:
        print("Accuracy of adding feature", best_feat_num, "is:",
              new_acc, '%')
        max_accuracy.append(new_acc)
        best_feat.append(best_feat_num)
        features.remove(best_feat_num)
        dist = aggregate_dist
```

```python
                    prev_acc = new_acc

            # Remove the initial accuracy test value
            del max_accuracy[0]

            # Return the feature selection set and its LOOCV accuracy list
            return(best_feat, max_accuracy)
```

(a) Show the set of selected features at each step as it grows from size zero to its final size (increasing in size by exactly one feature at each step).

```python
In [11]: start = time.time()
         # Run the above method on the dataset
         test_feat, test_acc = sfs(x)
         end = time.time()
         print("Time to Run Wrapper Method:", end - start)
```

```
Iteration 0 's Best Feature List: []
Accuracy of adding feature 20 is: 75.41371158392435 %
Iteration 1 's Best Feature List: [20]
Accuracy of adding feature 10 is: 83.33333333333333 %
Iteration 2 's Best Feature List: [20, 10]
Accuracy of adding feature 19 is: 89.24349881796691 %
Iteration 3 's Best Feature List: [20, 10, 19]
Accuracy of adding feature 1 is: 91.25295508274232 %
Iteration 4 's Best Feature List: [20, 10, 19, 1]
Accuracy of adding feature 8 is: 93.3806146572104 %
Iteration 5 's Best Feature List: [20, 10, 19, 1, 8]
Accuracy of adding feature 14 is: 95.0354609929078 %
Iteration 6 's Best Feature List: [20, 10, 19, 1, 8, 14]
Accuracy of adding feature 4 is: 95.98108747044917 %
Iteration 7 's Best Feature List: [20, 10, 19, 1, 8, 14, 4]
Accuracy of adding feature 7 is: 96.09929078014184 %
Iteration 8 's Best Feature List: [20, 10, 19, 1, 8, 14, 4, 7]
Time to Run Wrapper Method: 158.78602695465088
```

```python
In [12]: # Print the LOOCV accuracies using a dataframe
         print("LOOCV Accuracies:")
         df = pd.DataFrame({"Feature Added": test_feat,
                            "LOOCV Accuracy": test_acc})
         df
```

```
LOOCV Accuracies:
```

Out[12]:

	Feature Added	LOOCV Accuracy
0	20	75.413712
1	10	83.333333

```
2             19       89.243499
3              1       91.252955
4              8       93.380615
5             14       95.035461
6              4       95.981087
7              7       96.099291
```

(b) What is the LOOCV accuracy over the final set of selected features?

```
In [13]: # Run KNN LOOCV on the final set of selected features
         print("LOOCV accuracy over the final set of selected features: ",
               round(KNN_LOOCV(x, test_feat)[-1],3), '%')
```

```
Accuracy of adding feature 20 is: 75.41371158392435 %
Accuracy of adding feature 10 is: 83.33333333333333 %
Accuracy of adding feature 19 is: 89.24349881796691 %
Accuracy of adding feature 1 is: 91.25295508274232 %
Accuracy of adding feature 8 is: 93.3806146572104 %
Accuracy of adding feature 14 is: 95.0354609929078 %
Accuracy of adding feature 4 is: 95.98108747044917 %
Accuracy of adding feature 7 is: 96.09929078014184 %
LOOCV accuracy over the final set of selected features:  96.099 %
```