Project - 2 **CS205 - Artificial Intelligence**

Prof. Eamonn Keogh

Name: Achyuth Madhav Diwakar

SID#: 862055916

Email: adiwa001@ucr.edu

Date: Thursday, March 22, 2018

For the completion of this project, I referred to the following sources:

http://www.cs.ucr.edu/~eamonn/205/ : Slides on machine learning, evaluation

http://www.d.umn.edu/~deoka001/Normalization.html

https://docs.python.org/3/tutorial/floatingpoint.html

https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

http://scikit-learn.org/stable/modules/neighbors.html

https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.array.html : to select the data structures for storing the data matrix

The results, observations and conclusions obtained in this project are my work only.

All of the key code components were written by me and some trivial code snippets were written by referring to the sources above.

Introduction

The **searchApp** is a feature selection application developed as a project in the CS205 - Artificial Intelligence course at the University of California Riverside by Prof. Eamonn Keogh. The application provides a choice of three algorithms namely: Forward Search, Backward Elimination and a Special Search Algorithm. The goal of the application is to select those features that can determine the class to which the record belongs, with the highest possible accuracy. The algorithm at the core of the application is the K - Nearest Neighbors which would be trained and tested in a leave-one-out manner.

Dataset

The datasets available for this application are:

CS205_SMALLtestdata_58.txt

CS205_BIGtestdata_19.txt.

Both datasets have two class labels 1 and 2. The SMALL dataset has ten features and a hundred instances. The BIG dataset has fifty features and hundred instances. The data is in IEEE 754 floating point format. The first column is the class label and the subsequent columns are the features. The elements are space - separated.

Preprocessing

On input, each value in the dataset was converted to a python floating point value. The class labels were extracted into a one-dimensional array, leaving the feature values in a two-dimensional array. This feature matrix was sent to a normalization function, normalize_final(dataArray). This function calculates the mean value of a feature and the standard deviation for the same. The normalized value is calculated by:

Normalized_Val = (Value - Mean(Feature)) / Standard_deviation(Feature)

The result of this function would be values inside each of the features normalized to values between 0 and 1. The normalized matrix is then converted to a Numpy array for ease of processing.

Algorithms

Leave - one - out Cross Validation

In order to find how accurately a set of features would classify a record into one of the two classes, the KNN algorithm is used. Here, the euclidean distance between the values of the respective features between the training and the test data is computed. These distances are then

sorted in ascending order and the top “K” points in the training set are picked and returned. For the purpose of this application, “K” is taken as one. The label returned by the KNN method is then compared with the label of the test data. The accuracy is calculated by dividing the number of correct predictions with the number of records in the dataset. The percentage accuracy is returned to the function calling this algorithm which is then utilized as described below.

Forward Search

The forward search algorithm in involves, starting with an empty feature set and checking the accuracy by including each feature. This algorithm follows a level by level approach in which the first level would attempt to include each one of the features individually and pick the feature which results in the highest accuracy of class label prediction. In the next level, one more feature is picked from the remaining features in the dataset, and the prediction accuracy for the combined feature set is calculated. In this way, the best feature set in each level is selected. Finally, from among the feature sets picked at each level, the one with the highest accuracy is selected as the best feature set for the given dataset.

Backward Elimination

The Backward elimination is a reverse approach to the forward search. Here the application starts with all the features as being correct and the accuracy is calculated. In the first level, each individual feature is selected and removed, the prediction accuracy for the subsequent features is calculated and the best one is picked. In the next levels, further features are removed. Finally the feature set with the best accuracy is selected as the features that accurately indicate the class of the records in the dataset.

Special Search Algorithm

The special search algorithm is a faster variant of the forward search. The faster performance is achieved by pruning those features that definitely perform worse than the feature In this method, the best accuracy computed till that point is sent to the nearest neighbors function. This best accuracy indicates the number of incorrect class labels predicted by the best feature set found till that point. Therefore, as soon as the current feature set exceeds that number of incorrect predictions, it is discarded and accuracy is set to zero. This results in an approximately 30% faster performance.

Results

Dataset: CS205_SMALLtestdata_58.txt

Forward Search:

Best feature subset was [6, 4] , accuracy is 95.0

Time Taken: 5.6670191288

Backward Elimination:

Best feature subset was [2, 4, 5, 6, 8, 10] , accuracy is 87.0

Time Taken: 4.66513514519

Special Search:

Best feature subset was [6, 4] , accuracy is 97.0

Time Taken: 3.70356583595

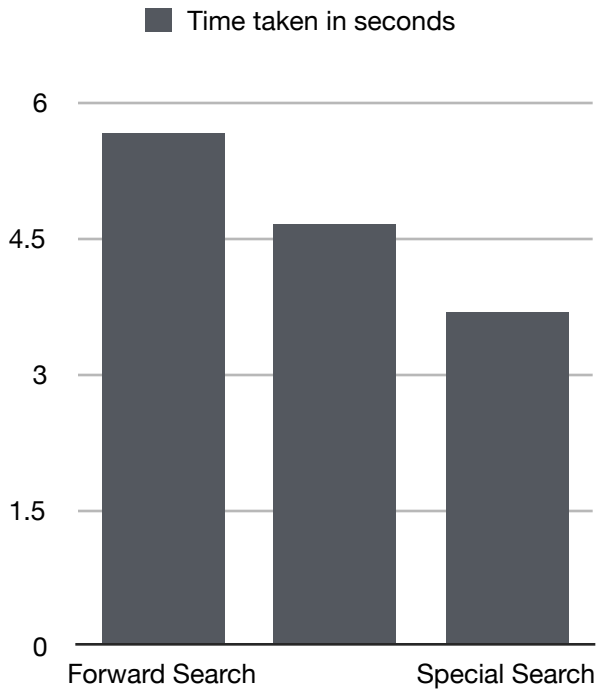

Fig. 1.1 Time taken by each algorithm for the SMALL_58 dataset

Dataset: CS205_BIGtestdata_19.txt

Forward Search:

Best feature subset was [6, 4] , accuracy is 95.0

Time Taken: 5.6670191288

Backward Elimination:

Best feature subset was [3, 4, 6, 8, 11, 12, 14, 17, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 36, 37, 42, 44, 45, 47, 48, 49, 50] , accuracy is 89.0

Time taken: 128.292180061

Special Search:

Best feature subset was [19, 48, 42, 7, 31] , accuracy is 91.0

Time Taken: 106.834325075

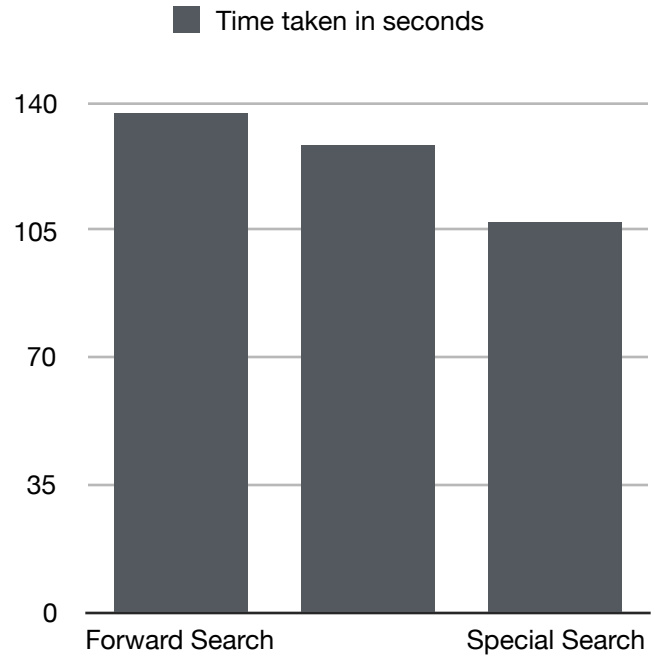

Fig. 1.2. Time taken by each algorithm for the BIG_19 dataset

Accuracy Comparison

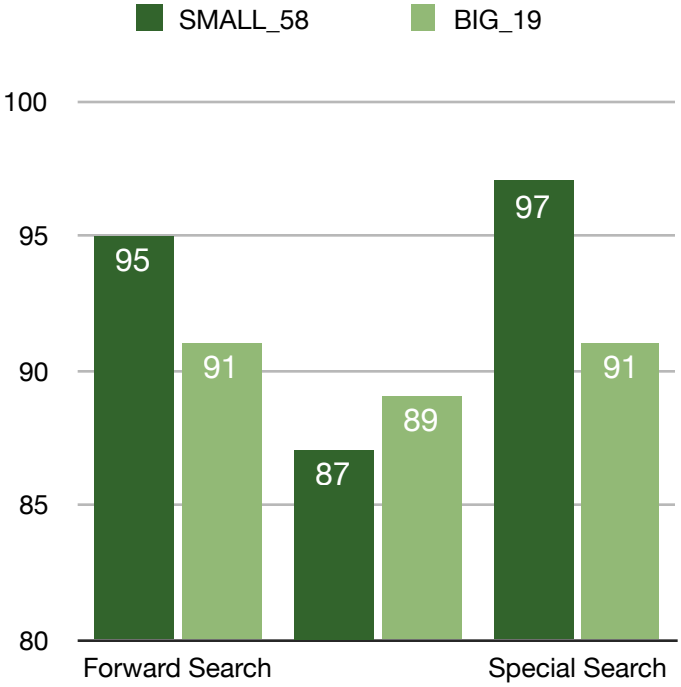

Fig. 1.3. Accuracy of the best feature in each algorithm

Conclusion

This application illustrates very clearly, the differences in performance and accuracy observed when the approach to the feature selection problem is different. As seen in fig. 1.3, the accuracy of the backward elimination algorithm is slightly lower when compared to the forward search and the special search algorithms. However, the backward elimination algorithm does perform slightly faster than the forward search. This can be attributed to the fact that the algorithm starts with a complete feature set and follows a greedy approach to find the best feature set. This could cause it to be skewed in its feature selection. Also, different feature results were obtained when the "K" value was increased to 3 for the backward elimination. From fig. 1.3 it can be seen that with the bigger dataset the accuracy drops slightly. Another observation (from fig.1.1 and fig. 1.2) is that the special search algorithm which prunes the evaluation of the unnecessary feature sets, they performing faster, and with the same results and accuracy as the forward search. Since it clearly outperforms both the algorithms it can be concluded that forward searching with pruning is the better technique for feature selection using KNN.

Output Trace

The output trace for the forward search with the SMALL_58 dataset is shown below:

Connected to pydev debugger (build 173.4301.16)

Welcome to the Feature Selection Algorithm.

Type in the name of the file to test:

/Users/madhav/Documents/Acads/ArtificialIntelligence/P2/CS205_SMALLtestdata__58.txt

Type the number of the algorithm you want to run.

1) Forward Selection

2) Backward Elimination

3) Special Search Algorithm

1

This dataset has 10 features(not counting the class) with 100 instances.

Please wait while I normalize the data. Done!!

Using feature(s) [1] accuracy is 75.0

Using feature(s) [2] accuracy is 73.0

Using feature(s) [3] accuracy is 73.0

Using feature(s) [4] accuracy is 76.0

Using feature(s) [5] accuracy is 75.0

Using feature(s) [6] accuracy is 88.0

Using feature(s) [7] accuracy is 75.0

Using feature(s) [8] accuracy is 76.0

Using feature(s) [9] accuracy is 80.0

Using feature(s) [10] accuracy is 81.0

Feature set [6] was best. Accuracy is 88.0

Using feature(s) [6, 1] accuracy is 86.0

Using feature(s) [6, 2] accuracy is 87.0

Using feature(s) [6, 3] accuracy is 86.0

Using feature(s) [6, 4] accuracy is 95.0

Using feature(s) [6, 5] accuracy is 83.0

Using feature(s) [6, 7] accuracy is 87.0

Using feature(s) [6, 8] accuracy is 88.0

Using feature(s) [6, 9] accuracy is 83.0

Using feature(s) [6, 10] accuracy is 90.0

Feature set [6, 4] was best. Accuracy is 95.0

********** REMOVED FEW LINES**********

**********TO SAVE PAPER**********

Using feature(s) [6, 4, 7, 3, 9, 8, 1, 2] accuracy is 71.0

Using feature(s) [6, 4, 7, 3, 9, 8, 1, 5] accuracy is 82.0

Using feature(s) [6, 4, 7, 3, 9, 8, 1, 10] accuracy is 79.0

Warning! Accuracy has decreased. Continuing search in case of local maxima.

Feature set [6, 4, 7, 3, 9, 8, 1, 5] was best. Accuracy is 82.0

Using feature(s) [6, 4, 7, 3, 9, 8, 1, 5, 2] accuracy is 76.0

Using feature(s) [6, 4, 7, 3, 9, 8, 1, 5, 10] accuracy is 74.0

Warning! Accuracy has decreased. Continuing search in case of local maxima.

Feature set [6, 4, 7, 3, 9, 8, 1, 5, 2] was best. Accuracy is 76.0

Using feature(s) [6, 4, 7, 3, 9, 8, 1, 5, 2, 10] accuracy is 74.0

Warning! Accuracy has decreased. Continuing search in case of local maxima.

Feature set [6, 4, 7, 3, 9, 8, 1, 5, 2, 10] was best. Accuracy is 74.0

Finished Search!! Best feature subset was [6, 4] , accuracy is 95.0

Time Taken: 5.39504885674

Process finished with exit code 0

Code

```
import numpy as np
import math
import time


def forwardSelection(classes, normalized):
    start_time = time.time()
```

```
    currentFeatures = []
    rows, columns = normalized.shape
    finalBest = 0
    featureAtThisLevel = []
    labels = classes[:]
    for i in range(0, columns):
        bestAccSoFar = 0
        maxForLevel = 0
        for j in range(0, columns):
            if j in featureAtThisLevel:
                continue
            accuracy = findAccuracy(labels, normalized, i, j, featureAtThisLevel)
            levelFeatures = [x + 1 for x in featureAtThisLevel]
            print "Using feature(s) ", levelFeatures, " accuracy is ", accuracy
            if accuracy > bestAccSoFar:
                bestAccSoFar = accuracy
                maxForLevel = j

        featureAtThisLevel.append(maxForLevel)
        if (bestAccSoFar > finalBest):
            finalBest = bestAccSoFar
            currentFeatures = featureAtThisLevel[:]
        else:
            print "Warning! Accuracy has decreased. Continuing search in case of local maxima."
        print_features = [x + 1 for x in featureAtThisLevel]
        print "Feature set ", print_features, " was best. Accuracy is ", bestAccSoFar
    finalFeatures = [x + 1 for x in currentFeatures]
    print "Finished Search!! Best feature subset was ", finalFeatures, ", accuracy is ", finalBest
    end_time = time.time()
    print "Time Taken: ", (end_time - start_time)


def backwardElimination(classes, normalized):
    start_time = time.time()
    featureAtThisLevel = []
    rows, columns = normalized.shape
    finalBest = 0
    for x in range(0,len(normalized[0,:])):
        featureAtThisLevel.append(x)
    currentFeatures = featureAtThisLevel[:]
    labels = classes[:]
    for i in range(0, columns):
        bestAccSoFar = 0
        maxForLevel = 0
        for j in range(0, len(currentFeatures)-1):
            delFeature = currentFeatures[j]
            del currentFeatures[j]
            accuracy = findAccuracy(labels, normalized, i, j, currentFeatures)
            levelFeatures = [x + 1 for x in currentFeatures]
            print "Using feature(s) ", levelFeatures, " accuracy is ", accuracy
            if accuracy > bestAccSoFar:
                bestAccSoFar = accuracy
                maxForLevel = j
            currentFeatures.insert(j, delFeature)
        del currentFeatures[maxForLevel]
        if bestAccSoFar > finalBest:
            finalBest = bestAccSoFar
            featureAtThisLevel = currentFeatures[:]
```

```
        else:
            print "Warning! Accuracy has decreased. Continuing search in case of local maxima."
        print_features = [x + 1 for x in currentFeatures]
        print "Feature set ", print_features, " was best. Accuracy is ", bestAccSoFar
    finalFeatures = [x + 1 for x in featureAtThisLevel]
    print "Finished Search!! Best feature subset was ", finalFeatures, ", accuracy is ", finalBest
    end_time = time.time()
    print "Time taken: ", (end_time - start_time)


def specialSearch(classes, normalized):
    start_time = time.time()
    currentFeatures = []
    rows, columns = normalized.shape
    finalBest = 0
    featureAtThisLevel = []
    labels = classes[:]
    for i in range(0, columns):
        bestAccSoFar = 0
        maxForLevel = 0
        for j in range(0, columns):
            if j in featureAtThisLevel:
                continue
            accuracy = betterAccuracyFinder(labels, normalized, finalBest, j, featureAtThisLevel)
            levelFeatures = [x + 1 for x in featureAtThisLevel]
            print "Using feature(s) ", levelFeatures, " accuracy is ", accuracy
            if accuracy > bestAccSoFar:
                bestAccSoFar = accuracy
                maxForLevel = j
            if accuracy == -1:
                continue
        featureAtThisLevel.append(maxForLevel)
        if (bestAccSoFar > finalBest):
            finalBest = bestAccSoFar
            currentFeatures = featureAtThisLevel[:]
        else:
            print "Warning! Accuracy has decreased. Continuing search in case of local maxima."
        print_features = [x + 1 for x in featureAtThisLevel]
        print "Feature set ", print_features, " was best. Accuracy is ", bestAccSoFar
    finalFeatures = [x + 1 for x in currentFeatures]
    print "Finished Search!! Best feature subset was ", finalFeatures, ", accuracy is ", finalBest
    end_time = time.time()
    print "Time Taken: ", (end_time - start_time)

def findAccuracy(classes, normalized, level, feature, featuresSelected):
    knn = KNeighborsClassifier(n_neighbors=1)
    labels = classes
    correctPredicts = 0
    currentFeatures = featuresSelected[:]
    currentFeatures.append(feature)
    for row in range(0, len(labels) - 1):
        labels = classes
        if len(featuresSelected) == 0:
            train = np.array(normalized[:, currentFeatures[:]]).reshape(-1, 1)
            test1 = np.array(normalized[:, currentFeatures[:]]).reshape(-1, 1)
            test = np.array(test1[row]).reshape(-1, 1)
        else:
            train = np.array(normalized[:, currentFeatures[:]])
```

```
            test1 = np.array(normalized[row, currentFeatures[:]])
            test = []
            test.append(test1)
        testLabel = labels[row]
        labels = np.delete(labels, (row), axis=0)
        train = np.delete(train, (row), axis=0)
        knn.fit(train, labels)

        predictedClass = knn.predict(test)[0]
        if predictedClass == testLabel:
            correctPredicts = correctPredicts + 1

    accuracy = (float(correctPredicts) / float(len(classes))) * 100
    #print "accuracy", accuracy
    return accuracy

def betterAccuracyFinder(classes, normalized, bestAccSoFar, feature, featuresSelected):
    knn = KNeighborsClassifier(n_neighbors=3)
    bestIncorrectCount = 100 - bestAccSoFar
    labels = classes
    correctPredicts = 0
    currentFeatures = featuresSelected[:]
    incorrectPredicts = 0
    currentFeatures.append(feature)
    for row in range(0, len(labels) - 1):
        labels = classes
        if len(featuresSelected) == 0:
            train = np.array(normalized[:, currentFeatures[:]]).reshape(-1, 1)
            test1 = np.array(normalized[:, currentFeatures[:]]).reshape(-1, 1)
            test = np.array(test1[row]).reshape(-1, 1)
        else:
            train = np.array(normalized[:, currentFeatures[:]])
            test1 = np.array(normalized[row, currentFeatures[:]])
            test = []
            test.append(test1)
        testLabel = labels[row]
        labels = np.delete(labels, (row), axis=0)
        train = np.delete(train, (row), axis=0)
        knn.fit(train, labels)

        predictedClass = knn.predict(test)[0]
        if predictedClass == testLabel:
            correctPredicts = correctPredicts + 1
        else:
            incorrectPredicts = incorrectPredicts + 1
        if incorrectPredicts > bestIncorrectCount:
            return -1

    accuracy = (float(correctPredicts) / float(len(classes))) * 100
    print "accuracy", accuracy
    return accuracy



def searchStart():
    print "Welcome to the Feature Selection Algorithm."
    print "Type in the name of the file to test: "
    fileName = raw_input()
```

```
    inputFile = open(fileName)
    rawData = inputFile.readlines()
    dataArray = [[]]
    classes = []
    for row in rawData:
        record = map(float, row.split())
        classes.append(record[0])
        del record[0]
        dataArray.append(record)

    del dataArray[0]
    print "Type the number of the algorithm you want to run."
    print "1)   Forward Selection"
    print "2)   Backward Elimination"
    print "3)   Special Search Algoithm"
    algorithmChoice = raw_input()
    instanceCount = len(dataArray)
    featureCount = len(dataArray[0])

    print "This dataset has ", featureCount, " features(not counting the class) with ", instanceCount, " instances."
    print "Please wait while I normalize the data."
    normalized = normalize_final(dataArray)
    npArray = np.array(normalized)
    if algorithmChoice == '1':
        forwardSelection(classes, npArray)

    if algorithmChoice == '2':
        backwardElimination(classes, npArray)

    if algorithmChoice == '3':
        specialSearch(classes, npArray)

def normalize_final(activeDataSet):
    dataSet = activeDataSet
    average = [0.00] * (len(dataSet[0]) - 1)
    stds = [0.00] * (len(dataSet[0]) - 1)
    #       get averages
    for i in dataSet:
        for j in range(1, (len(i))):
            average[j - 1] += i[j]
    for i in range(len(average)):
        average[i] = (average[i] / len(dataSet))
    #       get std's sqrt((sum(x-mean)^2)/n)
    for i in dataSet:
        for j in range(1, (len(i))):
            stds[j - 1] += pow((i[j] - average[j - 1]), 2)
    for i in range(len(stds)):
        stds[i] = math.sqrt(stds[i] / len(dataSet))
    #       calculate new values (x-mean)/std
    for i in range(len(dataSet)):
        for j in range(1, (len(dataSet[0]))):
            dataSet[i][j] = (dataSet[i][j] - average[j - 1]) / stds[j - 1]

    return dataSet


if __name__ == '__main__':
    searchStart()
```