Machine Learning Assignment
Ratul Chakraborty
001910501088

Assignment-

To devise a weighing algorithm to highlight the importance of certain attributes in a classification task using KNN

Algorithm-

- The dataset is divided into training and validation sets
- Principle components of the training set are taken
    - The covariance matrix is calculated for the attributes
    - SVD of the covariance matrix is done
    - We now have the ‘u’ which has the eigenvectors along its columns and ‘s’, which contains the corresponding eigenvalues.
- We take the first ‘n’ values from the decomposition based on our discretion.
- Then for each attribute a_i , we perform (let eigenvectors be of the form V_j and the corresponding value S_j

$$a_i = \sum_{j=1}^{n} \text{cosine\_similarity}(C_j, ai) * S_j$$

- Then the weights are multiplied to the distances along the attributes, resulting in a weighted distance calculation.

Code-

```
def dist(initial,final, weights):
    net = 0
    for i in range(len(initial)):
        # print(len(initial))
        #print(initial[i])
        net += (initial[i] - final[i]) * (initial[i] - final[i]) *weights[i]

    return numpy.sqrt(net)

def ws(info, k1):
    info = numpy.asarray(info)
    info_n = info[:,:-1]
    cova = numpy.cov(info_n, rowvar=False)
#     print(numpy.shape(cova))
    u,s, v = numpy.linalg.svd(cova)
```

```
    pca = PCA(n_components=k1)
#     print(numpy.shape(info_n))
    pca.fit(info_n)
    u = numpy.transpose(u)
#     print(pca.components_)
#     print(u[:k1])

    return u[:k1], s[:k1]


def knn(info, new_point,k, k1):
    k_nearest = []

#     print(ws(info,k1))
    ohohoho = numpy.asarray(ws(info,k1)[0])

    a,s = numpy.shape(ohohoho)

    weights = []
    for _ in range(s):
#         print(numpy.dot(ws(info,k1)[1],ohohoho[:,_]))
#         print("___")
        weights.append(abs(numpy.dot(ws(info,k1)[1],ohohoho[:,_])))

#     weights = numpy.ones(numpy.shape(new_point))

#     print(weights)

    for data in info:
        distance = dist(data[:-1],new_point, weights)
#         print(k_nearest)
#         print(distance)
        j=0
        flag = 1
        for j in range(len(k_nearest)):
            if k_nearest[j][0]<distance:
                flag=0
                break


#         print(j)
        k_nearest.insert(j+flag, (distance, data[-1]))

        if(len(k_nearest) > k):
            k_nearest.pop(0)

    dict_class = {}

    for inf in k_nearest:
        if inf[1] in dict_class:
            dict_class[inf[1]] +=1
        else:
            dict_class[inf[1]] =1

    max =0
```

```
    ret =''
#     print(dict_class)
    for key in dict_class:
        if dict_class[key]>max:
            ret = key
            max= dict_class[key]
    return(ret)
```

Results-

There were 5 iterations taken. In each iteration, both the normal knn and weighted knn were run with 50 random training sets. For each iteration, the average of the 50 runs was taken.

Dataset- Wine quality

	Unweighted KNN	Weighted KNN
1	0.5355314246	0.5414415108
2	0.5378742624	0.5389205537
3	0.5378012121	0.5416345124
4	0.535538272	0.5458412113
5	0.5365382721	0.5433463074
Average	0.5366566886	0.5422368191
Standard deviation	0.001153637418	0.0025600543

Inference-

Adding weights to the important attributes does give us slightly better results consistently.

However, the important assumption for this technique is that the attributes are independent of each other. Otherwise, if there are too many dependent attributes and they are not important for the classification task, the results of using this technique may be poor.