Pattern Recognition: Coursework 2

Malhar Jajoo
Imperial College London
mj2514@ic.ac.uk

Armando Piasko
Imperial College London
armando.piasko17@ic.ac.uk

Abstract

The report explores two Classification Algorithms, Nearest Neighbour (and k-means clustering to speed up Nearest Neighbour) and Artificial Neural Networks for classifying wines (UCI wine dataset) by winery based on its chemical attributes.

1. k-Nearest Neighbour Classification

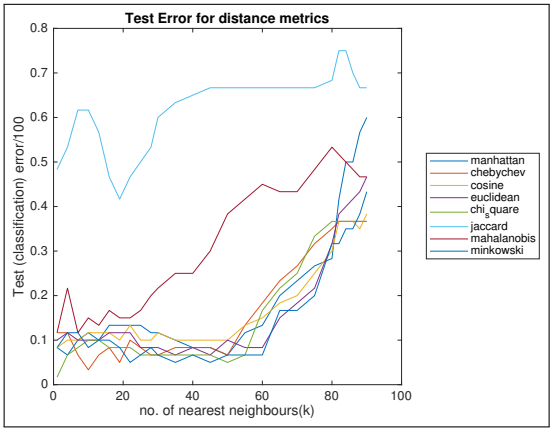

Figure 1: Chi-square distance results in lowest classification error(1.6%) at k=1, while chebychev results in 3.33% at k=10. Jaccard distance results in highest error (75%) at k=82.

1.1. Introduction

In k-Nearest Neighbour (NN) classification, a test point is classified by a majority vote of its k nearest neighbours from training set. The concept of proximity of a neighbour point is quantified using various distance metrics.

1.2. Error with increasing k

Increasing the number of neighbours (k), decreases sensitivity to outliers and reduces classification error as seen in the Figure 1. However, if k is increased upto k=118, then classification error increases and reaches 66.67% for all metrics, as seen in Figure 2 (in Appendix). This scenario is **not** considered when reporting results below as it does not truly reflect error due to a metric, and is instead a mere result of class 2 samples being a majority in the training set and the test set consisting of equal($\frac{1}{3}$) distribution of each class.

1.3. Final results

Metric	Error	k
Chi-square (best)	1.67%	1
Chebychev (best)	3.33%	10
Jaccard (worst)	75%	82
Mahalanobis (worst)	51%	80

Table 1: Best and worst-case results obtained for various distance metrics. k refers to number of nearest neighbours.

All results below are for standardized (using z-score) data[1].

- Chi-square[2]: Gives lowest error rate at k=1 after tuning weights for each attribute during distance calculation. The weights for the 13 attributes used were the following:

$$w = [1, 1, 1, 1, 1, 1, 1, 0.01, 1, 1, 0.01, 1, 10]$$

  These weights may indicate that the 13^{th} attribute (Proline) is relatively more important, while the 8^{th} and 11^{th} are relatively less important for classification of given test set. This would require further domain-specific knowledge for a concrete justification.

- Chebychev: gives the next lowest error rate at k=10 [3].

[1] See ranges of columns 1,4,5,13 of wine data attributes.

[2] Formula suggested by Matlab is $\sqrt{\sum_{i=1}^{n} wi * (x_i - y_i)^2}$. This is created as a custom metric.

[3] A general rule of thumb is selecting $k = \sqrt{n}$. Here $\sqrt{118} \approx 10 give s a lower error rorate.$

It has lower error rate than Chi-square for all k, except k=1.

- Jaccard: gives poor results across all k. This is because it assigns same distance value (=1) to samples from all classes because it relies on dissimilar elements.This prevents it from finding nearest neighbours correctly (since all training points are treated the same) for a test point of a given class. This is shown in Figure 3 (in Appendix).
- Mahalanobis: gives poor results compared to all other metrics (except Jaccard). It mis-classifies *mostly* the sample points from classes 2 and 3. This may indicate that the covariance of training samples of class 1 better reflect the true classification for test points of class 1 (and opposite for classes 2 and 3).
- Euclidean gives comparable or better results than Mahalanobis, Cosine Metric. It may be worth noting here that Mahalanobis and Euclidean distances are not the same which means that Covariance of training and test sample pairs is not Identity Matrix.

2. Clustering for speeding k-NN classifier

2.1. Introduction

The basic intuition of speeding k-NN classifier comes from the fact that for each test point, k nearest points need to be found from a set of n training points. If this set of n training points can be reduced (by clustering) it will lead to speedup.

Setup

In order to avoid confusion, the discussion below uses k to refer to number of nearest neighbours and C to refer to cluster size.

Kindly note that Matlab provides a k-means implementation that works only with the following three metrics - Manhattan, Correlation and Cosine. Hence only these are explored in the following section.

2.2. Random Initialization

Cluster centroid initialization[4] for k-means is done randomly (but based on certain heuristics). This can dramatically affect the clustering which eventually results in variations in classification Error rate (as seen below).

2.3. Algorithm

Prior to demonstrating the effect of Random intialization on output, the algorithm used for cluster selection is briefly explained. This is a layering over Matlabs kmeans().

[4]k-means++ algorithm is used. It is based on probability (which depends on $(distance)^2$) and selects cluster centroids as far apart from each other, allowing faster convergence of k-means.

Let cluster size by C.

1. For a given C and given metric, use Matlab kmeans() to partition data into clusters. The option of 'Replicates'=50 is used in Matlab command kmeans() to ensure that the cluster thus found has minimum sum of within-cluster-distances, among 50 iterations with different cluster centroid initialization positions.
2. Assign labels to a cluster based on (thresholded) majority vote of class labels.
3. Check if assigned cluster labels contain all the three classes otherwise go back to step 1.
4. Use k-NN classifier with a manually set value of k.

Example Matlab code for step 2 of above algorithm is shown in Appendix Code listing 1.

2.4. Effect of Random initialization

	$C = 3$	$C = 4$	$C = 5$	$C = 9$	$C = 10$
k = 1	$11.67 - 18.33$	16.67	18.33	11.67-18.33	10-45

Table 2: Effect of Random initialization for **Manhattan metric**. The "-" represents min and max ranges of Error percentages on different runs. Missing values of cluster size C indicate that the condition in step 3 of the algorithm in section 2.3 was never satisfied despite repeated runs.

The above Table 2 demonstrates how random initialization can affect output of k-means. Similar variability in results was obtained for the Correlation and Cosine metric.

2.5. Final Results

Metric	$Error_{k-NN}$	$Error_C$	C
Manhattan	8.33%	11.67%	1
Correlation	5%	11.67%	3
Cosine	8.33%	%	3

Table 3: Clearly, clustering is not an exact method and increases error rate slightly even after tuning cluster size C (which only matters for $Error_C$. The values for $Error_{k-NN}$ have been derived from the NN classification (not shown earlier).

3. Neural Networks

In this section, a Pattern Recognition Neural Network[5] is employed to classify the wine data. Matlab Neural Network toolbox is used for creating, training and testing the network.

[5]these are feedforward networks that can be trained to classify inputs according to target classes.

3.1. Experiment

Before proceeding on to the experiment kindly note the following abbreviations for training functions: **trainscg** refers to Scaled Conjugate Gradient Descent, **trainlm** refers to LevenBerg-Marquardt, **trainbr** refers to Bayesian Regularization and **traingdm** to Gradient Descent with Momentum.

An experiment was carried out using various parameters as follows-

Parameter	Values explored
No. of hidden layers	1
Perceptron units in hidden layer	[4,6,10]
Training Algorithms	trainscg,trainlm,trainbr, traingdm
Cost Functions	mse, cross-entropy

Table 4: Parameters explored.

For the experiment, only one hidden layer was used. This was based on the observation that simple classifiers like Nearest Neighbour were able to classify with an average error rate of 10%. Adding more hidden layers would result in more complexity (and overfitting).

All the training algorithms mentioned use Backpropagation for learning. Kindly note that unless specified otherwise, values of parameters of the training algorithms used are default values provided by Matlab.

3.2. Regularization

Since Neural Networks are prone to overfitting over small datasets (such as the given dataset), the key to generalizing well is to either use Regularization or a Validation set.

For the given training set partition, splitting it further would result in a very small Validation set and would not improve generalization. Hence the approach taken is Regularization.

In the final solution, Regularization has been incorporated in two ways :-

- Retraining Neural Network [1] since results of Back-Propagation can be quite sensitive to weights assigned initially [2].
- Calibrating Regularization parameter. This avoids overfitting by adding a penalty term to the Cost Function resulting in minimization of weight values (and hence a simpler hypothesis).

This is demonstrated here.

3.3. Chosen Architecture

The following section contains details of network with best test error.

Parameter	Chosen value
Training Algorithm	Levenberg-Marquardt
Cost Function	Mean squared error
Layer count	2 (1 hidden + 1 output)
Perceptron units in hidden layer	6
Regularization Parameter	0.4

Table 5: Final Network Architecture. Bottom half of table displays chosen value of hyperparameters of network.

Performance Metric	Value
Training Error	5.9322%
Test Error	0%
Mean Squared error	0.0969

Table 6: Training and Test error for best Network architecture.

In order to provide all details to the reader, this is a link to a "mat-file" that can be downloaded and content can be viewed in Matlab. It contains the five best Networks found. Each contains the following (in order): Training Function, Mean squared error, Test accuracy, Regularization parameter and **Matlab network object**. The network object contains all details of weights and their interconnections learnt by the network.For convenience a script is also provided for evaluating results as in Table 6.

3.4. Conclusion

- Performance of Distance Metrics used in NN classification depends on properties of the dataset.
- k-means Clustering can largely reduce complexity but the results obtained may not be *exact*. Also, results can vary largely based on cluster centroid initialization.
- Neural Networks can be prone to overfitting for small datasets. For better generalization to out-of-sample points, Regularization is a key element.

Appendices

A. Nearest Neighbour

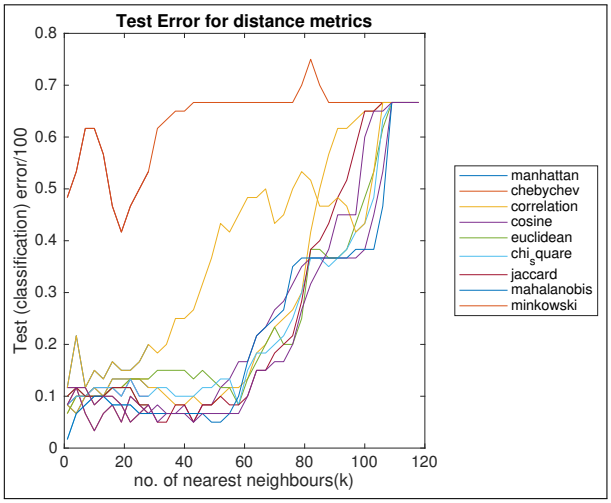

Figure 2: The worst result (75% error) is given by Jaccardi at k = 82 .The best result (1.67% error) is given by chebychev at k = 1.

.

Clearly, As k increases, the prediction by all metrics becomes the class label 2. This was also verified in Matlab variable window. This is because there are a large majority (class 1: 59, class 2:71, class 3:48) of training samples with label 2 in the given training partition. If all training points are considered as nearest neighbours (k=118), then class 2 samples will dominate the result and hence the error rate of all metrics tends to converge to 66.67% since 40 out of 60 test samples are **not** of class label = 2.

Note - All attributes have been standardized. This is really key since 3/13 features have mean value in (10,100).

Poor performance of Jaccard

standardized_set

118x13 double

	1	2	3	4	5	6	7
1	0.2294	-0.6097	0.5039	-0.5460	-0.3337	0.3147	0.3203
2	0.3419	1.5310	0.9520	-0.2937	0.1893	0.1988	0.3615
3	0.5107	-0.5611	0.6073	-0.4829	-0.1095	0.2650	0.3821
4	0.5107	-0.4832	-1.1506	-2.2489	-0.5578	0.7118	1.2475
5	0.5107	-0.4443	-0.9438	-0.7983	0.5629	1.1750	0.9693
6	0.5107	-0.1719	2.9167	1.7246	1.8330	0.5628	0.6397
7	0.5389	-0.7071	-0.9438	-1.2713	-0.1095	0.1823	0.5984
8	0.6655	0.1298	1.0209	-0.2937	0.1146	0.8441	1.2166
9	0.7217	-0.4346	-0.8059	-2.6273	0.0399	0.5959	0.7221
10	0.7780	0.3536	1.7103	0.4631	1.3847	0.8441	0.6500
11	0.7780	1.7062	-0.2889	-0.6406	0.2640	0.5794	0.5881
12	0.8343	-0.5708	1.6069	-1.2713	0.7870	0.5132	0.6397
13	0.8483	-0.2497	1.0554	-0.8613	0.1893	1.1750	1.2063
14	0.8624	-0.4930	-0.8059	-0.7983	-0.4084	0.1823	0.1348

Figure 3: Standardized training set. According to Jaccard distance formula from Matlab (see below), it assigns a distance $26 \div 26 = 1$ to *most* samples from same or different classes. The formula relies on dissimilar elements and in case of floating point data, the formula is even more error prone.

$$d = \frac{\#(x_i \neq y_i) \cap (x \cup y)}{\#(x \cup y)},$$

where x and y are any two non-zero samples and # refers to count.

B. k-means Clustering

Listing 1: Code for assigning label to a Clustering provided by Matlab kmeans().

```
% Checks if each cluster has a distribution of training points close
% to the ratio given ( class1 : 39, class 2:51 , class 3:28 ).
% It uses a thresholded majority vote to assign labels to clusters.

% cluster_index is output of Matlab kmeans(). It assigns a cluster index to
% each training sample.

function [cluster_labels] =  assignLabels(total_clusters, cluster_index,
   train_labels)

    cluster_labels = ones(1,total_clusters,1)*-1;

    % Currently, training labels have frequency {39,51,28} in class 1,2 and 3.
    % Thresholding is based on trial and observations of clustering of samples
    % returned by Matlab kmeans()
    threshold1 = 25/(total_clusters-1);
    threshold2 = 30/(total_clusters-1);
    % class 3 tends to be poorly distributed (in terms of majority) hence
    % low threshold.
    threshold3 = 4;

    for i = 1:total_clusters

        clabels = train_labels( find(cluster_index == i)) ;

        % Find majority class in given cluster.
        [label,Freq] = mode(clabels);

        % Assign label to cluster if conditions are satisfied.
        if((label == 1) && (Freq > threshold1))
             cluster_labels(i) = label;

        elseif((label == 2) && (Freq > threshold2))
             cluster_labels(i) = label;

        elseif((label == 3) && (Freq > threshold3))
             cluster_labels(i) = label;

        end

    end

end
```

REFERENCES

[1] Matlab.(2017) Improve Network Generalization. Retrieved December 15th, 2017, from Matlab: https://uk.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html?requestedDomain=uk.mathworks.com

[2] Kolen, J. F., & Pollack, J. B. (2010). Back Propagation is Sensitive to Initial Conditions. The Ohio State University. Retrieved from http://www.demo.cs.brandeis.edu/papers/bpsic.pdf