Dynamic Time Warping: Constraints on Warping

Pavle Mihajlovic, Rajat Sharma

University of Waterloo

Abstract

Classification and in particular time series classification (TSC) is an area of interest to many. Often times highly computational intensive methods are undertaken to solve these unique problems. However simple algorithms like a 1-nearest-neighbour (NN) classifier in combination with dynamic time warping (DTW) is considerably less computationally intensive and sometimes more or almost as accurate as much more intricate methods. Within this framework one can ideally increase the TSC accuracy by introducing constraints on the DTW distance calculation. In this paper we investigate constraints placed on the DTW warping window, the data, starting and ending points. With intuitive constraints one can add to the already strong performance of the unconstrained 1-NN DTW for TSC. In fact, placing constraints on the warping window can also increase the speed of the algorithm. We also confirm that increasing k-NN does not aid classification performance.

Keywords: Dynamic Time Warping, Time Series Classification, Nearest Neighbour, Sakoe-Chiba Band,

1. Introduction

Time series classification (TSC) problems arise in many fields and disciplines. TSC problems are normally much different than regular classification problems. The temporal characteristics of time series data poses a unique and different challenge than data that feature no temporal characteristics. TSC has been explored thoroughly[1]. First let us define the basics of a time series classification problem. We denote the pair {X, y}. X is a vector with n observations: $(x_1, ..., x_n)$, this is the ordered time series of data such that all entries are ordered by their time characteristic. y is a discrete class variable to which its corresponding time series vector belongs to. Thus we can create a list of m observations where $(\{X_1, y_1\}, .., \{X_m, y_m\})$.

To create a useful framework for producing predictions from a set of classified series, we want to identify discriminatory features between sets so as to be able to differentiate them into different sets.

One specific method that will be explored in this paper is dynamic time warping (DTW), which can be classified under a whole series similarity measure. Whole series TSC algorithms

February 27, 2019

quantify similarities between two whole series by some distance metric. One might reasonably expect that these algorithms form the best and most intuitive manner of comparing two series. Dynamic time warping was first introduced by Berndt and Clifford in 1994[2].

DTW in particular calculates a variable distance measure between two series and attempts to locate the closest match. This procedure is called a 1-nearest-neighbour approach. A 1-NN DTW is not very computationally intensive and can be used to search through trillions of time series subsequences in a amazingly short amount of time[3].

Through the application of constraints especially on the scope of the warping allows better prediction[4]. Methods including limiting the window size through a sakoe-chiba band can improve value of DTW distance calculation[5]. Other bands are also able to improve prediction accuracy[6].

In the next few sections we will be discussing DTW in depth. We will be covering how it is calculated and additional constraints that can be placed on the warping method. Afterwards the algorithm will be applied to a couple of TSC datasets under varying constraints, to show that a 1-NN DTW can be approved upon very quickly. A light discussion will occur near the end of the paper highlighting strengths and limitations of our analysis. Future direction will also be discussed.

2. Dynamic Time Warping

2.1. Algorithm

Given two time series vectors, x = $\{x_1, ..., x_n\}$ and y = $\{y_1, ..., y_n\}$. We can compute an arbitrary measure, usually referred to as a distance or similarity measure, d(x,y). Where this chosen measure is computed over a singular value from each set x and y. Some methods to calculate distance include, but are not limited to: manhattan distance, canberra distance or euclidean distance. Below feature the calculation for each:

Manhattan:

$$d(x_i, y_j) = |x_i - y_j|$$

Canberra:

$$d(x_i, y_j) = \frac{|x_i - y_j|}{|x_i| + |y_j|}$$

Euclidean:

$$d(x_i, y_j) = \sqrt{(x_i - y_j)^2}$$

Given a distance metric as shown above we can compute a so called distance matrix from each of the time series vectors provided at the start. We construct a n x n (length of x by length of y) matrix D. In this matrix we have each entry represented as $D_{i,j} = d(x_i, y_j)$. Where the distance is of a predetermined function, as reported above. Thus we can observe very simply that $D_{1,1}$ will provide us the distance measure between the first points in each vector.

We have a couple pieces of the algorithm. We have a distance matrix between all of the points in each time series. Each entry in the $D_{i,j}$ matrix reveals the characteristic of the distance between the ith time series value in x and the jth time series value in y. How does one extract inference from this?

If we know all the distances between all of the points through our distance matrix, we can apply a simple method to look through all the values to uncover the smallest distance path. We call this the warping path. A warping path is intrinsically a series of indexes that are a traversal of matrix D. We define the path as Path = $(\{a_1, b_1\}, .., \{a_z, b_z\})$, where the subscript refers to the nth move through the matrix, z is the number of moves, a represents the row index and b represents the column index.

We assume that this path must adhere to the following condition: $\{a_1, b_1\} = \{1, 1\}$ and $\{a_z, b_z\} = \{n, n\}$. All this entails is that the traversal path over the matrix must start in the upper left corner and end in the bottom right corner.

We add another assumption: $0 \leq a_{i+1} - a_i \leq 1$ and $0 \leq b_{i+1} - b_i \leq 1$, [1]. What this means in simpler terms is that we limit how the traversal path moves through the matrix. It implies that given a current position we can only move into an index that is directly below, directly to the right or in the bottom right index from our current one.

Now the DTW distance between these two series is the path through D that minimizes the total distance. Finding this path requires a dynamic programming procedure. However due to time and computational constraints we were not able to complete this.

Instead a much simpler and streamlined version to calculate a pseudo minimal path was used. Calculating the distance requires finding the minimal path. Instead of optimizing based on the best entire path we determined that a quick workaround should suffice. We know exactly the next spots our traversal path can lead to based on our current index. In light of this we select the next minimal distance possibility. This is done in good faith that the overall distance will not be hugely affected and that any move that is done to the detriment of the calculation should correct itself relatively quickly.

Thus we have calculated a pseudo DTW distance that should closely resemble the actual DTW distance between two series. But the question still remains, how does one make a prediction from this framework?

2.2. 1 Nearest Neighbour DTW

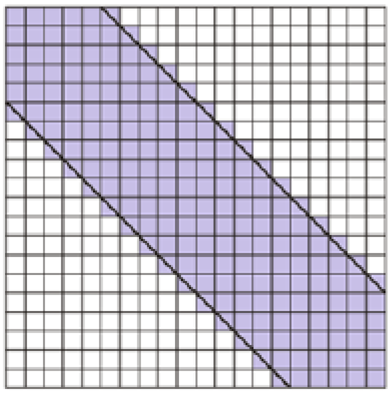
Given two sets of data a labelled train set and a unlabelled test set, we can extract predictive classifications. We take one unlabelled time series vector from the test set and compute the dtw distance as calculated above for all of the labelled time series vectors from the train set. With this we can get the minimum distance for all of these comparisons. We then assign the class of the minimum distance time series from the train set to this unlabelled time series vector from the test. We repeat the process sequentially over all unlabelled test time series vectors.

The idea here obviously being that the closest match in the train set should hypothetically mirror our test vector, thus assigning it to the same class makes sense. This can be extended to N closest neighbours. However methods to determine boundary conditions arise. The simplest solution entails picking the mode of the N closest neighbours. This introduces obvious biases within each classification task, especially the number of classes and possible class imbalance issues shall arise.

2.3. Warping Window Constraints

Here we mention a method to constraint the warping window. The warping window refers to the distance matrix, it is the "window" in which we want to allow traversal for a minimal path to occur. Normally this should not be an issue. However one can imagine scenarios where the path starts to move erratically, possibly very far away from the diagonal. Moving far away from the diagonal would be not perfect in a sense. We want to ideally compare temporally similar values or atleast ones that are relatively close to each other. If we move too far away from the diagonal, we might be creating the most minimal path, but it would have much less interpret-ability. The reason it would have less is because we want the DTW to be able to move around a bit in the window to reveal close, but similar features, but if were revealing similar features that are very far away from each other then the purpose is lost.

Thus we introduce the sakoe-chiba band. The idea of this band is to create a window size around the diagonal. A window size of 1 allows only moves down the diagonal, increasing the window size from here increases the spaces the traversal is allowed to move. Ideally this keeps the dtw distance from too far from the diagonal and keeps the distance measure realistic.



Sakoe-Chiba band

Here is a quick graphic illustrating the windowing effect[5]. The window size for this graphic is 5. A window size of 1 is the diagonal.

2.4. Starting and Ending Constraints

Earlier we defined that our DTW distance has to start at $D_{1,1}$ and should end at $D_{n,n}$. We do not need to adhere to this constraint. In practicality it makes sense for us to possibly allow for some variable starting index and ending index. The intuitive idea behind this is to allow our warping to forgo some possible minute differences in alignment at the very beginning and/or the very end of the warping.

Instead of starting at (1,1) we allow a variable to be set to indicate whether we want to have a "minimal" start. This means we are starting at index (q,1) where q is corresponding to the minimal distance entry in the first column.

Instead of ending at (n,n) we allow a variable to be set to indicate we want to end "minimally". What this means is whenever the very right end of the distance matrix is reached we stop calculating the distance.

If a combination of these two options are combined with the sakoe-chiba band it should yield powerful results. Hypothetically it should minimize slight and cosmetic differences at the start of time series where the time series' might be just misaligned by a couple of time points. This would be useful for say scenarios in which one has highly similar sets of data but there occured some misalignment in the method of measurement.

2.5. Z-Normalization

One can also choose to transform each series by Z-normalization. There seems to be consensus that standardizing the data should enhance performance[7]. This is especially prevalent when we want to compute the sliding window method of two vectors of unequal lengths.

2.6. Unequal Length Time Series

The above algorithm summarizes how the DTW works for two equal length vectors. However what if one were presented with two vectors that were not of equal length? For the DTW distance to be meaningful we want the "query" vector to be the smaller vector. The reasoning behind this is intuitive, we can take a smaller time series and compare it to a larger timer series, but looking at equal subsequences sequentially over all possible sliding windows, holding the temporal constraints the same.

The idea follows that we have two vectors, x = $\{x_1, ..., x_n\}$ and y = $\{y_1, ..., y_m\}$, where $m > n$. Thus using a "sliding-window" we can compute the DTW distance as above using equal subintervals from the y vector, thus creating a DTW distance from x = $\{x_1, ..., x_n\}$ and y = $\{y_1, ..., y_n\}$, then another DTW distance from x again and y = $\{y_2, ..., y_{n+1}\}$, until we exhaust all possible "sliding windows". Given these distances we can find the minimal matching sequence.

3. Analysis

3.1. Datasets

In the analysis, we take a look at 4 different datasets.

The olive dataset uses chemometrics and spectrography on different olive oils to create a frequency spectrum. We then use these measures to classify the country of origin for each sample of olive oil. There were 4 classes of olive oils representing different countries. The set was 30 observations for the training and test sets, with 570 measurements (variables) each.

Similarly, the coffee dataset also uses spectrography to create a frequency spectrum which we take measures from. The data contains only 2 classes which refer to two different types of coffee beans; Robusta and Arabica respectively. The training set and test sets both had 28 observations, with 286 points each.

The Wine dataset uses infrared spectrography to distinguish between wine types. In this dataset there were 2 wine classes. The test dataset contained 54 observations, while the training had 57, both with 234 points.

The BeetleFly dataset uses skeleton-based descriptors to classify whether the given image is a beetle or a fly. The data contains 2 classes corresponding to beetle and fly. The training and test datasets each contained 20 samples of bugs, with 512 points.

All of these datasets were derived and downloaded from the UCR Time Series Classification Archive. [8].

3.2. Results

3.2.1. Introduction

As for the results, we analyze how well our DTW method classifies each set of data. By using a training set of data with known classes, we can take the distance using DTW, of vectors in our test set and classify them accordingly. We then compare the results of actual classes of the test data with the results predicted by the DTW method.

We also explore different choices for constraint parameters on each dataset and check how it effects the classification. We choose from the window size warping constraint w, and indicating whether or not the calculated distance matrix needs to start/end at the corners.

3.2.2. K-NN unconstrained

For testing different values of the K nearest neighbours, we selected values of k=1,3,5,10,and 15 unconstrained by warping parameters. The weighting of the K-NN was determined by taking the most common class in the K neighbourhood using the training dataset, and assigning that class to our test vector. If no mode was found in the classification, we assign the test vector the class in the training set to which it had the lowest distance from.

The results from using a K-NN DTW on all datasets used is given in the table:

K	Olive	Coffee	Wine	BeetleFly
1	83.3	96.4	66.6	55
3	86.7	92.8	64.8	60
5	86.7	89.2	61.1	60
10	80	82.1	55.5	70
15	73.3	85.7	51.8	60

From our analysis, we determined that with unconstrained DTW, the 1-NN tends to outperform the other K-NN values. It should be noted this is not the case for the BeetleFly data. This could be due to the different nature of the BeetleFly set in comparison to the other 3. It should also be noted that oliveOil performs better when set to 2 and 3 nearest neighbours, but then takes a massive decline for values afterwards.

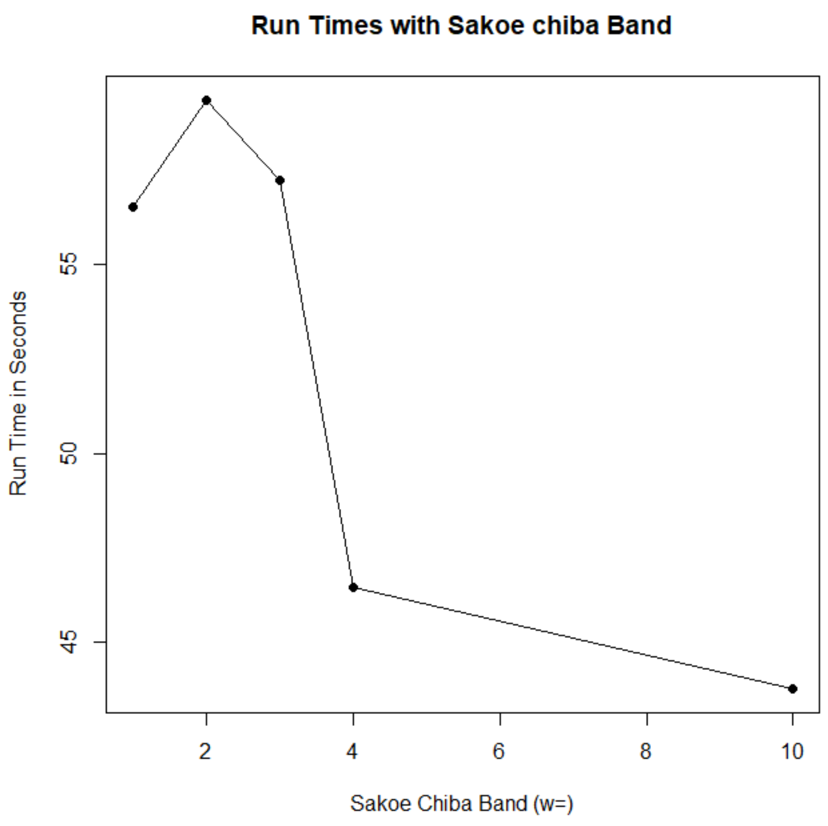
A possible reason why the BeetleFly dataset performs better at higher K is because of the clear distinction in skeletal features of each type of bug. Using the nearest 10-15 neighbours makes it so that the drastically different vectors in the training set do not match with our test vector. "Outliars" are somewhat avoided in a sense. It is easier to see drastic differences in skeletal measures in contrast to differences in spectrography for certain compounds.

Looking at our figures (appendix), we can clearly see a trend downwards in the correct classification rate for the most sets.

3.2.3. Sakoe Chiba and Starting/ending points

We now test how well our DTW method works when constraining it with some parameters. The parameters being the Sakoe Chiba band and the starting/ending points of the cost matrix. It is important to note that when w=1, we are just summing the distance down the diagonal of the cost Matrix .

Before we see how well it classifies the data, lets take a look at another use of the Sakoe-Chiba band; reducing computation time.



Creating a Sakoe-chiba band reduces the path our algorithm is allowed to warp around. We can see from our figure on the BeetleFly dataset, as we increase our warping path (w increases), we see and dowward trend. This is due to the fact that our algorithm can ”move” around more in our distance matrix. As the data gets increasingly larger, this computation time can make a crucial difference. Eventually the graph would converge to a value for large w.

All of the results in the upcoming parts are summarized in the appendix. Please refer to them as a guide for the analysis of the performance.

For the olive oil dataset, we see that changing the warping path and start/end points does not make a difference in the classification rate. For all values of w and TRUE/FALSE values for open.end and open.close, none of them seemed to impact the rate of 83.3%. The unconstrained 1-NN DTW performs at the exact same rate of 83.3%.

For the coffee data, changing the parameters makes a slight difference but only for the worse. When we set our band size to 2 (w=2), the accuracy decreases regardless of the starting/ending position. This dataset does not improve when adding more constraints in comparison to the unconstrained 1-NN DTW.

For the wine data, we can enhance classification prediction by changing the warping constraints. We observed that for all warping constraints w, the classification rate changes regardless of the starting/ending position. It seems to be optimized when we choose a warping constraint of w=5. It should be noted that this performs better than the unconstrained 1-NN DTW, which performs at 66.6%. The best constrained DTW performs as 68.5

When changing the warping parameters on the BeetleFly set, it seems to have a positive effect on the classification. We immediately observed that all combinations of the warping parameter w and the starting/ending position of the distance matrix performs better than the unconstrained 1-NN DTW which performs at 55%. Where our best performing model is at 70%.

Thus through this analysis we can highlight that it is possible that within constraints placed on the 1-NN DTW to increase performance either by a small amount or larger amounts. However hyperparameter tuning like this should lead to enhanced performance over most sets.

4. Discussion

From the analysis, we can see that a 1-NN unconstrained DTW seems to almost always performs better in contrast to higher unconstrained K values; for the most part. We can also see that the 1-NN DTW can be improved upon not by increasing the neighbourhood, but rather changing the warping constraints when calculating the distance from vectors.

Following the results of the Sakoe Chiba and starting/ending position constraints, we notice a few trends. It is easy to see that increasing the parameter w almost always tends to help or not affect the classification rate. It should also be noted that changing whether to start or end at the corners of our cost matrix have a negligible affect on our classification rate, the only meaningful parameter here seems to be w.

Looking at other possible constraining methods, we found a different idea that may aid in improving the algorithm. The suggestion being the Silhouette index (SI). This band attempts to determine how well clustered together the data is. "Well-clustered" datasets should have identical clusters (smaller distances) grouped closer together, while unlike clusters should be farther away from each other. We then constrain the distance matrix so that it can only take warping paths where the data is well clustered. This avoids "traps" in the distance matrix where non-similar clusters are grouped closer, which in turn increases our total distance measured, possibly skewing results. [6]

Note that since only 4 datasets were used, the results may drastically changed if more datasets were used instead. In the future, we could improve by using a larger variety of datasets of different types, and using different constraint parameters like the SI. Having more observations in our data may also serve useful as it is easier to see smaller differences in classification rates when having larger samples (100 beetles vs 20 beetles for example). By testing these parameter combinations again, it is likely we would get an even better method that yields better results. It is important to note that we can build even further upon this by seeing what parameters work best for certain types of data (i.e Frequency Spectrum Data vs Image data).

5. References

[1] Bagnall, A., Lines, J., Bostrom, A. et al. Data Min Knowl Disc (2017) 31: 606. https://doi.org/10.1007/s10618-016-0483-9

[2] Berndt, Donald J., Clifford J., Using Dyanmic Time Warping to Find Patterns in Time Series https://www.aaai.org/Papers/Workshops/1994/WS-94-03/WS94-03-031.pdf

[3] Rakthanmanon, T., Campana, B., Mueen, A., et al. Searching and mining trillions of time series subsequences under dynamic time warping: http://www.cs.ucr.edu/ eamon-n/SIGKDD_trillion.pdf

[4] Zoltan, Z., Kurbalija V., Radovanovic, M., et al. Impact of the Sakoe-Chiba Band on the DTWTime-Series Distance Measure for kNN Classification:
https://pdfs.semanticscholar.org/5c05/9916131c0b4066bb9c3b2d84fdbed5cbb3f6.pdf

[5] Cassisi, C., Montalto, P., Aliotta M., et al. Similarity Measure and Dimensionality Reduction Techniques for Time Series Data Mining,
https://www.researchgate.net/publication/233751236_Similarity_Measures_and_Dimensionality _Reduction_Techniques_for_Time_Series_Data_Mining

[6] Niennattrakul, V., Ratanamahatana, C A., Learning DTW Global Constraint for Time Series Classification: https://arxiv.org/pdf/0903.0041.pdf

[7] Mueen A., Keogh E., Extracting Optimal Performance from Dynamic Time Warping: Slide Show, http://www.cs.unm.edu/ mueen/DTW.pdf

[8] http://www.cs.ucr.edu/ eamonn/time_series_data/

Appendix A. Section in Appendix

Results for each parameter choice of w = window size (numerical) and (open.start/open.end) (T and F values):

Appendix A.1. Tables:

(open.start/open.end) indicates each possible value for the parameters
F/F indicates FALSE/FALSE
T/F indicates TRUE/FALSE
F/T indicates FALSE/TRUE
T/T indicates TRUE/TRUE
w is the window size, w=1 is the diagonal

Results for Wine

W	F/F	T/F	F/T	T/T
1	64.8	64.8	64.8	64.8
2	61.1	61.1	61.1	61.1
3	62.9	62.9	62.9	62.9
5	68.5	68.5	68.5	68.5

Results for Coffee

W	F/F	T/F	F/T	T/T
1	96.4	96.4	96.4	96.4
2	92.8	92.8	92.8	92.8
3	96.4	96.4	96.4	96.4
5	96.4	96.4	96.4	96.4

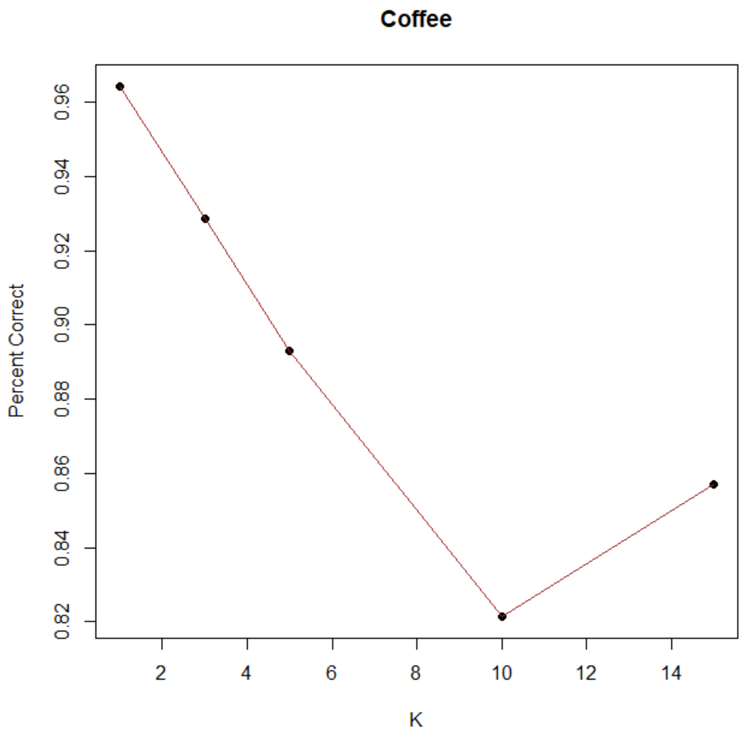
Results for OliveOil

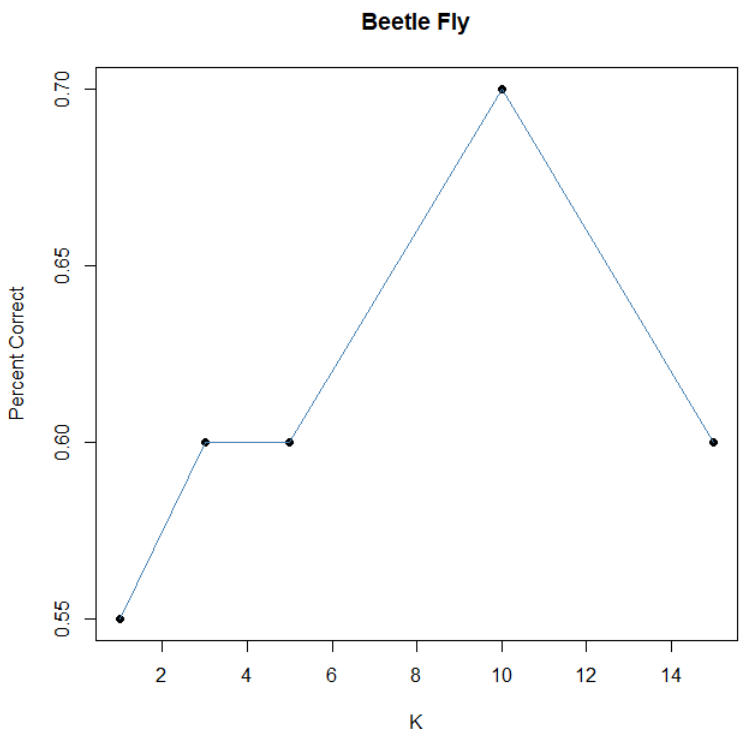
W	F/F	T/F	F/T	T/T
1	83.3	83.3	83.3	83.3
2	83.3	83.3	83.3	83.3
3	83.3	83.3	83.3	83.3
5	83.3	83.3	83.3	83.3

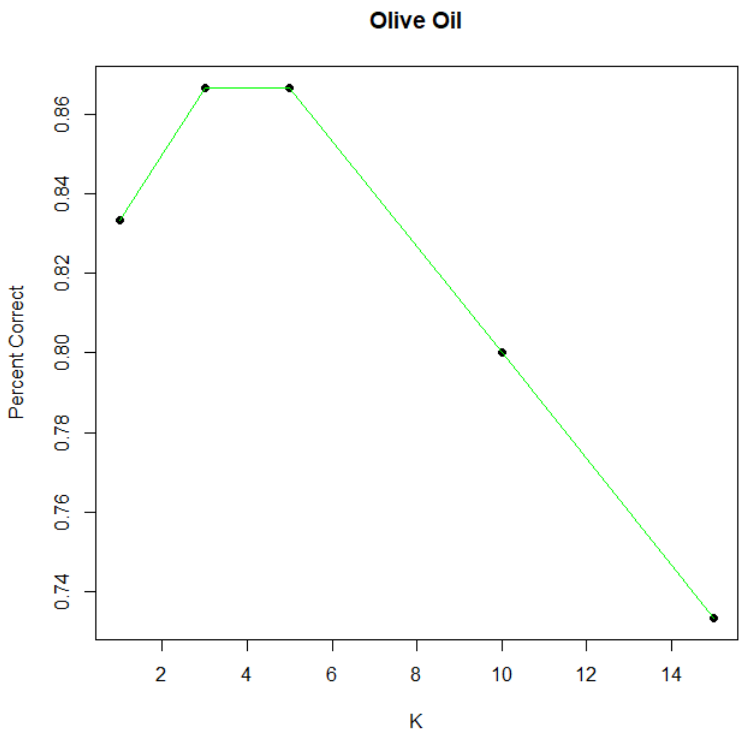
Results for BeetleFly

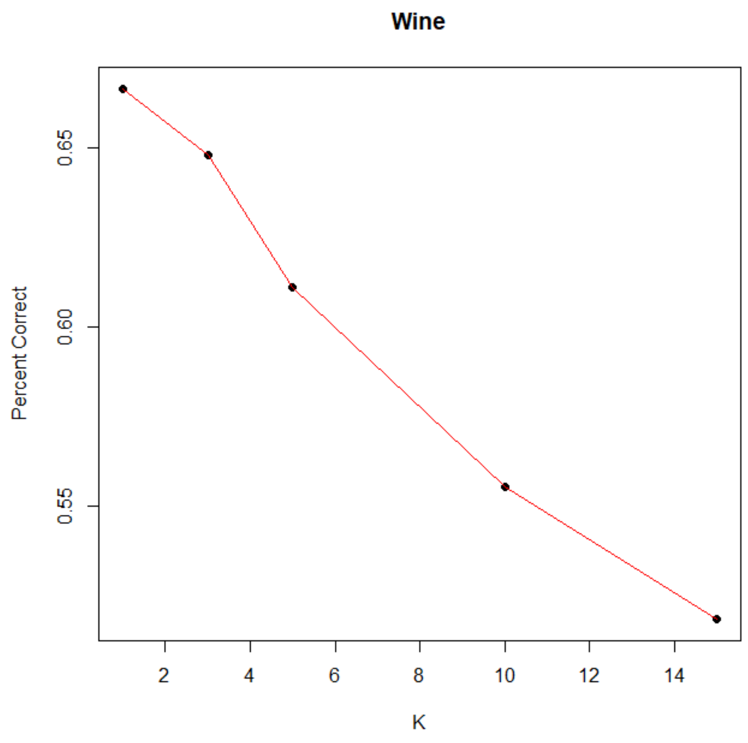
W	F/F	T/F	F/T	T/T
1	65	65	65	65
2	65	65	65	65
3	70	70	70	70
5	70	70	70	70

Classification Rates for Different sets:




Beetle Fly
Percent Correct
0.55
0.60
0.65
0.70
2
4
6
8
10
12
14
K


Olive Oil
Percent Correct
0.74
0.76
0.78
0.80
0.82
0.84
0.86
2
4
6
8
10
12
14
K



Extra R Code that was used:

```
## Loading each dataset:

coffee.train <- read.csv("coffee_train.csv",header=T)

coffee.test <- read.csv("coffee_test.csv",header=T)


wine.test <- read.csv("wine_test.csv",header=T)

wine.train <- read.csv("wine_train.csv",header=T)


## Wine data with only observations

wine.test.clean <- wine.test[,c(-1,-236)]

wine.train.clean <- wine.train[,c(-1,-236)]
```

```
## Remove ID and target variables, only matrix of observations

coffee.test.clean <- coffee.test[,c(-1,-288)]

coffee.train.clean <- coffee.train[,c(-1,-288)]


## Dataset with multiple classes, olive data

olive.test <- read.csv("olive_test.csv",header=T)

olive.train <- read.csv("olive_train.csv",header=T)


## Clean up the data

olive.test.clean <- olive.test[,c(-1,-572)]

olive.train.clean <- olive.train[,c(-1,-572)]


beetle.test <- read.csv("beetle_test.csv",header=T)

beetle.train <- read.csv("beetle_train.csv",header=T)


beetle.test.clean <- beetle.test[,c(-1,-514)]

beetle.train.clean <- beetle.train[,c(-1,-514)]



## "Cleaning" the data

data.test  [,c(-1, ncol(data.test)]
data.train[,c(-1, ncol(data.train]

## Gather the classification rate for K-NN


coffee_percent <-percents(coffee.test.clean,
coffee.train.clean,coffee.train,coffee.test)
```

```
olive_percent <-percents(olive.test.clean,
olive.train.clean,olive.train,olive.test)

wine_percent <- percents(wine.test.clean,
wine.train.clean,wine.train,wine.test)

beetle_percents <- percents(beetle.test.clean,
beetle.train.clean,beetle.train,beetle.test)


## Code for the Plots K-NN unconstrained plots

## Performance of k-NN with our data sets, NO warping constraints

plot(x=c(1,3,5,10,15),y=wine_percent,
pch=19,xlab="K", ylab="Percent Correct", main="Wine")
lines(x=c(1,3,5,10,15), y=wine_percent,col="Red")


plot(x=c(1,3,5,10,15),y=coffee_percent,
pch=19,xlab="K", ylab="Percent Correct",main="Coffee")
lines(x=c(1,3,5,10,15), y=coffee_percent, col="Brown")

plot(x=c(1,3,5,10,15),y=olive_percent,
pch=19, xlab="K", ylab="Percent Correct",,main="Olive Oil")
lines(x=c(1,3,5,10,15), y=olive_percent,col="green")


plot(x=c(1,3,5,10,15),y=beetle_percents,
pch=19,xlab="K", ylab="Percent Correct", main="Beetle Fly")
lines(x=c(1,3,5,10,15), y=beetle_percents, col="Steel blue")




## Code for the Run-Time Plot using Sakoe Chiba


a<- system.time(classify.sak(olive.test.clean,
olive.train.clean,
olive.train,1))
```

```
b<- system.time(classify.sak(olive.test.clean, olive.train.clean,
olive.train,2))

c<-system.time(classify.sak(olive.test.clean, olive.train.clean,
olive.train,3))

d<-system.time(classify.sak(olive.test.clean, olive.train.clean,
olive.train,4))

e<- system.time(classify.sak(olive.test.clean, olive.train.clean,
olive.train,10))



## Vector of run times

run_times <- c(as.numeric(a[3]),as.numeric(b[3]),
as.numeric(c[3]),as.numeric(d[3]),as.numeric(e[3]))


plot(y=run_times,x=c(1,2,3,4,10),pch=19,
xlab="Sakoe␣Chiba␣Band␣(w=)",ylab="Run␣Time␣in␣Seconds",
      main = "Run␣Times␣with␣Sakoe␣chiba␣Band")
lines(x=c(1,2,3,4,10), y=run_times)




Some useful results:

## Create vector of classification rates w every parameter
## combination, w is on rows, T/F end/start for cols

wine_best <- best_w(wine.test.clean,wine.train.clean,
wine.train,wine.test)

coffee_best <- best_w(coffee.test.clean,
coffee.train.clean,coffee.train,coffee.test)

olive_best <- best_w(olive.test.clean,olive.train.clean,
olive.train,olive.test)
```

```
beetle_best <- best_w(beetle.test.clean,
beetle.train.clean,beetle.train,beetle.test)


# Use these vectors to create 4x4 matricies to easily see
the classification rate with every parameter combination

wine_best_m <- matrix(wine_best, nrow=4, ncol=4, byrow=T)


coffee_best_m <- matrix(coffee_best, nrow=4,ncol=4,byrow=T)



olive_best_m <- matrix(olive_best, nrow=4,ncol=4,byrow=T)


beetle_best_m <- matrix(beetle_best,nrow=4,ncol=4,byrow=T)



## Match.count -> counts number classified correctly
## Compares results to the actual classes

match.count(class, test.raw)
```