Parkinson UPDRS Prediction with K-Nearest Neighbour

Barışcan Tunalı
Big Data Masters Student
Istanbul Bahcesehir University,
bariscantunali@outlook.com

ABSTRACT

In this paper, I will explain my study on Parkinson classification using K-NN.

Tracking Parkinson's disease (PD) symptom progression often uses the Unified Parkinson's Disease Rating Scale (UPDRS), which requires the patient's presence in clinic[1].

Our problem is predicting continious value (total UPDRS) from continious variables which will be explained later. UPDRS score was calculated by patient metrics. Main dataset has no test set so I need to extract our test set myself.

I applied 3 different distance measures which were eucliean, manhattan, minkowski(order of two) to compare results and take K as 5 since dataset is very small.

Categories and Subject Descriptors

[**Computing methodologies**]: Machine learning—*Machine learning algorithms; Ensemble methods*

Keywords

K Nearest Neighbour, K-NN,KNN, Data Mining, Classification, Prediction, Parkinson, UPDRS

1. INTRODUCTION

Neurological disorders affect people profoundly and claim lives at an epidemic rate worldwide. Parkinson‟s disease (PD) is the second most common neurodegenerative disorder after Alzheimer‟s [2], and it is estimated that more than one million people in North America alone are affected [3]

Tricky part of dataset is also its time series. I treated data every row is like other patient. Becasue with time this illness also proceeds. I treated time as another property.

RecSys '15 Challenge, September 16-20 2015, Vienna, Austria
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3665-9/15/09 $15.00
DOI: http://dx.doi.org/10.1145/2813448.2813516

The paper is organized as follows: In Section 2, we briefly describe the dataset and data preparation. Next, in Section 3, we explain our classification approach in detail. Experimental results and comparisons with other algorithms are provided in Section 4. Finally, Section 5 concludes the paper.

2. DATASET AND FEATURES

Parkinson dataset constists of 5875 values, I had 21 attributes, one of them was total UPDRS.

My attributes are : [FixedAcidity], [VolatileAcidity], [CitricAcid], [ResidualSugar], [Chlorides], [FreeSulfurDioxide], [TotalSulfurDioxide], [Density], [PH], [Sulphates], [Alcohol], [Quality]

Attribute name: max value – min value – different values
age : 85 - 36
test_time : 215.49 - 4.2625 - 23
motor_UPDRS : 39.511 - 5.0377 - 2442
total_UPDRS : 54.992 - 7 - 1080
Jitter_Pct : 0.09999 - 0.00083 - 1305
Jitter_Abs : 0.00044559 - 2.25E-06 - 4105
Jitter_RAP : 0.05754 - 0.00033 - 853
Jitter_PPQ5 : 0.06956 - 0.00043 - 840
Jitter_DDP : 0.17263 - 0.00098 - 1703
Shimmer : 0.26863 - 0.00306 - 3581
Shimmer_dB : 2.107 - 0.026 - 852
Shimmer_APQ3 : 0.16267 - 0.00161 - 2664
Shimmer_APQ5 : 0.16702 - 0.00194 - 2850
Shimmer_APQ11 : 0.27546 - 0.00249 - 3283
Shimmer_DDA : 0.48802 - 0.00484 - 4223
NHR : 0.74826 - 0.000286 - 5532
HNR : 37.875 - 1.659 - 4780
RPDE : 0.96608 - 0.15102 - 5430
DFA : 0.8656 - 0.51404 - 5282
PPE : 0.73173 - 0.73173 - 4777

Except age and test time all of the attributes were continious so I standardized them with mean stardardization ($x\text{-}x_{mean}$ / x_{stddev}).

In order to make my program more flexible I created a class and handled all data operations, test train seperations, standardizations etc. On presentation layer you only create class and call `Initialize()` function. After that you only access necessary train and test sets and send them to appropriate classification algorithms.

```
ParkinsonTrainSet = dprp.ParkinsonTrainSet;
ParkinsonTestSet = dprp.ParkinsonTestSet;
```

```
KNN KNNRParkinson = new KNN(ParkinsonTrainSet);
KNNRParkinson.GetIndividualScore(ParkinsonTestSet);
```

Data preparation class structure as below :

```
public void Initialize()
{
   ExtractData();
  TransformData();
  NormalizeData(ParkinsonOriginalSetTransformed);
  SeperateTestData(ParkinsonOriginalSetTransformed);
}
```

ExcractData() : Dataset was in .csv format so I made a reader function that reads file, creates class, converts string values to appropriate type and bind them into class. Appends this class object into Parkinsons List.
TransformData() : Gets list of different data classes and converts them to common Parkinson.cs class.
NormalizeData() : Calculates standart deviations, means and standardizes each column except quality score.
SeperateTestData() : Gets normalized data and randomly seperates %20 of data as test and %80 as train.

3. METHOD

I implemented K-NN with 5 neighbours.

My K-NN class has 5 functions :

```
public double[] GetIndividualScore(Parkinson _Parkinson)
public List<double[]> GetIndividualScore(List<Parkinson>
_Parkinson)
private double Euclidean(double x, double y)
private double Manhattan(double x, double y)
private double Minkowski(double x, double y, double order)
```

As we can see in implementations we have batch versions of each function. Distance functions doesn't calculate cumulative root, so we do it on caller side after summing each attributes distances. They are only mini calculation functions for clean code. There is example euclidean distance function call below, we have also similar Manhattan and Minkowski calls.

Example call for euclidean distance :

```
CumulativeEuclideanDistance = Math.Sqrt(
Euclidean(x.age, _Parkinson.age) +
Euclidean(x.test_time, _Parkinson.test_time) +
Euclidean(x.Jitter_Pct, _Parkinson.Jitter_Pct) +
Euclidean(x.Jitter_Abs, _Parkinson.Jitter_Abs) +
Euclidean(x.Jitter_RAP, _Parkinson.Jitter_RAP) +
Euclidean(x.Jitter_PPQ5, _Parkinson.Jitter_PPQ5) +
Euclidean(x.Jitter_DDP, _Parkinson.Jitter_DDP) +
Euclidean(x.Shimmer, _Parkinson.Shimmer) +
Euclidean(x.Shimmer_dB, _Parkinson.Shimmer_dB) +
Euclidean(x.Shimmer_APQ3, _Parkinson.Shimmer_APQ3) +
Euclidean(x.Shimmer_APQ5, _Parkinson.Shimmer_APQ5) +
Euclidean(x.Shimmer_APQ11, _Parkinson.Shimmer_APQ11) +
Euclidean(x.Shimmer_DDA, _Parkinson.Shimmer_DDA) +
Euclidean(x.NHR, _Parkinson.NHR) +
Euclidean(x.HNR, _Parkinson.HNR) +
Euclidean(x.RPDE, _Parkinson.RPDE) +
Euclidean(x.DFA, _Parkinson.DFA) +
Euclidean(x.PPE, _Parkinson.PPE));

x.temp1 = CumulativeEuclideanDistance;
```

I didn't use any c# library or dll to implement knn itself but I used lambda expressions to avoid code complexity. After calculating these distance measures I used these expressions to sort and take 5 nearest neighbours and get average :

```
double[] finalScores = new double[4];
finalScores[0] = _Parkinsons.OrderBy(x => x.temp1).Take(5).ToList().Average(zy =>
zy.total_UPDRS);
finalScores[1] = _Parkinsons.OrderBy(x => x.temp2).Take(5).ToList().Average(zy =>
zy.total_UPDRS);
finalScores[2] = _Parkinsons.OrderBy(x => x.temp3).Take(5).ToList().Average(zy =>
zy.total_UPDRS);
finalScores[3] = (int)_Parkinson.total_UPDRS;
```

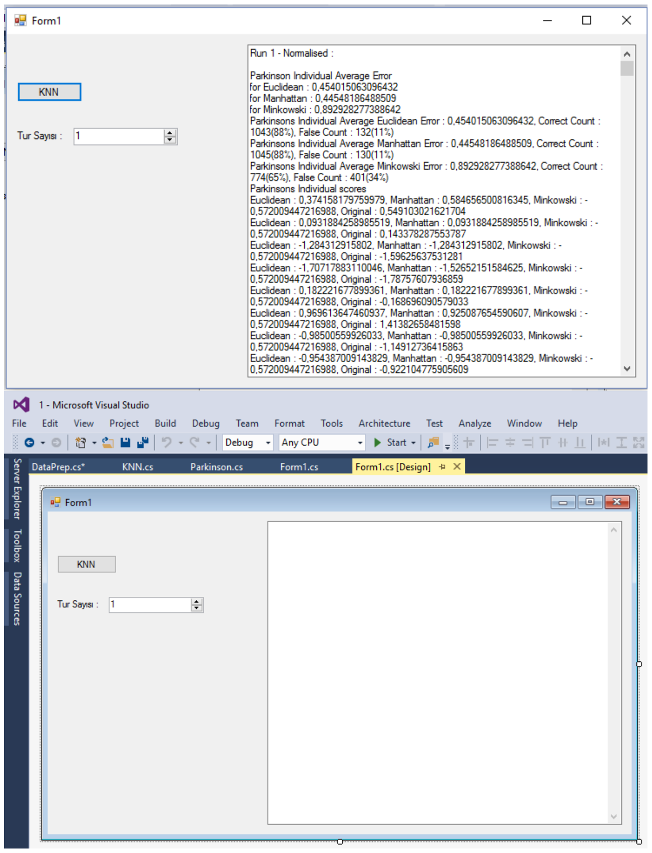
I kept these scores on wine class to calculate error and compare results.

4. EXPERIMENTS

I implemented 10 run loop, which creates new class, new train and test sets. On presentation layer I only call knn class and print out results into a file. On results I counted correct predictions, I decided that if Abs(prediction-real score)<=1 then I took it as correct prediction else false prediction.

Also in order to see difference between normalized and unnormalized results I call same function with no normalization.

If necessary you can change the experiment count from GUI as shown below. It writes data into txt file also. But shows in the gui as well:



```
Run 1 - Normalised :

Parkinson Individual Average Error
for Euclidean : 0,454015063096432
for Manhattan : 0,44548186488509
for Minkowski : 0,892928277388642
Parkinsons Individual Average Euclidean Error : 0,454015063096432, Correct Count :
1043(88%), False Count : 132(11%)
Parkinsons Individual Average Manhattan Error : 0,44548186488509, Correct Count :
1045(88%), False Count : 130(11%)
Parkinsons Individual Average Minkowski Error : 0,892928277388642, Correct Count :
774(65%), False Count : 401(34%)
Parkinsons Individual scores
Euclidean : 0,374158179759979, Manhattan : 0,584656500816345, Minkowski : -
0,572009447216988, Original : 0,549103021621704
Euclidean : 0,0931884258985519, Manhattan : 0,0931884258985519, Minkowski : -
0,572009447216988, Original : 0,143378287553787
Euclidean : -1,284312915802, Manhattan : -1,284312915802, Minkowski : -
0,572009447216988, Original : -1,59625637531281
Euclidean : -1,70717883110046, Manhattan : -1,52652151584625, Minkowski : -
0,572009447216988, Original : -1,78757607936859
Euclidean : 0,182221677899361, Manhattan : 0,182221677899361, Minkowski : -
0,572009447216988, Original : -0,168696090579033
Euclidean : 0,969613647460937, Manhattan : 0,925087654590607, Minkowski : -
0,572009447216988, Original : 1,41382658481598
Euclidean : -0,98500559926033, Manhattan : -0,98500559926033, Minkowski : -
0,572009447216988, Original : -1,14912736415863
Euclidean : -0,954387009143829, Manhattan : -0,954387009143829, Minkowski : -
0,572009447216988, Original : -0,922104775905609
```

5. CONCLUSIONS

Here are 2 run results for knn :

Run 1 - Normalised :

Parkinson Average Error
Euclidean Error : 0,456349628096565,
Correct Count : 1040(88%),
False Count : 135(11%)

Manhattan Error : 0,445386029940336,
Correct Count : 1058(90%),
False Count : 117(9%)

Minkowski Error : 0,859582537094246,
Correct Count : 781(66%),
False Count : 394(33%)

Run 1 - No Normalisation :

Parkinson Average Error
Average Euclidean Error : 0,227171508917367,
Correct Count : 2299(97%),
False Count : 51(2%)

Manhattan Error : 0,205369305889588,
Correct Count : 2307(98%),
False Count : 43(1%)

Minkowski Error : 1,08376043070654,
Correct Count : 1195(50%),
False Count : 1155(49%)

Run 2 - Normalised :

Parkinson Average Error
Euclidean Error : 0,439425687625529,
Correct Count : 1052(89%),
False Count : 123(10%)

Manhattan Error : 0,426017156962146,
Correct Count : 1064(90%),
False Count : 111(9%)

Minkowski Error : 0,822458203607021,
Correct Count : 759(64%),
False Count : 416(35%)

Run 2 - No Normalisation :

Parkinson Average Error
Euclidean Error : 0,224787383895685,
Correct Count : 2304(98%),
False Count : 46(1%)

Manhattan Error : 0,201287239606582,
Correct Count : 2322(98%),
False Count : 28(1%)

Minkowski Error : 0,863552714962304,
Correct Count : 1214(51%),
False Count : 1136(48%)

Results are interesting with normalisation and without normalisation. Normalizin process decreases prediction rate about %10 percent. But Minkovski measure gives always bad results compared to manhattan and euclidean. But normalisation increases prediction rate on Minkowski

There is a trade off between these algorithms knn's runtime is very high, it compares test wine with every wine in train set and then sorts them so run time is about $O(n + O_{sort}(n))$ so on real time applications or even nightly jobs its cost very high on large datasets. And average error ((prediction-actual)/count) is about 0.2 for unnormalised data, 0.4 for normalised data. Minkowski about 0.8 for both

References

[1] Athanasios Tsanas, Max A. Little, Patrick E. McSharry, Lorraine O. Ramig (2009),
'Accurate telemonitoring of Parkinson.s disease progression by non-invasive speech tests', IEEE Transactions on Biomedical Engineering
[2] A.E. Lang, A.M. Lozano. Parkinson‟s disease – First of two parts, *New England Journal Medicine*, 339, 1044-1053, 1998
[3] M. Rajput, A. Rajput, A.H. Rajput. Epidemiology (chapter 2). In *Handbook of Parkinson's disease*, edited by R. Pahwa and K. E. Lyons, 4th edition, Informa Healthcare, USA, 2007