CS 760: Homework 3
Training Perceptrons and Using Kernels

Sajika Gallege

Login: sgallege

4/7/2010

Perceptron Learning

I have created a neural-network system that can learn from data formatted in the same manner as used in HWs 1 and 2. It is neural networks with no hidden units

The different data types of are preprocessed by the DataReader class as follows before the training and test sets are sent to the Perceptron. The new feature set if well suited for the inputs of numeric optimizer such as a Perceptron

Discrete:

The 1-of-n representation is used, so a new boolean feature is added for each distant value of the discrete feature. Resulting in the following feature set

```
Manufacture_Country=UK
Manufacture_Country=Germany
Manufacture_Country=France
Manufacture_Country=USA
Manufacture_Country=Italy
Manufacture_Country=Japan
Manufacture_Country=Korea
Manufacture_Country=Sweden
Trans_Type=A
Trans_Type=AM
Trans_Type=CVT
Trans_Type=M
Trans_Type=OT
Trans_Type=SA
Drive_Type=4
Drive_Type=A
Drive_Type=F
Drive_Type=P
Drive_Type=R
Vehicle_Class=S_Car
Vehicle_Class=M_Car
Vehicle_Class=L_Car
Vehicle_Class=S_Truck
Vehicle_Class=L_Truck
Vehicle_Class=Van
Vehicle_Class=SUV
Variable_Valve_Timing=Y
Variable_Valve_Timing=N
Air_Aspiration_Method=NA
Air_Aspiration_Method=TC
Air_Aspiration_Method=SC
```

Continuous:

The thermometer representation is used for continuous features. The number digits can be specified. The DataReader splits the difference between the upper bound and the lower bound as specified in the .names file in to equal intervals. A new feature is boolean feature is added after comparing the continuous value against the intervals. For 10 digits it gives the following features

```
Engine_Displacement[0]
Engine_Displacement[1]
Engine_Displacement[2]
Engine_Displacement[3]
Engine_Displacement[4]
Engine_Displacement[5]
Engine_Displacement[6]
Engine_Displacement[7]
```

```
Engine_Displacement[8]
Engine_Displacement[9]
Num_Cylynders[0]
Num_Cylynders[1]
Num_Cylynders[2]
Num_Cylynders[3]
Num_Cylynders[4]
Num_Cylynders[5]
Num_Cylynders[6]
Num_Cylynders[7]
Num_Cylynders[8]
Num_Cylynders[9]
Num_Gears[0]
Num_Gears[1]
Num_Gears[2]
Num_Gears[3]
Num_Gears[4]
Num_Gears[5]
Num_Gears[6]
Num_Gears[7]
Num_Gears[8]
Num_Gears[9]
```

Output:

The output is normalized to [+1, -1] and the user can specify which symbol maps to positive and which symbol maps to negative. The is also a default mapping that maps the following symbols (1, +1, +, y, yes) to positive

The Perceptron automatically adjust the learning rate (eta) with the following default settings. It also uses 20% of the training set as a tuning set, for use in "early stopping" to prevent overfitting. The test runs are limited to 1000 epochs. The Perceptron will utilize weight decay if specified. The default settings of the Perceptron are given below they can be adjusted by the user.

```
Eta= 0.25;
Eta_Step= 0.05;
Eta_max = 1.00;
Eta_min = 0.000001;
Bias= 1;
weightDecay= true;
Lambda = 0.01;
EtaAdjustSize= 100;
```

The HW3 takes the following parameters

```
HW3 samples.names train.data test.data modeFlag
```

```
modeFlag
0: Perceptron
1: Gaussian Kernel
2: Voted Perceptron
3: Gaussian Kernel with Voted Perceptron
4: Info. Gain Scaled Gaussian Kernel
5: 10 fold Cross validation on all algorithms
```

Perceptron output on cross validation fold#1 with default settings

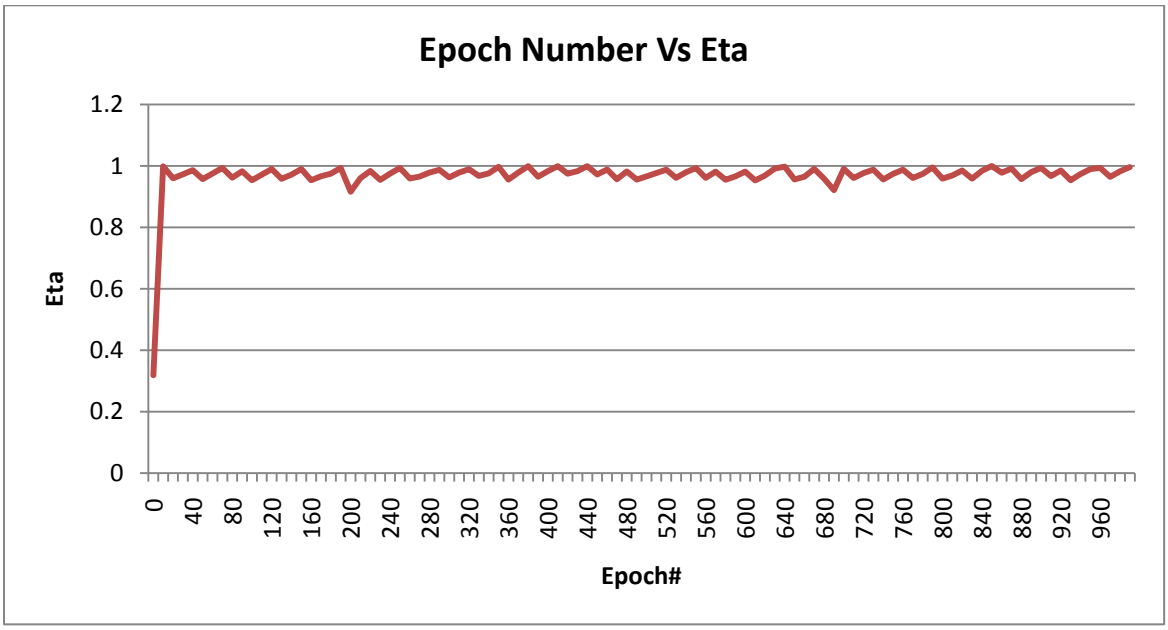
```
HW3 cars.names train1.data test1.data 0
```

epoch#	Eta
0.0	0.31827271464843754
10.0	0.9985838408772587
20.0	0.9593807580832757
30.0	0.972659793215125
40.0	0.9861226268780029
50.0	0.9569423528502772
60.0	0.9750568274218392
70.0	0.9935142005891486
80.0	0.9617049151002924
90.0	0.9823654564057027
100.0	0.9532963605022411
110.0	0.9713418181309539
120.0	0.9897288678967208
130.0	0.9580407771811376
140.0	0.9713012652936457
150.0	0.9896875474130246
160.0	0.953216763259536
170.0	0.9664104809581061
180.0	0.9748940058733949
190.0	0.9933482969043668
200.0	0.91575649823237
210.0	0.9603215935900202
220.0	0.9834109440220101
230.0	0.9543109111799719
240.0	0.9748126052953457
250.0	0.9932653554511575
260.0	0.9590603771293331
270.0	0.9650606815409313
280.0	0.9784183339501281
290.0	0.9870072684424487
300.0	0.9626078399593135
310.0	0.9783774856732651
320.0	0.9894396607354538
330.0	0.9673985985107607
340.0	0.9758907975040342
350.0	0.9968560975818646
360.0	0.9553265415986116
370.0	0.9782957942356276
380.0	0.9993127614449575
390.0	0.9648995292161726
400.0	0.9831646294429918
410.0	0.9992710408404509
420.0	0.9745684443390793
430.0	0.9831235830116317
440.0	0.9992293219777499
450.0	0.9720914374004167
460.0	0.9880164454182385
470.0	0.9563831812320225
480.0	0.981832420078234
490.0	0.9551670147460232
500.0	0.9659667570596737
510.0	0.976888608315782
520.0	0.9879339491578301
530.0	0.9611028335244246
540.0	0.9792960640640475
550.0	0.9928507520537972
560.0	0.961062708157518
570.0	0.9817094527745063
580.0	0.9550473870885825
590.0	0.965845776813781
600.0	0.9816684670957481
610.0	0.952619995745972
620.0	0.968226018478526
630.0	0.9915054177831351
640.0	0.9977087126687553
650.0	0.9561436362868516
660.0	0.9645370348694003
670.0	0.9902032467582238
680.0	0.958499967882646
690.0	0.9208705250048972
700.0	0.9901619064695443
710.0	0.9608621064496718
720.0	0.9766031531871603
730.0	0.987645266487026
740.0	0.9560238863147309
750.0	0.974120974741084
760.0	0.9876040329922225
770.0	0.9607818774910165
780.0	0.9740803058759436
790.0	0.9950067107204462
800.0	0.958339911109989
810.0	0.9691755282628977
820.0	0.9850527672390349
830.0	0.9582999010931484
840.0	0.983800143619166
850.0	0.9999169661445928
860.0	0.9776425082418871
870.0	0.9911743089414028
880.0	0.9570413397761885
890.0	0.9800518219099756
900.0	0.9936169705379053
910.0	0.9666315107766094
920.0	0.9849293966104627
930.0	0.9533949702836214
940.0	0.9738769870172688
950.0	0.9898312462874632
960.0	0.9935340066512934
970.0	0.9641344230906417
980.0	0.9823850402136312
990.0	0.995982483514519

Chosen Epoch# = 154.0 Eta = 0.9749347087113561

Weight	Feature
-12.944120339310413	Bias=1.0
10.51280426865537	Manufacture_Country=UK
-15.731900126951448	Manufacture_Country=Germany
9.98766324438985	Manufacture_Country=France
16.897071215654087	Manufacture_Country=USA
-18.47173616180264	Manufacture_Country=Italy
20.17952054873459	Manufacture_Country=Japan
32.02255997088246	Manufacture_Country=Korea
22.313870157710863	Manufacture_Country=Sweden
-12.944689409107642	Engine_Displacement[0]
3.967853603921369	Engine_Displacement[1]
164.55741637565256	Engine_Displacement[2]
43.696692466881444	Engine_Displacement[3]
-10.406216075409171	Engine_Displacement[4]
-15.118632221753193	Engine_Displacement[5]
7.02323511885752	Engine_Displacement[6]
9.986236703771489	Engine_Displacement[7]
12.951506830559335	Engine_Displacement[8]
12.95075462363511	Engine_Displacement[9]
-12.937013821425197	Num_Cylynders[0]
-143.88133978037968	Num_Cylynders[1]
-62.85144683758723	Num_Cylynders[2]
65.9478032648515	Num_Cylynders[3]
24.098356089758536	Num_Cylynders[4]
-1.807000227104119	Num_Cylynders[5]
9.98026441360668	Num_Cylynders[6]
9.980545339006635	Num_Cylynders[7]
12.952697490145365	Num_Cylynders[8]

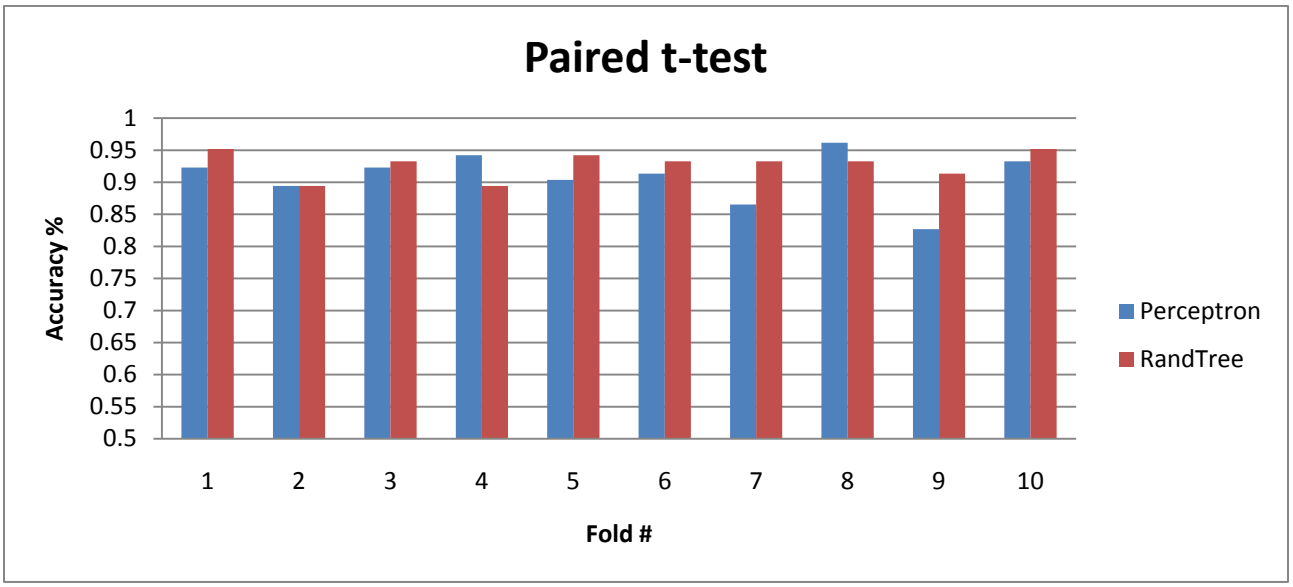
```
12.952598290133128    Num_Cylynders[9]
-12.943887706567342   Num_Gears[0]
155.31647851535757    Num_Gears[1]
155.31789316803622    Num_Gears[2]
155.32405262805707    Num_Gears[3]
4.230499764275257     Num_Gears[4]
-14.019757840583242   Num_Gears[5]
60.51397116646434     Num_Gears[6]
-97.64911100615117    Num_Gears[7]
12.954370092084499    Num_Gears[8]
12.946237721865149    Num_Gears[9]
11.4256192066915      Trans_Type=A
96.31889020853828     Trans_Type=AM
-122.19741311611298   Trans_Type=CVT
55.62265463947813     Trans_Type=M
-20.155058696428856   Trans_Type=OT
30.81302564769511     Trans_Type=SA
5.819331831162502     Drive_Type=4
-7.597069915049708    Drive_Type=A
-30.947153854363357   Drive_Type=F
79.55014075580226     Drive_Type=P
-7.957348979609936    Drive_Type=R
-38.387181004370525   Vehicle_Class=S_Car
-9.073051012184463    Vehicle_Class=M_Car
-16.38822145568477    Vehicle_Class=L_Car
8.301496540205083     Vehicle_Class=S_Truck
15.209967318339991    Vehicle_Class=L_Truck
72.84999874373733     Vehicle_Class=Van
32.2541069727212      Vehicle_Class=SUV
-9.803479255090975    Variable_Valve_Timing=Y
9.808301736602294     Variable_Valve_Timing=N
1.027778999352655     Air_Aspiration_Method=NA
18.87441204192213     Air_Aspiration_Method=TC
-6.9347951352002335   Air_Aspiration_Method=SC

Errors:
 posTestEx16
 negTestEx22
 posTestEx45
 posTestEx47
 posTestEx48
 posTestEx50
 posTestEx57
 posTestEx59
 negTestEx61
 negTestEx63
 posTestEx74
 posTestEx80
 negTestEx87
 posTestEx101
Error Rate : 0.1346153846153846
```



The plot shows the variation of eat against the number of epochs, the eat value is taken every 10 *epochs*. The eta value seem to reach the upper limit of eat quickly and fluctuate below the upper limit. When tested without bounding the Eta value, Eta grew exponentially as the number of Epochs went up.

10-fold Cross Validation

Paired t-test comparison between Perceptron and Random-DecisionTree



Fold	Perceptron	RandTree
1	0.923076923	0.951923077
2	0.894230769	0.894230769
3	0.923076923	0.932692308
4	0.942307692	0.894230769
5	0.903846154	0.942307692
6	0.913461538	0.932692308
7	0.865384615	0.932692308
8	0.961538462	0.932692308
9	0.826923077	0.913461538
10	0.932692308	0.951923077
Ave	0.908653846	0.927884615
St Dev	0.039057877	0.02089488
P value		0.165459123

The comparisons show that the difference between the Perceptron and Random-DecisionTree is not statistically significant. This implies that the Perceptron also performs well on the cars dataset.

Using a Gaussian Kernel

For this implementation a Gaussian kernel is used to create the features for a Perceptron. Then the gradient-descent method developed for the first part, perceptron with weight decay is used for training and classification.

The GaussianKernel class creates a new feature set with similarity to 10% of randomly chosen exemplars and passes the training/ test sets in the new feature format to the PerceptronLearner to tran and categorize the samples.

The Gaussian kernel as the similarity between two examples, A and B, where A_i and B_i are the i^{th} feature of the examples

kernel(A, B) = exp {- [SUM $(A_i - B_i)^2$] / $sigma^2$}

A kernel tuning set (10% of training set) is used to find the best sigma value from

Sigma = {0.03, 0.1, 0.3, 1, 3, 10}

The GaussianKerne uses the following default settings they can be changed by the user

```
Sigma = new double[]{0.03, 0.1, 0.3, 1, 3, 10};
useInfoGain=false;
exemplarFraction =0.1;
kernelTrainFraction =0.1;
epochCount =1000;
```

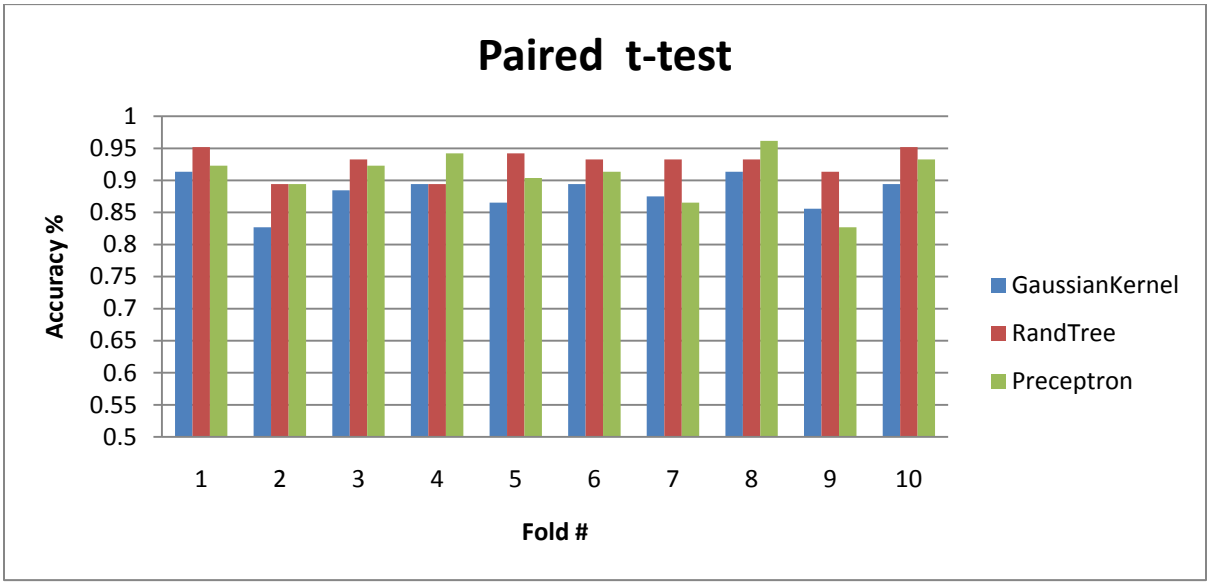
Gaussian Kernel output on cross validation fold#1 with default settings

```
HW3 cars.names train1.data test1.data 1
```

```
Sigma   KerenelTune Error
 0.03   0.4166666666666667
 0.1    0.4166666666666667
 0.3    0.4166666666666667
 1.0    0.40476190476190477
 3.0    0.14285714285714285
 10.0   0.10714285714285714
Chosen Sigma = 10.0

Weight                  Feature
 3.7897595917757627     Bias=1.0
 -27.373525364798077    SimilarityPosToEx0
 14.315033082363417     SimilarityNegToEx1
 33.973478859424084     SimilarityPosToEx2
 10.559827208645817     SimilarityNegToEx3
 35.22359203161715      SimilarityNegToEx4
 -66.87347557362567     SimilarityNegToEx5
 -104.8728818594165     SimilarityNegToEx6
 1.4463203835767517     SimilarityPosToEx7
 -18.020579045222508    SimilarityPosToEx8
 -61.34978448871777     SimilarityNegToEx9
 -20.307475430646445    SimilarityNegToEx10
 0.7503999778635034     SimilarityNegToEx11
 -70.40113252852632     SimilarityNegToEx12
 3.9223307609648828     SimilarityNegToEx13
 -75.48708099838318     SimilarityNegToEx14
 -105.06817708525618    SimilarityNegToEx15
 -77.98293052326181     SimilarityNegToEx16
 56.8988290729307       SimilarityNegToEx17
 33.9795091412245       SimilarityPosToEx18
 10.555167624930327     SimilarityNegToEx19
 -88.32203068303244     SimilarityNegToEx20
 -80.59101092129676     SimilarityNegToEx21
 12.748963704296616     SimilarityPosToEx22
 9.504536088637074      SimilarityNegToEx23
 33.47952062811341      SimilarityPosToEx24
 23.726585421189363     SimilarityNegToEx25
 1.9822550291289394     SimilarityPosToEx26
 8.982781647884753      SimilarityPosToEx27
 2.1537802135852515     SimilarityPosToEx28
 -1.6110299151329601    SimilarityNegToEx29
 -46.699190103717235    SimilarityPosToEx30
 6.531618628295543      SimilarityNegToEx31
 -15.24544107227348     SimilarityPosToEx32
 33.33536974288112      SimilarityNegToEx33
 0.8647727744146552     SimilarityNegToEx34
 35.225425162710266     SimilarityNegToEx35
 -86.55079194500398     SimilarityNegToEx36
 57.72042932069562      SimilarityPosToEx37
 -48.480004535186744    SimilarityNegToEx38
 -29.058638811258145    SimilarityNegToEx39
 -70.38049134101827     SimilarityNegToEx40
 35.22401795828558      SimilarityNegToEx41
 1.1376644110793321     SimilarityNegToEx42
 -0.7070803872104031    SimilarityPosToEx43
 -51.23240952308685     SimilarityNegToEx44
 3.9271687993797393     SimilarityNegToEx45
 -20.62180797923874     SimilarityNegToEx46
 -17.313919890411455    SimilarityNegToEx47
 -32.820719763543956    SimilarityNegToEx48
 -3.2967270960044424    SimilarityPosToEx49
 -38.251578240100535    SimilarityNegToEx50
 -22.582304284671444    SimilarityNegToEx51
 19.930547016852888     SimilarityPosToEx52
 8.37968926095748       SimilarityNegToEx53
 -28.05776602739571     SimilarityPosToEx54
 23.73160010603693      SimilarityNegToEx55
 -95.70065847585661     SimilarityNegToEx56
 9.891970802637925      SimilarityPosToEx57
 -2.706936161754199     SimilarityNegToEx58
 13.57696808974289      SimilarityPosToEx59
 4.392432651841295      SimilarityNegToEx60
 -4.093629721568893     SimilarityPosToEx61
 28.305946951929172     SimilarityPosToEx62
 -24.948696043764457    SimilarityNegToEx63
 14.117828180502306     SimilarityPosToEx64
 18.4091211113541       SimilarityPosToEx65
 20.44317288553112      SimilarityNegToEx66
 29.705961438521353     SimilarityPosToEx67
 10.559422418852012     SimilarityNegToEx68
 -2.900773379868115     SimilarityNegToEx69
 5.227836273905651      SimilarityPosToEx70
 25.597452761423646     SimilarityNegToEx71
 11.71272964878677      SimilarityPosToEx72
 -88.8567942613965      SimilarityNegToEx73
 1.1772204472896428     SimilarityPosToEx74
 16.33283462704023      SimilarityNegToEx75
 -6.437985581359787     SimilarityPosToEx76
 87.2244464181883       SimilarityNegToEx77
 24.22819756719882      SimilarityPosToEx78
 -8.40783635288739      SimilarityPosToEx79
 0.6410691845626564     SimilarityPosToEx80
 11.47709300020274      SimilarityNegToEx81
 49.99441712609208      SimilarityPosToEx82
 -32.27330861564673     SimilarityPosToEx83
 -2.2913745034344597    SimilarityPosToEx84
 56.89243722770676      SimilarityNegToEx85
 -17.28048371826675     SimilarityNegToEx86
 -20.20270266750873     SimilarityPosToEx87
 -9.782807258443889     SimilarityNegToEx88
 -21.391282098797486    SimilarityNegToEx89
 1.4399450430649017     SimilarityPosToEx90
 2.3091769910934765     SimilarityPosToEx91
 7.188088554492789      SimilarityPosToEx92

Errors:
 posTestEx4
 negTestEx5
 posTestEx16
 posTestEx45
 posTestEx47
 posTestEx50
 posTestEx57
 posTestEx59
 posTestEx74
 negTestEx82
 posTestEx94
 posTestEx97
 posTestEx101
Error Rate : 0.125
```

10-fold Cross Validation

Pairted *t*-test comparison between GaussianKernel vs Random-DecisionTree and Gaussian kernel vs Perceptron on the cars data set



Fold#	GaussianKernel	RandTree	Preceptron
1	0.913461538	0.951923077	0.923076923
2	0.826923077	0.894230769	0.894230769
3	0.884615385	0.932692308	0.923076923
4	0.894230769	0.894230769	0.942307692
5	0.865384615	0.942307692	0.903846154
6	0.894230769	0.932692308	0.913461538
7	0.875	0.932692308	0.865384615
8	0.913461538	0.932692308	0.961538462
9	0.855769231	0.913461538	0.826923077
10	0.894230769	0.951923077	0.932692308
Ave	0.881730769	0.927884615	0.908653846
St Dev	0.026835174	0.02089488	0.039057877
p value		0.000134576	0.017447017

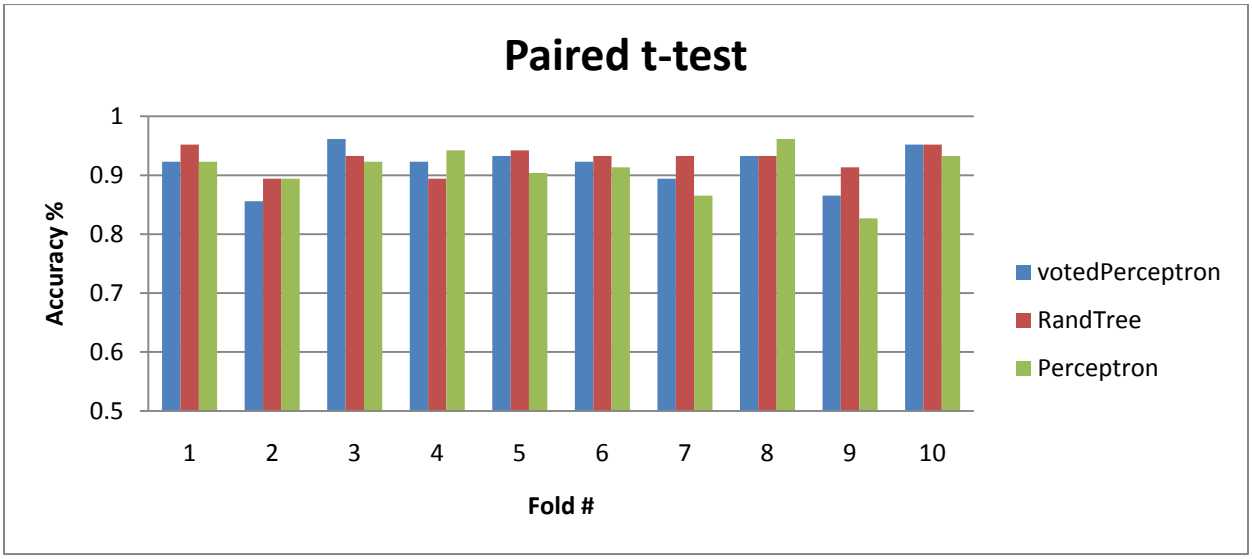
The comparisons show that the difference between the GaussianKernel and Random-DecisionTree is statistically significant. Also the difference between the GaussianKernel and Perceptron is statistically significant. According to the observations the GaussianKernel seem to perform poorly on the cars dataset.

Additional Experiments

I have selected voted perceptrons and information gain scaled Gaussian kernel as the additional experiments. I chose voted perceptrons to see if it had an improvement in accuracy over the single perceptrons, because in the ID3 test the Random forests did not make a significant improvement. I chose the info gain scaled Gaussian kernel experiment because the original Gaussian kernel performed significantly worse, and I'm curious to see if the info gain based scaling would improve the performance.

Voted Perceptron

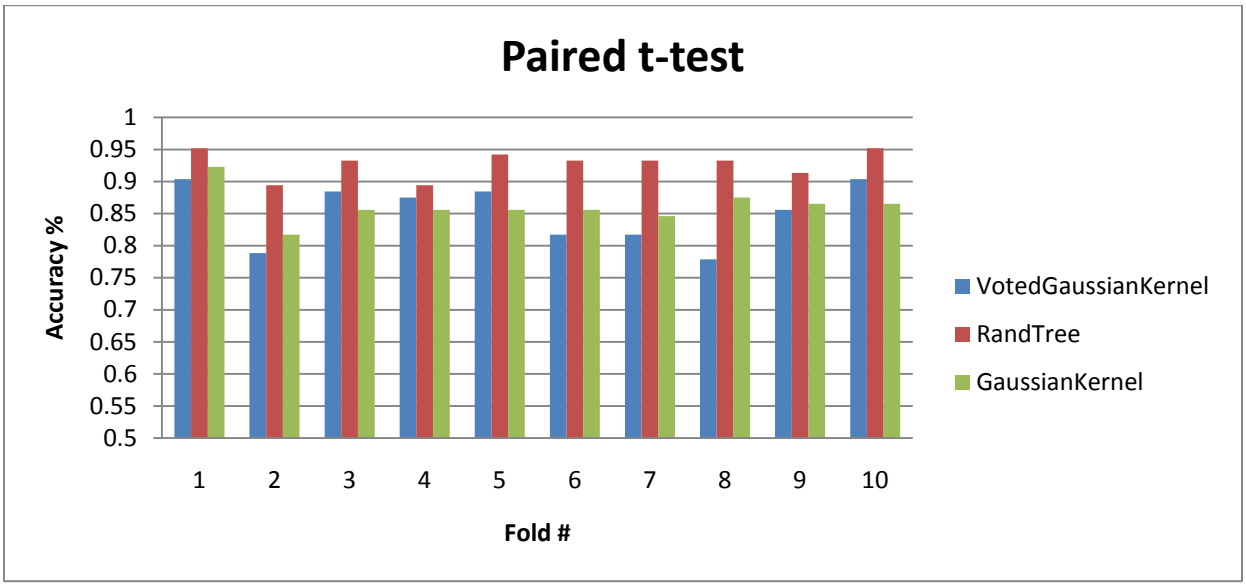
In this t-test I have compared the voted Perceptron to Random-DecisionTree and single Perceptron.



Fold#	votedPerceptron	RandTree	Perceptron
1	0.923076923	0.951923077	0.923076923
2	0.855769231	0.894230769	0.894230769
3	0.961538462	0.932692308	0.923076923
4	0.923076923	0.894230769	0.942307692
5	0.932692308	0.942307692	0.903846154
6	0.923076923	0.932692308	0.913461538
7	0.894230769	0.932692308	0.865384615
8	0.932692308	0.932692308	0.961538462
9	0.865384615	0.913461538	0.826923077
10	0.951923077	0.951923077	0.932692308
Ave	0.916346154	0.927884615	0.908653846
St Dev	0.034535163	0.02089488	0.039057877
P value		0.21141541	0.411301225

There is no statistically significant difference between either of the two comparisons. However the Voted Perceptron seems to be slightly better than the Single Perceptron.

In the next t-test I have compared the Voted Gaussian Kernel to Random-DecisionTree and Gaussian Kernel. I had to reduce size of the exemplars set and the number of training epochs due to run time and memory restrictions.

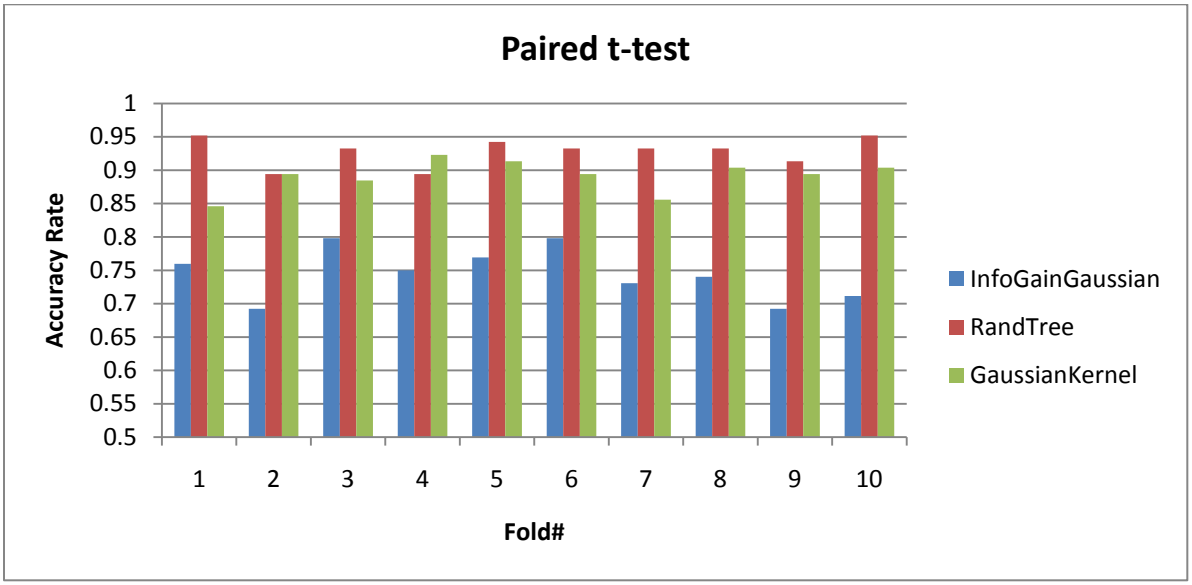


Fold#	VotedGaussianKernel	RandTree	GaussianKernel
1	0.903846154	0.951923077	0.923076923
2	0.788461538	0.894230769	0.817307692
3	0.884615385	0.932692308	0.855769231
4	0.875	0.894230769	0.855769231
5	0.884615385	0.942307692	0.855769231
6	0.817307692	0.932692308	0.855769231
7	0.817307692	0.932692308	0.846153846
8	0.778846154	0.932692308	0.875
9	0.855769231	0.913461538	0.865384615
10	0.903846154	0.951923077	0.865384615
Ave	0.850961538	0.927884615	0.861538462
St Dev	0.046941726	0.02089488	0.026507786
p value		0.000286674	0.437386155

According to the observations the Voted Gaussian Kernel is statistically significantly different from Random-DecisionTree. The difference between Voted Gaussian Kernel and Gaussian Kernel is not statistically significant. It seems like the Voted Gaussian Kernel performs quite poorly, this could be due to the reduced exemplars and epochs.

ID3's information gain to scale the distances in the Gaussian kernel

In this t-test I have compared the Info. Gain scaled Gaussian Kernel to Random-DecisionTree and Gaussian Kernel.



Fold#	InfoGainGaussian	RandTree	GaussianKernel
1	0.759615385	0.951923077	0.846153846
2	0.692307692	0.894230769	0.894230769
3	0.798076923	0.932692308	0.884615385
4	0.75	0.894230769	0.923076923
5	0.769230769	0.942307692	0.913461538
6	0.798076923	0.932692308	0.894230769
7	0.730769231	0.932692308	0.855769231
8	0.740384615	0.932692308	0.903846154
9	0.692307692	0.913461538	0.894230769
10	0.711538462	0.951923077	0.903846154
Ave	0.744230769	0.927884615	0.891346154
St Dev	0.03851492	0.02089488	0.024006389
p value		6.57522E-08	3.53734E-06

The comparisons show that the difference between the InfoGainGaussian and Random-DecisionTree is statistically significant. Also the difference between the InfoGainGaussian and GaussianKernel is statistically significant. According to the observations the InfoGainGaussian seem to perform poorly on the cars dataset.