Project #1:

Arrow Recognizer Report

Contents:

Submitted By:
Mayank Sharma
(mayank.sharma@tamu.edu)

1. Preprocessing

The sketch data retrieved from the server may be cluttered too much or dirty to be processed easily. Thus, pre-processing is done on the data to make it easier to process and apply algorithms in a better way. The data received is first sampled into equidistant points. The interspacing distance is calculated by dividing the bounding box diagonal by an empirical factor of 40. Each Stroke is then subsequently sampled, creating new points which are "resampling distance" apart from the previous one. These points are separately stored as an array, in a new stroke array of sampled points. Several points which might be having the same coordinates in the sketch will get removed after this process. Although points having same timestamp are still present. We can further remove such points but I have not considered it in my implementation as most of the points were getting removed after the process, leaving very few points to run the algorithms.

Some of the strokes in the provided sketch had a missing timestamp valued as -1 and were hampering the sequence by occurring in the middle, thus I considered sorting the strokes prior to resampling based on the timestamp of the first point in the stroke. All empty strokes were removed in the resampled data. Some features and essential variables were also calculated during the sampling process to avoid looping through all the strokes again thereby saving execution time. All the reusable features values were calculated in the preprocessing step so that dependent feature do not suffer in case the original feature is kept out of the subset.

Pseudo Code for Resampling:

```
Input: points = A series of points
Input: S = an interspacing distance
Output: Resampled points
D = 0
q = resampled[0] <- points[0]       // Initialization
for i = 1 to (|points|-1)
   p = points[i]
   D = DISTANCE(q, p)
   if D >= S then
       r.x = q.x + (p.x - q.x)*(S/D)
       r.y = q.y + (p.y - q.y)*(S/D)
       APPEND(resampled, r)
       q = r
return resampled
```

2. Features

1. Cosine of starting angle

   Explanation: This is the first Rubine feature and returns the cosine of starting angle of the sketch. The angle is calculated between first and third resampled point to avoid noise fluctuations. It can be calculated using the formula:

   Equation: (x2-x0)/ Math.sqrt(Math.pow((y2-y0),2) + Math.pow((x2-x0),2)) where (x0,y0) and (x2, y2) are the first and third resampled point respectively.

   Justification: This feature was not included in the subset as arrows can be drawn in different direction and not necessarily have the same cosine of starting angle.

2. Sine of starting angle

   Explanation: This is the second Rubine feature and returns the sine of starting angle of the sketch. The angle is calculated between first and third resampled point to avoid noise fluctuations. It can be calculated using the formula:

   Equation: (y2-y0)/ Math.sqrt(Math.pow((y2-y0),2) + Math.pow((x2-x0),2)) where (x0,y0) and (x2, y2) are the first and third resampled point respectively.

   Justification: This feature was not included in the subset as arrows can be drawn in different direction and not necessarily have the same sine of starting angle.

3. Length of the Bounding Box Diagonal

   Explanation: This is the third Rubine feature and returns the of the bounding box formed by the sketch. We first calculate the maximum and minimum values of x and y coordinates and then calculate the distance between the points (minX, minY) and (maxX, maxY). It can be calculated using the formula:

   Equation:

   Math.sqrt((maxY-minY)*(maxY-minY) + (maxX-minX)*(maxX-minX))

Justification: This feature was included in the subset as arrows for the testing data seems to correlate with this feature very well. In general, diagonal may not be necessarily similar for all the arrows.

4. ## Angle of the Bounding Box Diagonal

Explanation: This is the fourth Rubine feature and returns the angle of the bounding box formed by the sketch. We first calculate the maximum and minimum values of x and y coordinates and then calculate the arctan angle between the points (minX, minY) and (maxX, maxY). It can be calculated using the formula:

Equation:

```
Math.atan2(maxY-minY, maxX-minX)
```

Math.atan2 is a javascript library function and returns the arctan value for the given coordinates.

Justification: This feature was not included in the subset as angle formed by the diagonal may not be necessarily similar for all the arrows.

5. ## Start and Endpoint Distance

Explanation: This is the fifth Rubine feature and returns the distance between the first and the last point of the sketch. We first take the initial and end values of x and y coordinates and then calculate the distance between the points (x0, y0) and (xL, yL). It can be calculated using the formula:

Equation:

```
Math.sqrt((yL-y0)*(yL-y0) + (xL-x0)*(xL-x0))
```

Where (x0, y0) and (xL, yL) are the first and the last sketch points.

Justification: This feature was not included in the subset as distance between the start and end points of the sketch may not be necessarily similar for all the arrows as they can be drawn in different sizes.

6. ## Cosine of angle from Start to Endpoint

Explanation: This is the sixth Rubine feature and returns the cosine of the angle made by the line between the first and the last point of the sketch. We first take the initial and end values of x and y coordinates and then calculate the cosine of angle between the points (x0, y0) and (xL, yL). It can be calculated using the formula:

Equation:

(xL -x0) / Math.sqrt((yL-y0)*(yL-y0) + (xL-x0)*(xL-x0))

Where (x0, y0) and (xL, yL) are the first and the last sketch points.

Justification: This feature was not included in the subset as cosine angle between the start and end points of the sketch may not be necessarily similar for all the arrows.

7. <u>Sine of angle from Start to Endpoint</u>

Explanation: This is the seventh Rubine feature and returns the sine of the angle made by the line between the first and the last point of the sketch. We first take the initial and end values of x and y coordinates and then calculate the sine of angle between the points (x0, y0) and (xL, yL). It can be calculated using the formula:

Equation:

(yL - y0) / Math.sqrt((yL-y0)*(yL-y0) + (xL-x0)*(xL-x0))

Where (x0, y0) and (xL, yL) are the first and the last sketch points.

Justification: This feature was not included in the subset as sine angle between the start and end points of the sketch may not be necessarily similar for all the arrows.

8. <u>Stroke Length</u>

Explanation: This is the eighth Rubine feature and returns the total length of the stroke drawn by the sketch. We need to loop through all the points of each stroke and add up the distance between all consecutive points. It can be calculated using the formula:

Equation:

StrokelengthF8 = StrokelengthF8 + Math.sqrt((val.y-pval.y)*(val.y-pval.y) + (val.x-pval.x)*(val.x-pval.x)) where val and pval are the current and the last point. It is very necessary to reinitialize this value to 0 for every sketch.

Justification: This feature was included in the subset as arrows for the testing data seems to correlate with this feature very well. In general, stroke length may vary a lot for different types of arrows

9. <u>Total Rotation</u>

Explanation: This is the ninth Rubine feature and returns the rotation of the angles at all the points of the sketch. We have to loop through all the points and calculate the turn value at every point. We compute delX and delY values with subsequent point and delXprev and delYprev values with the previous points. "del" represents delta meaning the difference in either X or Y values. It can be calculated using the formula:

Equation:

Rotation = Rotation + Math.atan2(delXprev*delY - delX*delYprev, delX*delXprev + delY*delYprev)

where delX & delY values are the difference in X & Y coordinates of the current point with subsequent point and delXprev & delYprev values are the difference from the previous point.

Justification: This feature was included in the subset as mostly arrows tend to be in the form of straight lines with limited curvature, thus the total rotation should tend to be very less for the sketch.

10. <u>Total Absolute Rotation</u>

Explanation: This is the tenth Rubine feature and returns the absolute value of the rotation of the angles at all the points of the sketch. We have to loop through all the points and calculate the absolute turn value at every point. We compute delX and delY values with subsequent point and delXprev and delYprev values with the previous points. "del" represents delta meaning the difference in either X or Y values. It can be calculated using the formula:

Equation:

AbsRotation = AbsRotation + Math.abs(Math.atan2(delXprev*delY - delX*delYprev, delX*delXprev + delY*delYprev))

where delX & delY values are the difference in X & Y coordinates of the current point with subsequent point and delXprev & delYprev values are the difference from the previous point.

Justification: This feature was included in the subset as mostly arrows tend to be in the form of straight lines with limited curvature, thus the total absolute rotation although more than total rotation but should be still smaller as compared to other sketches.

11. <u>Total Squared Rotation</u>

Explanation: This is the eleventh Rubine feature and returns the squared value of the rotation of the angles at all the points of the sketch. It is a measure of sharpness of the stroke. It can be calculated using the formula:

Equation:

SqRotation = SqRotation + Math.pow(Math.atan2(delXprev*delY - delX*delYprev, delX*delXprev + delY*delYprev), 2)

where delX & delY values are the difference in X & Y coordinates of the current point with subsequent point and delXprev & delYprev values are the difference from the previous point.

Justification: This feature was included in the subset as mostly arrows tend to be in the form of straight lines with limited curvature, thus the total squared rotation will become even smaller for small rotational value providing better intuition on sharp figures like arrows.

12. <u>Maximum Speed</u>

Explanation: This is the twelfth Rubine feature and returns the maximum speed the pen during the sketch. We need to loop through all the sketch points and find the distance over time values for all the consecutive points. It can be calculated using the formula:

Equation:

max ((Math.pow(val.x - pval.x, 2) + Math.pow(val.y - pval.y, 2))/ Math.pow(val.time - pval.time, 2))

where val and pval are the current and the last point.

Justification: This feature was not included in the subset as for sketches maximum speed of pen depends on individual user hence it does not make a good differentiator.

13. <u>Total Time</u>

Explanation: This is the last Rubine feature and returns the total time taken to draw the sketch. We just subtract the timestamp of first point from the last sketch point. It can be calculated using the formula:

Equation:

Total Time = (tL-t0)

where tL and t0 are the timestamp values as measured from epoch for the first anf the last point of the overall sketch.

Justification: This feature was not included in the subset as same sketches can be drawn in different time depending on user's pen speed.

14. <u>Aspect</u>

Explanation: This is the twelfth Long feature and returns the aspect i.e. the similarity along the y=x line. It measures if the sketch is long and thin or not. We subtract the angle of Bounding Box Diagonal from Pi/4 to obtain this value. It can be calculated using the formula:

Equation:

Math.abs (Math.PI/4 - Feature4)

Justification: This feature was not included in the subset as arrows can be drawn in a variety of ways and doesn't necessarily need to have an orientation symmetrical to y=x line.

15. <u>Curviness</u>

Explanation: This is the thirteenth Long feature and returns the summation of only those angular rotations which are less than 19 degrees. It can be calculated using the formula:

Equation:

```
val = Math.abs (Math.atan2(delXprev*delY - delX*delYprev,
delX*delXprev + delY*delYprev))

if (val < (19*Math.PI)/180) {curvinessF15 += val;}
```

where delX & delY values are the difference in X & Y coordinates of the current point with subsequent point and delXprev & delYprev values are the difference from the previous point.

Justification: This feature was included in the subset as arrows are mostly drawn in a straight line fashion with very less curviness, thus it is a good metric for classification of arrows.

16. <u>Total Angle traversed by Total Length</u>

Explanation: This is the fourteenth Long feature and returns the division of Total Angle by Total Stroke Length. It can be calculated using the formula:

Equation:

Feature9 / Feature8

Justification: This feature was included in the subset as arrows are mostly drawn in a straight line fashion hence total rotation with respect to the length of stroke will better classify similar arrows with enlarged size. It only takes into consideration the amount of rotation per unit length of stroke.

17. <u>Density Metric #1</u>

Explanation: This is the fifteenth Long feature and returns the division of Total Stroke Length by Start to Endpoint Distance. It can be calculated using the formula:

Equation:

Feature8 / Feature5

Justification: This feature was included in the subset as arrows are mostly drawn composed of straight lines and this feature is a great line classifier as for lines Stroke Length and distance between start to endpoints are nearly the same.

18. <u>Density Metric #2</u>

Explanation: This is the sixteenth Long feature and returns the division of Total Stroke Length by Length of the Bounding Box Diagonal. It can be calculated using the formula:

Equation:

Feature8 / Feature3

Justification: This feature was included in the subset as most arrows have shaft as a major part of the stroke which is almost comparable to the diagonal length and its percentage in total stroke should be similar for all the arrows.

19. <u>Non-Subjective Openness</u>

Explanation: This is the seventeenth Long feature and returns the division of Start to Endpoint Distance by Length of the Bounding Box Diagonal. It can be calculated using the formula:

Equation:

Feature5 / Feature3

Justification: This feature was included in the subset as it reflects the openness of the sketch, but dividing by Diagonal length it becomes irrelative of the stroke size, making a good measure to classify open strokes such as arrows.

20. <u>Area</u>

Explanation: This is the eighteenth Long feature and returns the total area under the bounding box. We first calculate the maximum and minimum values of x and y coordinates and then use the formula:

Equation:

areaF20 = (maxX-minX)*(maxY-minY)

Justification: This feature was not included in the subset as arrows can be drawn in a variety of sizes having different areas.

21. Log of Area

Explanation: This is the nineteenth Long feature and returns the log of the total area under the bounding box. It can be calculated using the formula:

Equation:

Log (Feature20)

Justification: This feature was not included in the subset as arrows can be drawn in a variety of sizes having different areas and taking its log will also not be a good classifier for arrows.

22. Total Angle by Absolute Angle

Explanation: This is the twentieth Long feature and returns the division of total angular rotation by Total absolute rotation. It can be calculated using the formula:

Equation:

Feature9 / Feature10

Justification: This feature was included in the subset as it compares two major angular parameters which are very helpful to classify the straight line fashioned arrows.

23. Log of Total Length

Explanation: This is the twenty first Long feature and returns the log of the total Stroke Length. It can be calculated using the formula:

Equation:

Log (Feature8)

Justification: This feature was included in the subset as similar to stroke length this feature was also correlating very well with the training data.

24. Log of Aspect

Explanation: This is the twenty second Long feature and returns the log of the Aspect. It can be calculated using the formula:

Equation:

Log (Feature14)

Justification: This feature was not included in the subset as arrows can be drawn in diverse ways – long and thin or small and thick. Likewise Aspect, this feature will also not be a good classifier.

25. Normalized Distance Between Direction Extremes

Explanation: This feature is taken from Paulson PaleoSketch paper. To calculate this feature, we first take the point with the highest direction value (change of y over change of x) and the point with the lowest direction value and compute the stroke length between these two points. This length is then divided by the length of the entire stroke, essentially giving us the percentage of the stroke that occurs between the two direction extremes. For curved shapes, such as arcs, the highest and lowest directional values will typically be near the endpoints of the stroke, thus yielding very high NDDE values while this value is low for polylines.

Pseudo Code:

- First calculate the points with minimum and maximum Direction values by comparing

  Math.atan2 (delY, delX)

  where delY and delX are difference in the coordinates of current point and previous point.

- Calculate the stroke length only between those two points by the formula

SL = SL + Math.sqrt((val.y-pval.y)*(val.y-pval.y) + (val.x-pval.x)*(val.x-pval.x))

where val and pval are the current and the last point. It is very necessary to reinitialize this value to 0 for every stroke.

- Divide this value by the total Stroke Length to get NDDE

  NDDE = SL / Feature8

Justification: This feature was included in the subset as arrows contain polyline in the head part thus can be classified in a better way having very limited amount of stroke length in the head part between Direction Extremes.

26. Direction Change Ratio

Explanation: This feature is also taken from Paulson PaleoSketch paper and also measures the presence of spikes in Direction Graph. This value is computed as the maximum change in direction divided by the average change in direction. Because there tends to be a great deal of noise at the beginning and ending of a stroke we ignore the first and last 5% of the stroke we calculating this feature. Polylines have higher DCR values than curved strokes.

Pseudo Code:

- Remove the tails by cutting off first and last 5% of the stroke.

  skip = Math.floor(Number of Points / 20)
- Then calculate the change in direction values for all the points which would be similar to computing absolute rotational value at each point and find the maximum value among all the points.

```
for i in Points:
        if (i >= skip and i < Points.length - skip) then
                if (maxDirectionChange < absval)  then
                        maxDirectionChange = absval;
                        avgChangeSum += absval;
                end if
        end if
End for
avgDirectionChange = avgChangeSum / (Points.length – 2*skip)
```

where absval is calculated similar to Feature10

- Take Average of all direction change values
- Divide max Direction Change value by its average value to get DCR.

DCR = (maxDirectionChange)/ avgDirectionChange

Justification: This feature was included in the subset as arrows contain polyline in a limited amount among straight lines and some curvy tails. Thus Direction change ratio seems to be a good metric to classify this mixed behavior.

27. Polyline Arrows

Explanation: Arrows having polylines in them can be better classified when we divide DCR value by Density Metric #1 which calculates the amount of straight line present in the sketch. It can be calculated using the formula:

Equation:

(Feature26 / Feature17)

Justification: This feature was included in the subset as most arrows contains mainly straight lines or polylines. Measuring the portion with straight line when compared to polyline seems to classifying arrows even better than the individual metrics.

28. Portion of Maximum Bounding Box by individual Stroke

Explanation: For multi-stroke sketches, we can calculate the ratio of Maximum Bounding Box of formed by any stroke to the overall Bounding Box Diagonal Value. For single stroke sketch this would essentially be 1. It can be calculated as:

Pseudo Code:

- For each stroke calculate the Bounding Box Diagonal Length similar to the process described in Feature 3.

  bblen = Math.sqrt(Math.pow((ymax-ymin),2) + Math.pow((xmax-xmin),2));

where xmax, ymax, xmin, ymin are minimum and maximum value of x and y coordinates in a particular stroke.

- Find the stroke with maximum Diagonal length

```
if (maxStrokeBBlen < bblen)  then
        maxStrokeBBlen = bblen;

endif
```

- Divide this value to overall Bounding Box Diagonal Length value.

```
maxStrokeBBlen / Feature3
```

Justification: This feature was included in the subset as for arrows drawn with multiple strokes, most of the part of overall diagonal is covered by shaft. This ratio would be in certain threshold values for multi-stroke arrows. As the training data involved single stroke arrows mostly, hence it did not confirm very well to the classification.

29. Total Strokes and SubStrokes

Explanation: Getting Inspired from the number of Sub-stroke feature in Paulson PaleoSketch paper, I included multiplication of Stroke count and Subs-troke Count as a feature. It can be calculated using the formula:

Equation:

$$\text{Number of Strokes} * \text{Number of Sub-strokes}$$

Justification: This feature was included in the subset as most arrows are drawn using smaller number of strokes and sub-strokes as compared to other sketches. It confirmed very well with the classifier results mainly because of the inclusion of sub-stroke counts.

30. Similarity along the axis

Explanation: Arrows having are more or less similar along the shaft. Thus we measure this metric by comparing similarity of corners points with respect to the line made by the furthest corners. It can be calculated as:

Pseudo Code:

- Find all the corners. Corners include start and end points of a stroke and any non-curvy point i.e. the point with direction change more than 20 degrees.
- Compute the maximum distance between any two corners forming the shaft.

```
dis = Math.sqrt(Math.pow((Corners[i].y - Corners[0].y),2) + Math.pow(
(Corners[i].x - Corners[0].x), 2) );

if (maxLen < dis) then
        maxLen = dis;
        end = i;
end if
```

- Compute two maximum the perpendicular distance (arrow heads) of all remaining corners with the line made by two farthest points (shaft).

```
disPerp = ( (Corners[end].y - Corners[0].y)*Corners[i].x - (Corners[end].x -
Corners[0].x)*Corners[i].y  +  Corners[end].x * Corners[0].y - Corners[end].y *
Corners[0].x ) / maxLen;

if ( (Math.abs(disPerp) > Math.abs(perpMax1)) and
         (Math.abs(disPerp) > Math.abs(perpMax2)) )   then
        perpMax2 = perpMax1;
        ah2 = ah1;
        perpMax1 = disPerp;
        ah1 = i;
 else if (Math.abs(disPerp) > Math.abs(perpMax2))       then
        perpMax2 = disPerp;
        ah2 = i;
end if
```

- For both arrow heads divide the perpendicular distance to its distance from shaft end and Subtract the two values to obtain similarity metric.

```
Similarity  =   Math.abs (

(perpMax1 / Math.sqrt(Math.pow((Corners[ah1].y - Corners[end].y),2) +
Math.pow((Corners[ah1].x - Corners[end].x),2)))      -
```

(perpMax2 / Math.sqrt(Math.pow((Corners[ah2].y - Corners[end].y),2) + Math.pow((Corners[ah2].x - Corners[end].x),2)))   )

Where ah1 and ah2 represents the index of arrow heads, and 'end' represents the index of the shaft end.

Justification: This feature was not included in the subset as the results from classifier were not encouraging enough. The similarity coefficient for arrows was not much different as compared for other sketches.

31. <u>Similarity coefficient multiplied by stroke counts</u>

Explanation: As an additional feature I tried to see the effect when we multiply the similarity coefficient calculated above with the total count of Strokes and sub-strokes.
The inspiration behind this feature came from the fact that if a small sketch exhibit higher similarity along the shaft, it is more likely to be an arrow like structure. It can be calculated using the formula:

Equation:

Featuer29 * Feature30

Justification: This feature was not included in the subset as similar to similarity metric, this feature also did not confirm well with the interpretation generated by the classifier.

3. Feature Summary Table

* measured for Random Forest Classifier (in %).

Feature Index	Feature Name	Accuracy*	Included in Subset
Feature1	Cosine of starting angle	79.135	No
Feature2	Sine of starting angle	79.335	No
Feature3	Length of Bounding Box Diagonal	95.755	Yes
Feature4	Angle of Bounding Box Diagonal	92.02	No
Feature5	Start and Endpoint Distance	91.17	No
Feature6	Cosine of Angle from Start to Endpoint	88.99	No
Feature7	Sine of Angle from Start to Endpoint	89.29	No
Feature8	Stroke Length	98.055	Yes
Feature9	Total rotation	96.46	Yes
Feature10	Absolute rotation	95.86	Yes
Feature11	Squared rotation	95.49	Yes
Feature12	Maximum speed	85.26	No
Feature13	Total Time	91.675	No
Feature14	Aspect	89.885	No
Feature15	Curviness	96.05	Yes
Feature16	Total angle traversed / Total length	95.61	Yes
Feature17	Density metric 1 [F8 / F5]	97.365	Yes
Feature18	Density metric 2 [F8 / F3]	97.59	Yes
Feature19	Non-subjective openness [F5 / F3]	92.48	No
Feature20	Area of bounding box	92.545	No
Feature21	Log of area	92.55	No
Feature22	Total angle / Absolute angle [F9 / F10]	94.425	Yes
Feature23	Log of Total length	98.055	Yes
Feature24	Log of Aspect	89.885	No
Feature25	Normalized Distance between Direction Extremes	95.23	Yes
Feature26	Direction Change Ratio	95.08	Yes
Feature27	Arrow with Polyline [F26/F17]	94.91	Yes
Feature28	Ratio of maximum Bounding Box Length of a single stroke to Total Bounding Box Length	91.585	Yes
Feature29	Multiplication of number of Strokes and Sub-strokes	87.075	Yes
Feature30	Similarity along the axis	90.295	No
Feature31	Multiplication of F29 and F30	90.26	No

4. Feature Accuracy Table

File Name (.csv)	ZeroR (in%)	J48 (in%)	Random Forest (in%)	Random Committee (in%)
Feature1	50	77.45	79.135	79.08
Feature2	50	77.845	79.335	79.295
Feature3	50	93.03	95.755	95.68
Feature4	50	88.26	92.02	92.01
Feature5	50	88.715	91.17	91.24
Feature6	50	84.92	88.99	89.06
Feature7	50	86.01	89.29	89.315
Feature8	50	95.11	98.055	98.055
Feature9	50	90.4	96.46	96.46
Feature10	50	84.88	95.86	95.87
Feature11	50	86.23	95.49	95.505
Feature12	50	76.84	85.26	85.27
Feature13	50	87.13	91.675	91.715
Feature14	50	85.32	89.885	89.895
Feature15	50	87.09	96.05	96.055
Feature16	50	79.77	95.61	95.64
Feature17	50	91.87	97.365	97.36
Feature18	50	92.575	97.59	97.585
Feature19	50	87.53	92.48	92.485
Feature20	50	89.525	92.545	92.53
Feature21	50	89.525	92.55	92.535
Feature22	50	83.99	94.425	94.45
Feature23	50	95.11	98.055	98.055
Feature24	50	85.385	89.885	89.895
Feature25	50	85.095	95.23	95.24
Feature26	50	86.68	95.08	95.115
Feature27	50	83.44	94.91	94.935
Feature28	50	83.605	91.585	91.605
Feature29	50	87.07	87.075	87.075
Feature30	50	85.99	90.295	90.295
Feature31	50	86.395	90.26	90.26
Features_ALL	50	99.525	99.805	99.77
Features_Subset	50	99.545	99.81	99.81

5. Recognition Results

The features were trained using 10 folds cross validation on four classifiers using a training data set of 10,000 arrow sketches and 10,000 other sketches. The classifier used were **ZeroR**, **J48**, **Random Forest** and **Random Committee**. The recognition results were obtained for Full set of features, selected feature subset and on each individual feature. Random Forest is considered as the source of truth for comparisons. The training accuracy for the whole feature set came out to be 99.805 % while for the selected subset the accuracy was 99.81 %.

Complete Feature List Result Summary:

Correctly Classified Instances	19961	99.805 %
Incorrectly Classified Instances	39	0.195 %
Kappa statistic	0.9961	
Mean absolute error	0.0064	
Root mean squared error	0.0462	
Relative absolute error		1.2816 %
Root relative squared error		9.2411 %
Total Number of Instances	20000	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.998	0.001	0.999	0.998	0.998	0.996	1.000	1.000	other
0.999	0.003	0.998	0.999	0.998	0.996	1.000	1.000	arrow
0.998	0.002	0.998	0.998	0.998	0.996	1.000	1.000 ←	(Average)

=== Confusion Matrix ===

a	b	← classified as
9975	25	| a = other
14	9986	| b = arrow

Subset Feature List Result Summary:

Correctly Classified Instances	19962	99.81 %
Incorrectly Classified Instances	38	0.19 %
Kappa statistic	0.9962	
Mean absolute error	0.0079	
Root mean squared error	0.0463	
Relative absolute error		1.5769 %
Root relative squared error		9.2643 %
Total Number of Instances	20000	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.998	0.002	0.998	0.998	0.998	0.996	1.000	1.000	other
0.998	0.002	0.998	0.998	0.998	0.996	1.000	1.000	arrow
0.998	0.002	0.998	0.998	0.998	0.996	1.000	1.000	← (Average)

=== Confusion Matrix ===

a	b	← classified as
9978	22	| a = other
16	9984	| b = arrow

We observe a small increase in accuracy for subset features which reflects that we removed some features which were having negative effect on classification.