**Project - 1 (Arrow Recognizer)**

**Introduction:**

This is the final project report for the arrow recognizer project. The problem statement of the project was that , given a data set of 20,000 sketch figures, design a program which recognizes arrows in this data set. The recognition was accomplished by coding 30 features (includes Rubine, Long features, NDDE, DCR, etc. as well as few self thought features) , the results were tested in Weka , after looking at the accuracy values provided by Weka an optimal subset of features was created, the optimal subset contains 10 features. The code was written in Javascipt. The following report describes the pre-processing steps, feature explanation and justification, summary and explanation of results.

**Pre-processing Steps:**

A number of pre-processing steps had to be conducted on the data, since it was real world data. The various pre-processing steps taken are explained below:

1. First, empty strokes were removed from the data, since this is false data, if there are no points then the stroke means nothing. Strokes with one point were also removed, this was done because again a stroke with a single point is equivalent to a dot on the paper, this is of no value and is possibly false data.
2. Next the data was resampled. The resampling was done according to the algorithm provided in by Dr. Hammond. This resampling basically spaces out all the points by a set distance. This resampling is important, since real world drawing is affected drastically by the sampling rate of the device receiving the image. If received points are very close together or duplicates of each other, then feature calculation on such points will produce erroneous results. Therefore, cleaning up the duplicates and making all points equidistant by a well though factor is necessary.
3. Finally, the resampled data was also checked for empty strokes and cleaned of the same, it was checked again, since resampling would remove duplicates and this might result in empty strokes. Strokes with less than 3 points were also removed, this was important since we can't calculate a lot of features for strokes with only 2 points. Also an image with only two points would not be any recognizable figure (since resampling is done, and we know the distance between two points can't be greater than a set threshold), it is simply a squiggle and must be removed.

**Feature Explanation and Justification:**

1. Cos of the start angle --

* Psuedocode:

x\_2 = newstrokes[0][2].x; x\_0 = newstrokes[0][0].x; y\_2 = newstrokes[0][2].y; y\_0 = newstrokes[0][0].y;

return ((x\_2-x\_0) / (Math.sqrt(Math.pow((x\_2-x\_0),2) + Math.pow((y\_2-y\_0),2))));

* Details: This feature takes the cos of the starting angle of the gesture. This is a Rubine feature. All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: The starting angle of a gesture gives us a lot of information. Therefore it is a relevant feature, the reason why we take the cos of the starting angle is to avoid discontinuity when the angle will wrap from 2pi to 0. This is a good feature since people generally draw an arrow starting horizontally or vertically, so the starting angle is very distinguishable as compared to the starting angle for other shapes. So even though this feature was good intuitively, and had decent accuracy values when tested in Weka, there were features with better accuracy and therefore this feature was not part of the optimal subset.

1. Sin of start angle ---

* Psuedocode:

x\_2 = newstrokes[0][2].x; x\_0 = newstrokes[0][0].x; y\_2 = newstrokes[0][2].y; y\_0 = newstrokes[0][0].

return ((y\_2-y\_0) / (Math.sqrt(Math.pow((x\_2-x\_0),2) + Math.pow((y\_2-y\_0),2))));

* Details: This feature takes the sin of the starting angle of the gesture. This is a Rubine feature. All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: The starting angle of a gesture gives us a lot of information. Therefore it is a relevant feature, the reason why we take the sin of the starting angle is to avoid discontinuity when the angle will wrap from 2pi to 0. This is a good feature since people generally draw an arrow starting horizontally or vertically, so the starting angle is very distinguishable as compared to the starting angle for other shapes. So even though this feature was good intuitively, and had decent accuracy values when tested in Weka, there were features with better accuracy and therefore this feature was not part of the optimal subset.

1. Bounding box diagonal length -- square root of ((xmax - xmin)^2 +(ymax - ymin)^2)

* Psuedocode:

x\_val = max\_x - min\_x;

y\_val = max\_y - min\_y;

diagonal = Math.sqrt(Math.pow(x\_val,2) + Math.pow(y\_val,2));

return diagonal;

* Details: This feature takes the diagonal length of the bounding box formed around the entire sketch. This is a Rubine feature . All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: The diagonal of BB gives us the general size of the figure/sketch. So this works for us in case of detecting arrows, because arrows comparatively have a smaller size compared to other complex shapes, consequently the length of the diagonal would be smaller too. This feature provided high accuracy during detection and therefore is in the final optimal subset. Of course there are outliers to this feature if a person draws a huge arrow.

1. Bounding box diagonal angle -- arctan ((ymax-ymin)/(xmax-xmin))

* Psuedocode:

return (Math.atan((max\_y-min\_y)/((max\_x-min\_x)+0.0001)));

* Details: This feature takes the angle of the diagonal of the bounding box formed around the entire sketch. This is a Rubine feature . All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: The angle of the diagonal of the BB basically tells us the fatness or slimness of a gesture. This is helpful in arrow detection again, because generally an arrow will be slim, both horizontally and vertically. This feature is a good feature, but there were features with better accuracy compared to this. Therefore it is not included in the optimal subset.

1. Cos of start to end angle -- (x\_p-1 - x\_0)/ ((x\_p-1 - x0)^2 +(y\_p-1 - y0)^2)

* Psuedocode:

if(f5){

x\_0 = newstrokes[0][0].x;

x\_p = newstrokes[(newstrokes.length) - 1][newstrokes[newstrokes.length-1].length -1].x;

else{

feature\_dist\_start\_end(sketch);

x\_0 = newstrokes[0][0].x;

x\_p = newstrokes[(newstrokes.length) - 1][newstrokes[newstrokes.length-1].length -1].x;

return ((x\_p-x\_0)/f5);

* Details: This feature takes the cos of the angle between the starting and the ending angle of the gesture. This is a Rubine feature. All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: Just as the starting angle, the angle between the starting and end of a gesture gives us a lot of information. Therefore it is a relevant feature, the reason why we take the cos of the starting angle is to avoid discontinuity when the angle will wrap from 2pi to 0. This is a good feature since people generally draw an arrow starting horizontally or vertically, so the starting to ending angle is very distinguishable as compared to the starting to ending angle for other shapes. So even though this feature was good intuitively, and had decent accuracy values when tested in Weka, there were features with better accuracy and therefore this feature was not part of the optimal subset.

1. Sin of start to end angle --- (y\_p-1 - y\_0)/ ((x\_p-1 - x0)^2 +(y\_p-1 - y0)^2)

* Psuedocode:

if(f5){

y\_0 = newstrokes[0][0].y;

y\_p = newstrokes[(newstrokes.length) - 1][newstrokes[newstrokes.length-1].length -1].y;

else{

feature\_dist\_start\_end(sketch);

y\_0 = newstrokes[0][0].y;

y\_p = newstrokes[(newstrokes.length) - 1][newstrokes[newstrokes.length-1].length -1].y;

return ((y\_p-y\_0)/f5);

* Details: This feature takes the sin of the angle between the starting and the ending angle of the gesture. All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: Just as the starting angle, the angle between the starting and end of a gesture gives us a lot of information. Therefore it is a relevant feature, the reason why we take the sin of the starting angle is to avoid discontinuity when the angle will wrap from 2pi to 0. This is a good feature since people generally draw an arrow starting horizontally or vertically, so the starting to ending angle is very distinguishable as compared to the starting to ending angle for other shapes. So even though this feature was good intuitively, and had decent accuracy values when tested in Weka, there were features with better accuracy and therefore this feature was not part of the optimal subset.

1. Distance from start point to end point --- square root of ((x\_p-1 - x0)^2 +(y\_p-1 - y0)^2)

* Psuedocode:

if(f5){}

else{var x\_0; var y\_0; var y\_p; var x\_p;

x\_0 = newstrokes[0][0].x;

y\_0 = newstrokes[0][0].y;

x\_p = newstrokes[(newstrokes.length) - 1][newstrokes[newstrokes.length-1].length -1].x;

y\_p = newstrokes[(newstrokes.length) - 1][newstrokes[newstrokes.length-1].length -1].y;

f5 = Math.sqrt(Math.pow((x\_p-x\_0),2) + Math.pow((y\_p-y\_0),2) );}

return f5;

* Details: This feature takes the total distance between the starting point of a gesture to the ending point of a gesture. This is a Rubine feature . All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: This feature basically helps us tell if the shape was closed or open. This good for arrows, since they are relatively open shapes, they don't close in on themselves, so this feature would basically help us differentiate between a circle and an arrow say. Therefore, this feature was taken to be a part of the optimal subset.

1. Total Stroke length

* Psuedocode:

if(total\_stroke\_length){return total\_stroke\_length;}

else{

for(i=0;i<newstrokes.length;i++){

for (j=0;j<(newstrokes[i].length)-1;j++){

x\_temp = newstrokes[i][j+1].x - newstrokes[i][j].x;

y\_temp = newstrokes[i][j+1].y - newstrokes[i][j].y;

final\_add = Math.sqrt(Math.pow(x\_temp,2) + Math.pow(y\_temp,2));

result = result + final\_add;

total\_stroke\_length = result;

return result;

* Details: This feature takes the total stroke length of a gesture. This is a Rubine feature . All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: This feature basically helps us identify if the figure has a lot of ink in it, as in this feature would help us distinguish between a normal circle and a spiral. See here the bounding box diagonal length could have confused us, if both circle and spiral were of same radius, but this feature saves us from that confusion. In terms of arrows this feature is good because the amount of ink used in drawing an arrow , a relatively simple shape, would be less compared to say a square. Therefore this feature was used in our optimal subset. This feature was also needed because it used for the line recognizing test in another feature.

1. Total angle

* Psuedocode:

if(total\_angle){return total\_angle;}

else{

result = 0;

if(flag\_for\_theta\_p){

for(i=0;i<theta\_p\_arr.length;i++){

result = result + theta\_p\_arr[i];

else {

calculate\_theta\_p();

for(i=0;i<theta\_p\_arr.length;i++){

result = result + theta\_p\_arr[i];

if(!isFinite(result))

{ for(i=0;i<theta\_p\_arr.length;i++){

console.log(theta\_p\_arr[i] );

total\_angle = result;

return result;

* Detail : This feature takes the total angle of a gesture. This is a Rubine feature . All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: This feature helps in distinguishing between figures with very little total rotational change and figures with a large rotational change. So, in our case, an arrow will have less rotational change compared to, say a circle or a spiral. This feature was good intuitively, and had decent accuracy values when tested in Weka, but there were features with better accuracy and therefore this feature was not part of the optimal subset.

1. Total absolute angle

* Psuedocode:

if(total\_abs\_angle){return total\_abs\_angle;}

else{

result = 0;

if(flag\_for\_theta\_p){

for(i=0;i<theta\_p\_arr.length;i++){

result = result + Math.abs(theta\_p\_arr[i]);

else {

calculate\_theta\_p();

for(i=0;i<theta\_p\_arr.length;i++){

result = result + Math.abs(theta\_p\_arr[i]);

total\_abs\_angle = result;

return result;}

* Details: This feature takes the total absolute angle of a gesture. This is a Rubine feature . All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: This feature basically tells how much a stroke moves around. So this feature really helps in distinguishing between a curvy sin like wave and a straight line. Since, arrows generally have a straight body, this was a good feature to have in the main set, it had good accuracy values too, but there were other features with better values and therefore this feature was not included in the final subset.

1. Total squared angle

* Psuedocode:

if(flag\_for\_theta\_p){

for(i=0;i<theta\_p\_arr.length;i++){

result = result + Math.pow(theta\_p\_arr[i],2);

else {

calculate\_theta\_p();

for(i=0;i<theta\_p\_arr.length;i++){

result = result + Math.pow(theta\_p\_arr[i],2);

* Details: This feature takes the total squared angle of a gesture. This is a Rubine feature . All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: This feature helps in distinguishing between images with sharp corners and images with soft features. Since an arrow is a sharp shape, this was a good feature to have in the original set, it had good accuracy values too, but there were other features with better values and therefore this feature was not included in the final subset.

1. Total time

* Psuedocode:

return (newstrokes[newstrokes.length - 1][newstrokes[newstrokes.length-1].length -1].time - newstrokes[0][0].time);

* Details: This feature calculates the total time of a gesture. This is a Rubine feature . All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: Total time tells us about the size/complexity of an image. A complex image will take more time to be drawn compared to a simpler image. Since arrow is a comparatively simpler figure, therefore this feature would be quite distinguishable. Still this feature can be pretty deceptive, since a person might just draw a simple figure slowly. Therefore, this feature wasn't picked in the optimal subset.

1. Maximum speed of the gesture

* Psuedocode:

for(i=0;i<newstrokes.length;i++){

for (j=0;j<(newstrokes[i].length)-1;j++){

x\_temp = newstrokes[i][j+1].x - newstrokes[i][j].x;

y\_temp = newstrokes[i][j+1].y - newstrokes[i][j].y;

t\_temp = newstrokes[i][j+1].time - newstrokes[i][j].time;

if(t\_temp!=0)

{final\_add = (Math.pow(x\_temp,2) + Math.pow(y\_temp,2))/(Math.pow(t\_temp,2));

if(final\_add > result) {result = final\_add;}}

* Details: This feature calculates the maximum speed at which a gesture was drawn. This is a Rubine feature . All details (including the calculation equation)of the feature was taken from **[1].**
* Justification: This feature performed surprisingly well. It must be because arrow is a very intuitive symbol to many people, they are able to draw it quickly. Following this thought process this feature was chosen in the final optimal subset.

1. Aspect Ratio

* Psuedocode:

if (aspect\_ratio){return aspect\_ratio;}

else{

aspect\_ratio = Math.abs (0.785398 -(Math.atan((max\_y-min\_y)/(max\_x-min\_x))))

return aspect\_ratio;

* Details: This feature calculates the aspect ratio of a gesture. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: This feature makes all tall skinny shapes similar to flat long shapes. This is important since similar shapes should have similar values. However, this doesn't seem very intuitive, since it's not solely specific to any geometrical property of an arrow. Therefore this feature was not chosen in the final subset.

1. Area of the bounding box

* Psuedocode:

if((max\_x == min\_x)||(max\_y == min\_y))

{ if(total\_stroke\_length)

{return total\_stroke\_length;}

else

{feature\_total\_stroke\_len();

return total\_stroke\_length;}

}

else{x\_val = max\_x - min\_x; y\_val = max\_y - min\_y;

return(Math.abs(x\_val) \* Math.abs(y\_val));}

* Details: This feature calculates the area of the bounding box of the gesture. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: This measure tells us a lot about the size of a shape. Still it's not very intuitive, since area of bounding box of a 45 degree line and circle can be same, but they are not similar shapes. Still, surprisingly this feature provided high accuracy, possibly because our input data had a mix of very varied shapes (in terms of bounding box ). Therefore this feature was selected in the final subset.

1. Log of area of the bounding box

* Psuedocode:

if((max\_x == min\_x)||(max\_y == min\_y))

{ if(total\_stroke\_length)

{return total\_stroke\_length;}

else

{feature\_total\_stroke\_len();

return total\_stroke\_length;}

}

else{x\_val = max\_x - min\_x; y\_val = max\_y - min\_y;

return(Math.log(Math.abs(x\_val) \* Math.abs(y\_val)));}

* Details: This feature calculates the log of the area of the bounding box of the gesture. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: This is a measure which brings all small and big shapes under one range and in a small area, differences can be detected easily, therefore just as it's parent measure, it had very high accuracy. Therefore this was chosen in the optimal subset.

1. Curviness

* Psuedocode:

if(flag\_for\_theta\_p){

for(i=0;i<theta\_p\_arr.length;i++){

if (theta\_p\_arr[i] < 0.331613){

result = result + Math.abs(theta\_p\_arr[i]);

}

else{

result = result + 0;

}

}

}

else {

calculate\_theta\_p();

for(i=0;i<theta\_p\_arr.length;i++){

if (theta\_p\_arr[i] < 0.331613){

result = result + Math.abs(theta\_p\_arr[i]);

}

else{

result = result + 0;

return result;

* Details: This feature calculates the curviness of the gesture. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: This is same as the 10th feature of Rubine, basically telling how curvy a shape is. It was a good feature to include in the original subset, since an arrow is a sharp figure, not very curvy, but still the accuracy of this feature was not as good as some other features and therefore it was not chosen in the optimal subset.

1. Non Subjective openness

* Psuedocode:

if(f5)

{

if(diagonal)

{

return (f5/diagonal);

}

else

{

feature\_bb\_diagonal\_len();

return (f5/diagonal);

}

}

else

{

feature\_dist\_start\_end();

if(diagonal)

{

return (f5/diagonal);

}

else

{

feature\_bb\_diagonal\_len();

return (f5/diagonal);

* Details: This feature gives us a more relative measure of the openness of the gesture. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: This feature is very similar to the feature 3 of Rubine. Since we already kept that feature in our optimal subset, we didn't consider this, since both basically tell us the same thing, openness of a figure.

1. Density metric 1

* Psuedocode:

if(total\_stroke\_length)

if(f5)

return (total\_stroke\_length/f5);

else

feature\_dist\_start\_end();

return (total\_stroke\_length/f5);

else

feature\_total\_stroke\_len();

if(f5)

return (total\_stroke\_length/f5);

else

feature\_dist\_start\_end();

return (total\_stroke\_length/f5);

* Details: This feature tells us about the density of the gesture wrt the distance between the endpoints. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: This feature tells us the density or the ink of the figure between first and last point. Now arrows that way are less dense figures, with their entries stroke length and the distance between endpoints being almost the same. Therefore this feature is unique to them compared to other shapes and therefore this is a optimal feature.

1. Density Metric 2

* Psuedocode:

if(total\_stroke\_length)

{

if(diagonal)

{

return (total\_stroke\_length/diagonal);

}

else

{

feature\_bb\_diagonal\_len();

return (total\_stroke\_length/diagonal);

}

}

else

{

feature\_total\_stroke\_len();

if(diagonal)

{

return (total\_stroke\_length/diagonal);

}

else

{

feature\_bb\_diagonal\_len();

return (total\_stroke\_length/diagonal);

* Details: This feature gives us a density measure of the gesture. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: Similar to the density metric 1 , this feature also holds important information which is unique to arrows therefore this feature was also picked in the optimal subset.

1. Log of aspect Ratio

* Psuedocode:

if(aspect\_ratio)

{return(Math.log(aspect\_ratio));}

else{

feature\_aspect\_ratio();

return(Math.log(aspect\_ratio));

* Details: This feature gives us the log value of the aspect ratio of the gesture. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: This feature makes all tall skinny shapes similar to flat long shapes. This is important since similar shapes should have similar values. However, this doesn't seem very intuitive, since it's not solely specific to any geometrical property of an arrow. Therefore this feature was not chosen in the final subset.

1. Log of total length

* Psuedocode:

if(total\_stroke\_length)

{return(Math.log(total\_stroke\_length));}

else

feature\_total\_stroke\_len();

return(Math.log(total\_stroke\_length));

* Details: This feature gives us the log value of the total stroke length of the gesture. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: This feature provides us almost the same information as the total stroke length and since total stroke length was included in our optimal subset, it made sense to keep the log of stroke length in the optimal subset too. This feature provided us high accuracy.

1. Relative Rotation

* Psuedocode:

if(flag\_for\_theta\_p){

for(i=0;i<theta\_p\_arr.length;i++){

result = result + theta\_p\_arr[i];

else

calculate\_theta\_p();

for(i=0;i<theta\_p\_arr.length;i++){

result = result + theta\_p\_arr[i];

if(total\_stroke\_length){return (result/total\_stroke\_length);}

else{feature\_total\_stroke\_len(sketch); return (result/total\_stroke\_length);}

* Details: This feature gives us the relative rotation of the gesture. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: This feature provides us all the details feature 9 of Rubine provides. It's just that the rotation is scaled wrt to the total stroke length. This doesn't seem a very valuable to arrow recognition. For the same reasons that we didn't pick feature 9 of Rubine, we didn't pick this feature either in our optimal subset.

1. Total angle by absolute angle

* Psuedocode:

if(total\_angle)

if(total\_abs\_angle)

if(total\_abs\_angle > 0)

{return (total\_angle/total\_abs\_angle);}

else

{ return 0;}

else

feature\_total\_abs\_angle();

if(total\_abs\_angle > 0)

{return (total\_angle/total\_abs\_angle);}

else

{ return 0;}

else

feature\_total\_angle();

if(total\_abs\_angle)

if(total\_abs\_angle > 0)

{return (total\_angle/total\_abs\_angle);}

else { return 0;}

else

feature\_total\_abs\_angle();

if(total\_abs\_angle > 0)

{return (total\_angle/total\_abs\_angle);}

else

{ return 0;}

* Details: This feature gives us ratio of rotational change to the rotational motion of the gesture. This is a Long feature . All details (including the calculation equation)of the feature was taken from **[2].**
* Justification: This feature basically depicts the rotational change compared to the rotational motion. This is not very significant for an arrow. It has very less rotation involved in it. Therefore, this feature was not included in the optimal subset.

1. Threshold lines arrow centric

* Psuedocode:

if(newstrokes.length <=3) {return 1;}

else {return 0;}

* Details: This feature checks if the number of strokes in a figure and less than or equal to 3.
* Justification: The thinking behind this feature is that people either draw an arrow in a single stroke or maximum draw three strokes to draw an arrow. I thought it might help in recognizing an arrow. Unfortunately, the accuracy of this feature was very low. Possibly the thought process behind the feature isn't all that valid.

1. NDDE

* Psuedocode:

for(i=0;i<newstrokes.length;i++){

for (j=0;j<(newstrokes[i].length)-1;j++){

x\_temp = newstrokes[i][j+1].x - newstrokes[i][j].x;

y\_temp = newstrokes[i][j+1].y - newstrokes[i][j].y;

if(x\_temp!=0)

{final\_add = y\_temp/x\_temp; z++;

avg\_change = avg\_change + final\_add;

Keep track of max and min here }

avg\_change = avg\_change/z;

if(avg\_change>0 && isFinite(avg\_change))

{DCR = max/avg\_change;}

else DCR = 0;

final\_add = 0; x\_temp = 0; y\_temp = 0; var t =0;

if(xpmin>xpmax)

{t=xpmin; xpmin=xpmax; xpmax =t; t=ypmin; ypmin=ypmax; ypmax =t;}

for(i=xpmin;i<newstrokes.length;i++){

for (j= 0;j<(newstrokes[i].length)-1;j++){

if(flag\_first\_run == 0)

{j = ypmin; flag\_first\_run++;}

x\_temp = newstrokes[i][j+1].x - newstrokes[i][j].x;

y\_temp = newstrokes[i][j+1].y - newstrokes[i][j].y;

final\_add = final\_add + (Math.sqrt(Math.pow(x\_temp,2) + Math.pow(y\_temp,2)));

if(i == xpmax && j == ypmax) break;}

if(total\_stroke\_length) {return (final\_add/total\_stroke\_length);}

else {feature\_total\_stroke\_len(); return (final\_add/total\_stroke\_length);}

* Details: This feature gives the normalized distance between direction extremes. All details (including the calculation equation)of the feature was taken from **[3].**
* Justification: This feature was chosen since **[3]** has shown that NDDE gives low values for polylines and arrow is a polyline. This feature was not chosen in the final optimal subset, since it didn't help out a lot in accuracy. Possibly because the other data/shapes also had a lot of polylines in it as well.

1. DCR

* Pusedocode:

if(DCR) {return DCR;}

else {feature\_NDDE(); return DCR;}

* Details: This feature gives the direction change ratio for a gesture . All details (including the calculation equation)of the feature was taken from **[3].**
* Justification: This feature was chosen since **[3]** has shown that DCR gives high values for polylines and arrow is a polyline. This feature was not chosen in the final optimal subset, since it didn't help out a lot in accuracy. Possibly because the other data/shapes also had a lot of polylines in it as well or could be that there isn't relatively a lot of direction change that happens in an arrow.

1. Number of strokes

* Psuedocode: return newstrokes.length;
* Details: This feature returns the number of strokes in a figure. All details of the feature was taken from **[4].**
* Justification: The intuition behind this feature was that a complex figure will require more lines/strokes for construction compared to an arrow. Unfortunately this feature provided very low accuracy values. Therefore it could not be included in the final subset. Possibly, number of strokes is a very vague differentiator between shapes/figures, since different people draw in different ways.

1. Percentage of strokes passing the line test

* Psuedocode:

for(i = 0;i<newstrokes.length;i++)

{

x\_0 = newstrokes[i][0].x;

y\_0 = newstrokes[i][0].y;

x\_p = newstrokes[i][newstrokes[i].length -1].x;

y\_p = newstrokes[i][newstrokes[i].length -1].y;

start\_to\_end = Math.sqrt(Math.pow((x\_p-x\_0),2) + Math.pow((y\_p-y\_0),2) );

for(j=0;j<newstrokes[i].length - 1;j++)

{

x\_temp = newstrokes[i][j+1].x - newstrokes[i][j].x;

y\_temp = newstrokes[i][j+1].y - newstrokes[i][j].y;

final\_add = Math.sqrt(Math.pow(x\_temp,2) + Math.pow(y\_temp,2));

result = result + final\_add;

}

if(start\_to\_end/result >= 0.90)

{pass++;}} return (pass/newstrokes.length);

* Details: This features basically tests if each stroke passes the line test i.e. the ratio of start to end distance by the stroke length. If passed, 1 is returned otherwise 0 value is returned. Passing means the ratio value should be greater than 0.90. All details of the feature was taken from **[4].**
* Justification: The thinking behind this feature is that an arrow is generally made of three straight lines. Of course this feature isn't significant for curvy arrows. The accuracy of this feature was relatively decent, still not very good. It was not included in the final subset.

1. Log of bounding box diagonal ---- log(arctan((ymax-ymin)/(xmax-xmin)))

* Psuedocode:

if(diagonal){ return (Math.log(diagonal));}

else{feature\_bb\_diagonal\_len(); return (Math.log(diagonal));}

* Details: This feature provides the log value of the bounding box diagonal.
* Justification: The thinking behind this was inspired from Dr. Hammond's lecture, where she mentioned that instead of log of area of the bounding box , log of the bounding box diagonal might be more intuitive, since that brings about all similar shapes in the same range. This feature had high accuracy and was included in the final subset.

**Feature Results:**

|  |  |  |
| --- | --- | --- |
| **Optimal Subset** | **Classifier** | **Correctly classified Instances(%)** |
| 10 features | ZeroR | 50.025 |
|  | J48 | 98.4 |
|  | Random Forest | 99.25 |
|  | Part | 97.26 |
|  |  |  |
| **Original Set** | **Classifier** | **Correctly classified Instances (%)** |
| 30 features | ZeroR | 50.025 |
|  | J48 | 98.89 |
|  | Random Forest | 99.45 |
|  | Part | 99.064 |

So as we can see from the above two tables, the accuracy was tested across 4 classifiers -- ZeroR, J48, Random Forest, Part. The maximum accuracy achieved on the full set of features, i.e. 30 features was 99.45 % (Random Forest) and the maximum accuracy achieved on the optimal subset of features, i.e. 10 features was 99.25 % (Random Forest). Therefore we can see that we were able to capture almost the same accuracy with 10 features, instead of 30 features. This proves that our optimal subset actually consist of the most optimal features out of the total lot.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Classifier** | **Correctly classified Instances(%)** | **In optimal subset** |
| **Area of bounding box** | ZeroR | 50.025 | Yes |
|  | J48 | 91.8793 |  |
|  | Random Forest | 96.99 |  |
|  | Part | 85.97 |  |
| Arrow-Centric | ZeroR | 50.025 | No |
|  | J48 | 54.94 |  |
|  | Random Forest | 54.94 |  |
|  | Part | 54.94 |  |
| Aspect | ZeroR | 50.025 | No |
|  | J48 | 88.62 |  |
|  | Random Forest | 96.15 |  |
|  | Part | 74.55 |  |
| Bounding\_Box angle | ZeroR | 50.025 | No |
|  | J48 | 88.33 |  |
|  | Random Forest | 96.3 |  |
|  | Part | 79.38 |  |
| **Bounding box length** | ZeroR | 50.025 | Yes |
|  | J48 | 93.7 |  |
|  | Random Forest | 97.51 |  |
|  | Part | 88.8 |  |
| Cos of start angle | ZeroR | 50.025 | No |
|  | J48 | 82.5 |  |
|  | Random Forest | 89.6 |  |
|  | Part | 73.3 |  |
| Cos start end angle | ZeroR | 50.025 | No |
|  | J48 | 88.2 |  |
|  | Random Forest | 95.5 |  |
|  | Part | 74.4 |  |
| Curviness | ZeroR | 50.025 | No |
|  | J48 | 85.51 |  |
|  | Random Forest | 95.7 |  |
|  | Part | 73.3 |  |
| DCR | ZeroR | 50.025 | No |
|  | J48 | 78.14 |  |
|  | Random Forest | 80.4 |  |
|  | Part | 69.17 |  |
| **Density\_metric1** | ZeroR | 50.025 | Yes |
|  | J48 | 90.4 |  |
|  | Random Forest | 97.3 |  |
|  | Part | 85.8 |  |
| **Density\_metric2** | ZeroR | 50.025 | Yes |
|  | J48 | 91.6 |  |
|  | Random Forest | 97.7 |  |
|  | Part | 88.3 |  |
| **Distance\_start\_to\_end** | ZeroR | 50.025 | Yes |
|  | J48 | 90.1 |  |
|  | Random Forest | 96.7 |  |
|  | Part | 81.9 |  |
| **Log of area of bounding box** | ZeroR | 50.025 | Yes |
|  | J48 | 91.8 |  |
|  | Random Forest | 97.02 |  |
|  | Part | 86.1 |  |
| log of aspect ratio | ZeroR | 50.025 | No |
|  | J48 | 89.2 |  |
|  | Random Forest | 96.15 |  |
|  | Part | 74.2 |  |
| **log of bounding box diagonal** | ZeroR | 50.025 | Yes |
|  | J48 | 93.7 |  |
|  | Random Forest | 97.5 |  |
|  | Part | 88.74 |  |
| **Log of total stroke length** | ZeroR | 50.025 | Yes |
|  | J48 | 94.8 |  |
|  | Random Forest | 98.1 |  |
|  | Part | 91.66 |  |
| **Max speed of gesture** | ZeroR | 50.025 | Yes |
|  | J48 | 81.5 |  |
|  | Random Forest | 93.1 |  |
|  | Part | 63.37 |  |
| NDDE | ZeroR | 50.025 | No |
|  | J48 | 77.6 |  |
|  | Random Forest | 94.7 |  |
|  | Part | 56.6 |  |
| Number of strokes | ZeroR | 50.025 | No |
|  | J48 | 65.1 |  |
|  | Random Forest | 65.1 |  |
|  | Part | 65.1 |  |
| Openness | ZeroR | 50.025 | No |
|  | J48 | 88.44 |  |
|  | Random Forest | 92.1 |  |
|  | Part | 83.1 |  |
| Relative Rotation | ZeroR | 50.025 | No |
|  | J48 | 75.4 |  |
|  | Random Forest | 95.3 |  |
|  | Part | 65.8 |  |
| Sin start angle | ZeroR | 50.025 | No |
|  | J48 | 82 |  |
|  | Random Forest | 89.6 |  |
|  | Part | 73.9 |  |
| Sin of angle start to end | ZeroR | 50.025 | No |
|  | J48 | 88.3 |  |
|  | Random Forest | 95.5 |  |
|  | Part | 75.4 |  |
| Strokes passing line test | ZeroR | 50.025 | No |
|  | J48 | 72.1 |  |
|  | Random Forest | 72.1 |  |
|  | Part | 72.1 |  |
| Total absolute angle | ZeroR | 50.025 | No |
|  | J48 | 87 |  |
|  | Random Forest | 95.7 |  |
|  | Part | 73.9 |  |
| Total angle | ZeroR | 50.025 | No |
|  | J48 | 84.1 |  |
|  | Random Forest | 95.8 |  |
|  | Part | 68.96 |  |
| Total angle by absolute angle | ZeroR | 50.025 | No |
|  | J48 | 84.4 |  |
|  | Random Forest | 94.2 |  |
|  | Part | 63.4 |  |
| Total squared angle | ZeroR | 50.025 | No |
|  | J48 | 85.2 |  |
|  | Random Forest | 95.7 |  |
|  | Part | 71.15 |  |
| **Total\_stroke\_length** | ZeroR | 50.025 | Yes |
|  | J48 | 94.8 |  |
|  | Random Forest | 98.1 |  |
|  | Part | 91.6 |  |
| Total\_time | ZeroR | 50.025 | No |
|  | J48 | 86.3 |  |
|  | Random Forest | 91.2 |  |
|  | Part | 79 |  |

As we can see from the above table, the high accuracy values across 4 classifiers helped us pick the 10 most optimal features i.e. -- Area of bounding box, Length of bounding box diagonal, Density metric 1 and 2, distance between starting and end point, Log of area of bounding box, log of diagonal length of bounding box, log of stroke length, maximum speed, total stroke length. **For explanation on why each of these 10 features were pricked kindly refer to the justification provided in section 2.** In general, we can see the bounding box proved to be a very important attribute, which catches a lot of information, which in turn helps us in recognizing arrows. Maximum speed also proved to be quite a valuable feature, since arrows are generally drawn quickly. Density Metrics were also helpful features, since the density of arrows is quite light and placed almost entirely between the two endpoints.

**References:**

**[1]** Specifying Gestures by example by Dean Rubine

**[2]** A. Chris Long, Jr., James A. Landay, Lawrence A. Rowe, Joseph Michiels. Visual similarity of pen gestures.

**[3]** Brandon Paulson and Tracy Hammond. "PaleoSketch: Accurate Primitive Sketch Recognition and Beautification".

**[4]** Brandon Paulson, Pankaj Rajan, Pedro Davalos, Ricardo Gutierrez-Osuna, Tracy Hammond. "What!?! No Rubine Features?: Using Geometric-based Features to Produce Normalized Confidence Values for Sketch Recognition".