SMAI Assignment 2

Naila Fatima
201530154

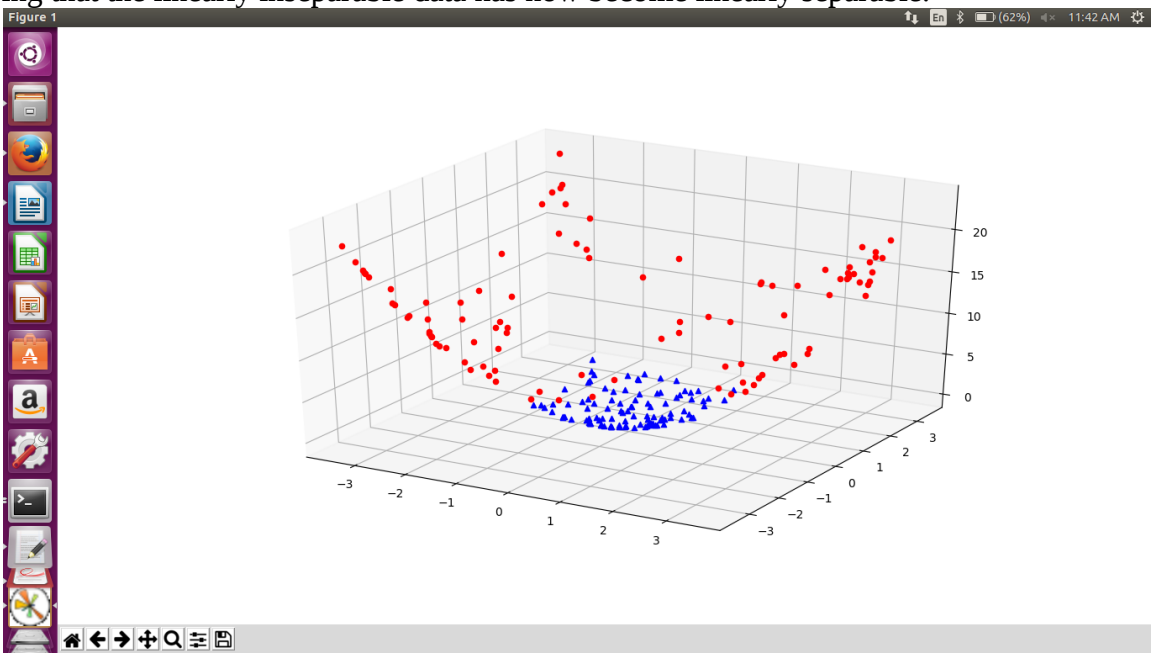
Question 1:

a) Kernel Trick

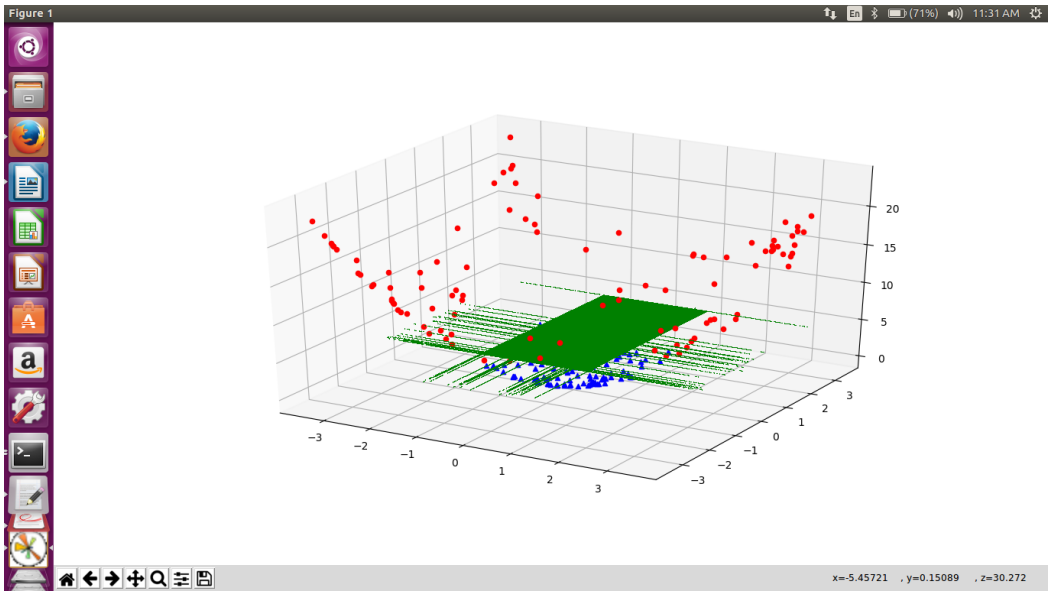
Kernel 1:

The first kernel used maps a 2d input (x1,x2) to three dimensions (x1, x2, $x1^2 + x2^2$). The data is linearly separable when mapped to these dimensions and the training accuracy is 1.000 showing that the data is linearly separable.

The perceptron model used below uses a L1 penalty with an alpha equal to 0.01. The points which are mapped to 3d can easily be seen by the following figure. We can see that it is possible to separate the red and blue points (which belong to different classes) by using a hyperplane thereby showing that the linearly inseparable data has now become linearly separable.



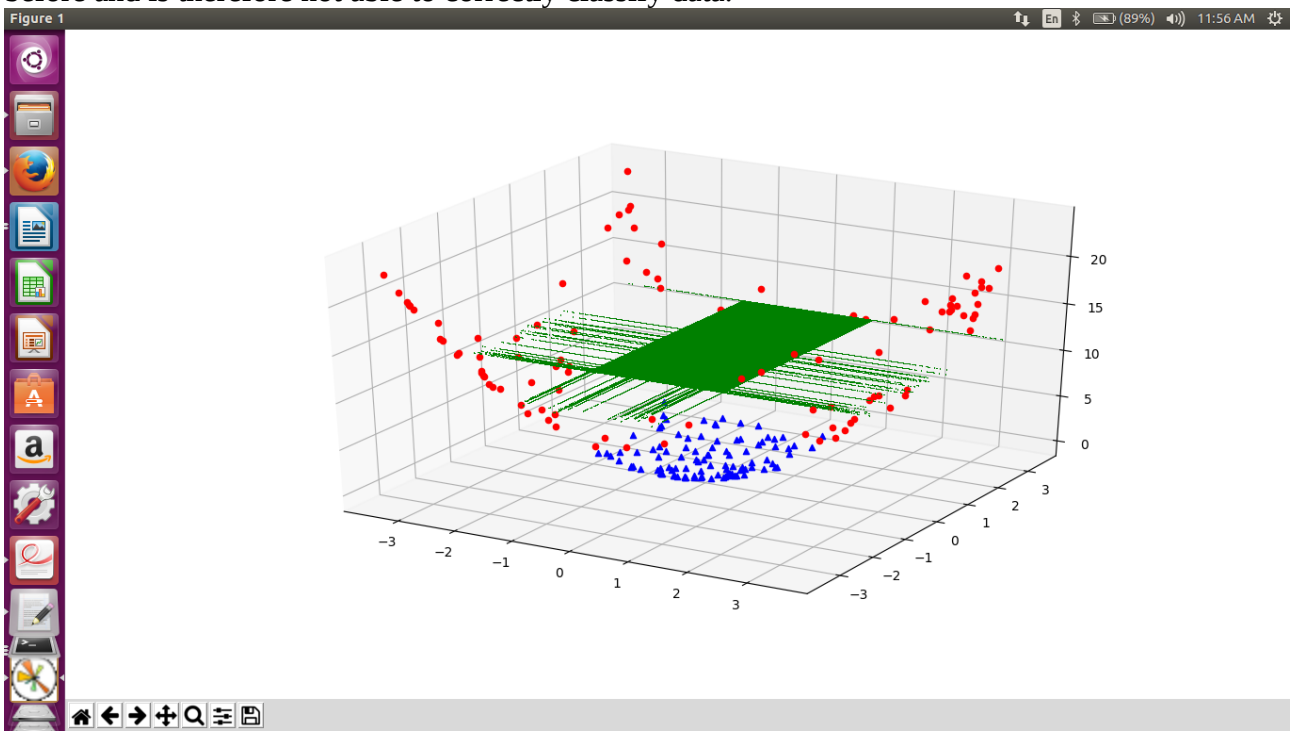
It can clearly be seen that the data is now linearly separable by the green hyperplane (the 3d decision surface)



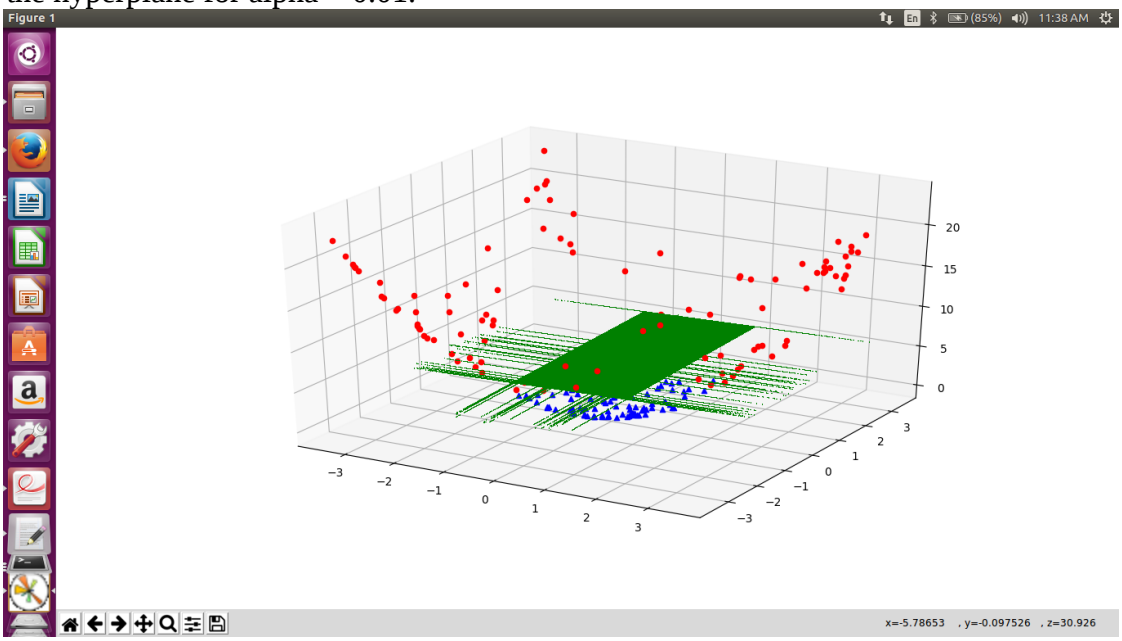
The training accuracy, precision, recall and f1 score are also 1.000.

I have noticed that on using the L1 penalty model, when alpha is increased, the training accuracy, precision, recall and f1 scores are no longer 1 and therefore the data is no longer linearly separable. For example, for alpha = 0.1, the accuracy, precision, recall and f1 scores become 0.9700, 0.9400, 1.0000 and 0.9691. I have observed that on increasing alpha above 0.075, the data is no longer linearly separable. At alpha = 0.074, the data can be separated by a hyperplane and all metrics are 1.000 whereas for alpha = 0.075, the metrics reduce to become 0.8200, 0.6400, 1.0000 and 0.7805. Any value of alpha less than 0.075 will give linearly separable data.

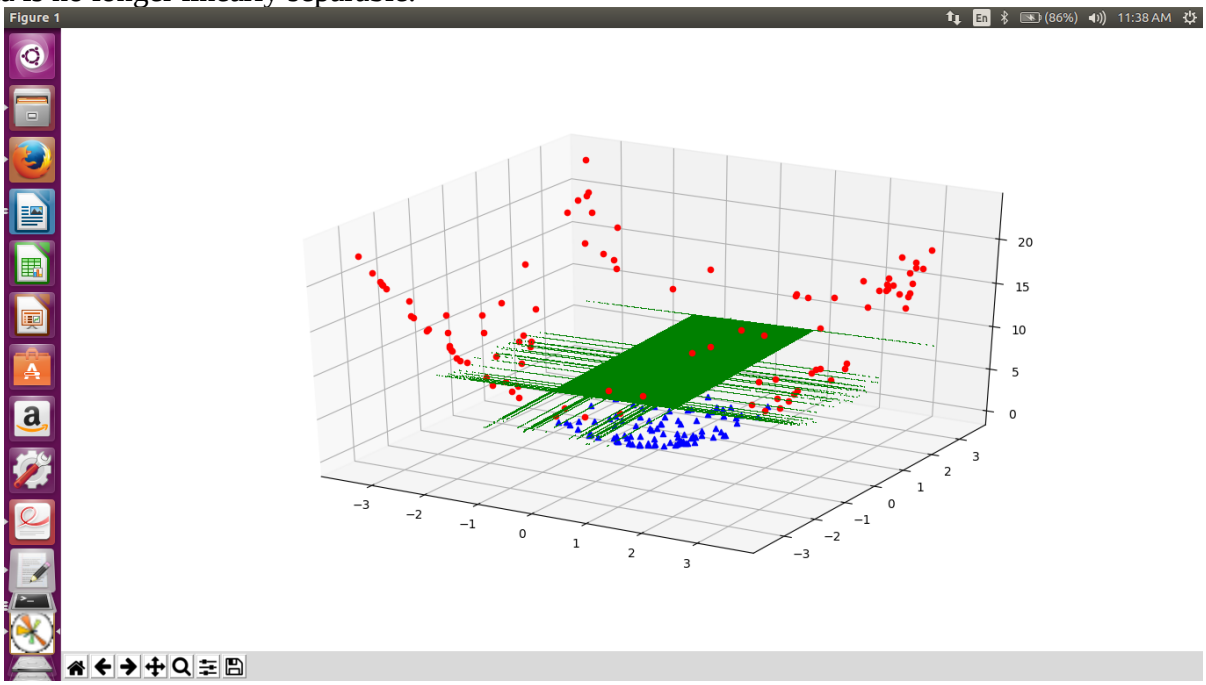
For alpha = 0.075, the hyperplane is shown below. We can see that it is much higher than before and is therefore not able to correctly classify data.



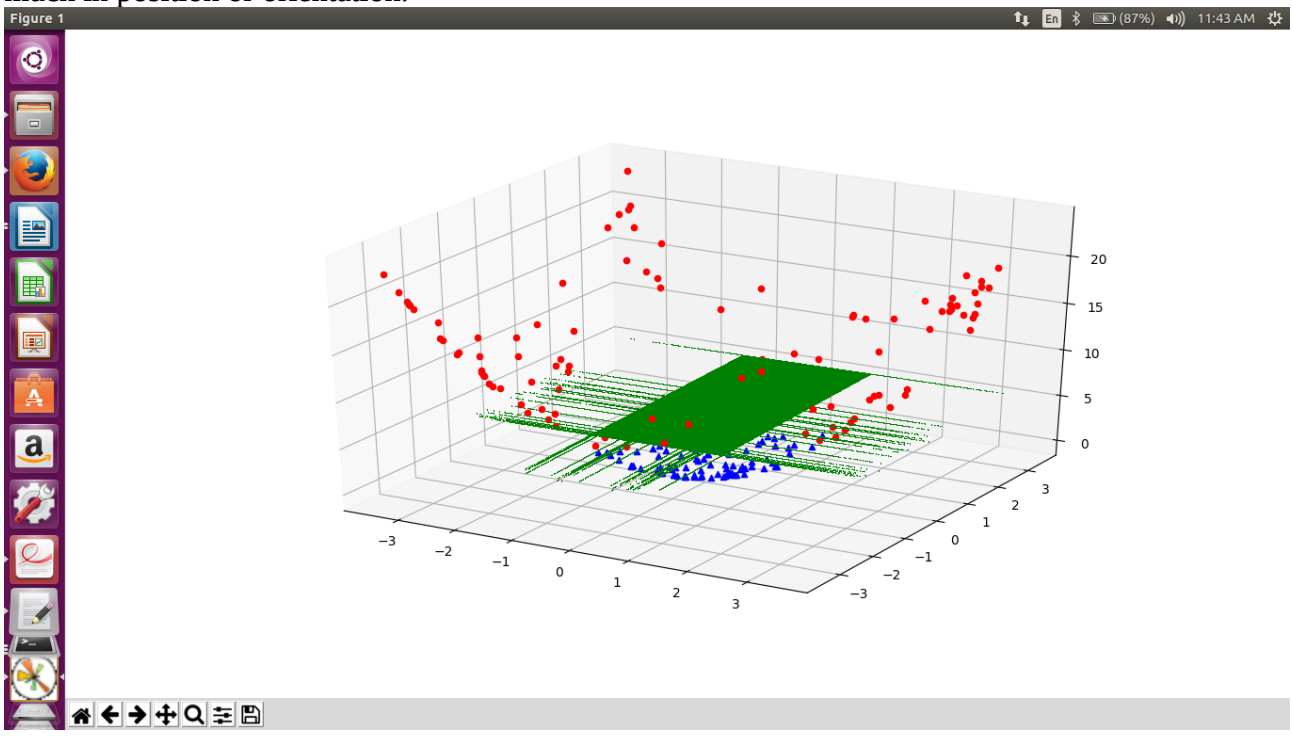
On using the L2 penalty model, the data is separable as long as alpha is under 0.01. When alpha is under 0.01, the accuracy, precision, recall and f1 scores are all 1.000. The below picture shows the hyperplane for alpha = 0.01.



We can easily see that on increasing alpha to 0.02, the hyperplane rises up a bit therefore causing misclassifications. The metrics become 0.9900, 0.9800, 1.0000 and 0.9899 showing that the data is no longer linearly separable.

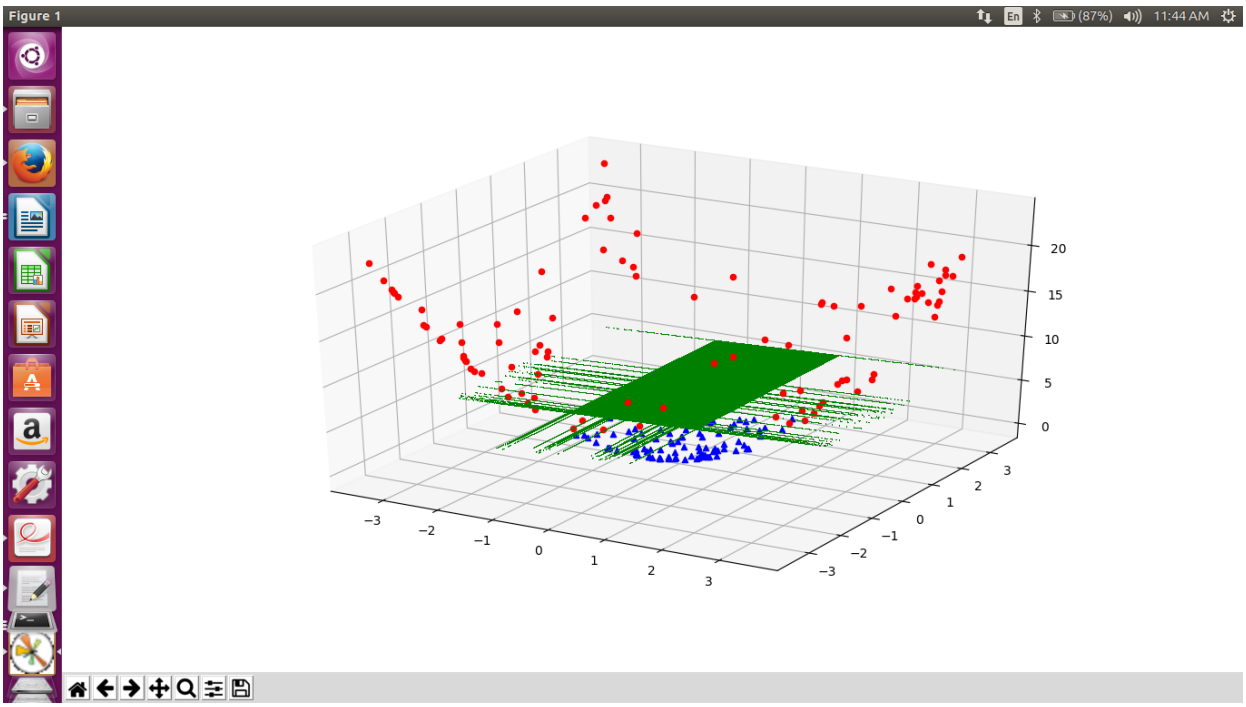


For alpha = 0.00001, the picture below shows that the hyperplane does not seem to change much in position or orientation.



On using a model with no penalty, the data becomes linearly separable. Note that on using no penalty, there is no concept of alpha (a term which is used for regularization). We can note that our training accuracy, precision, recall and f1 scores are 1.000.

For no penalty, the hyperplane is shown below

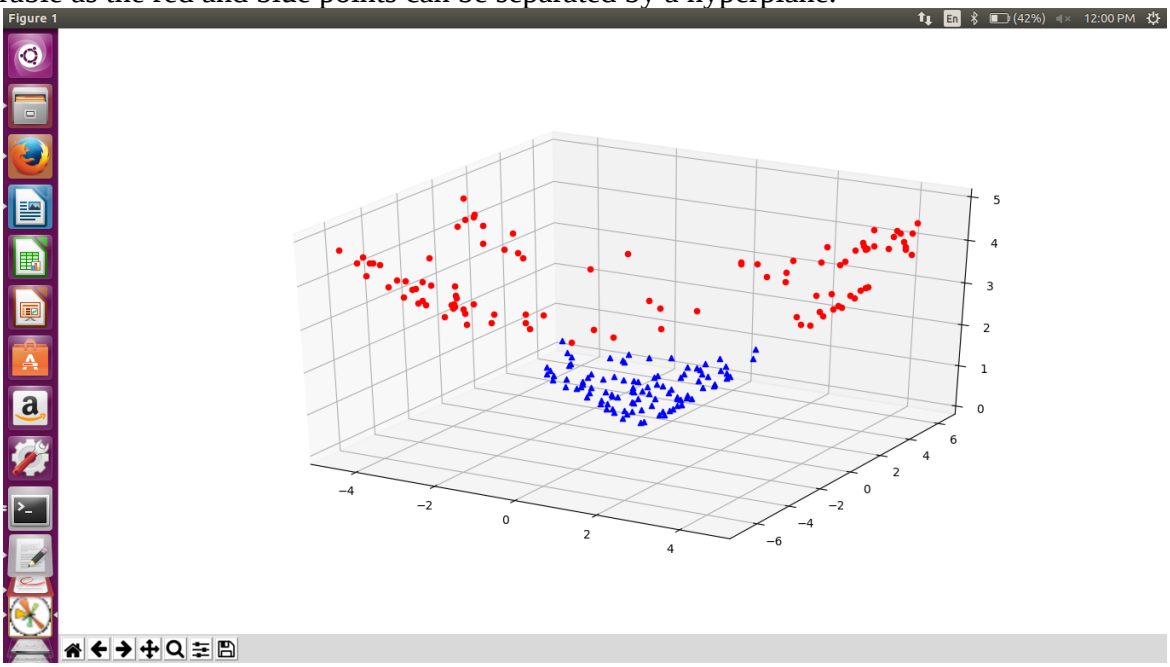


We can easily notice that the hyperplane rises much higher for L1 regularization as compared to L2 regularization thereby causing more misclassifications.

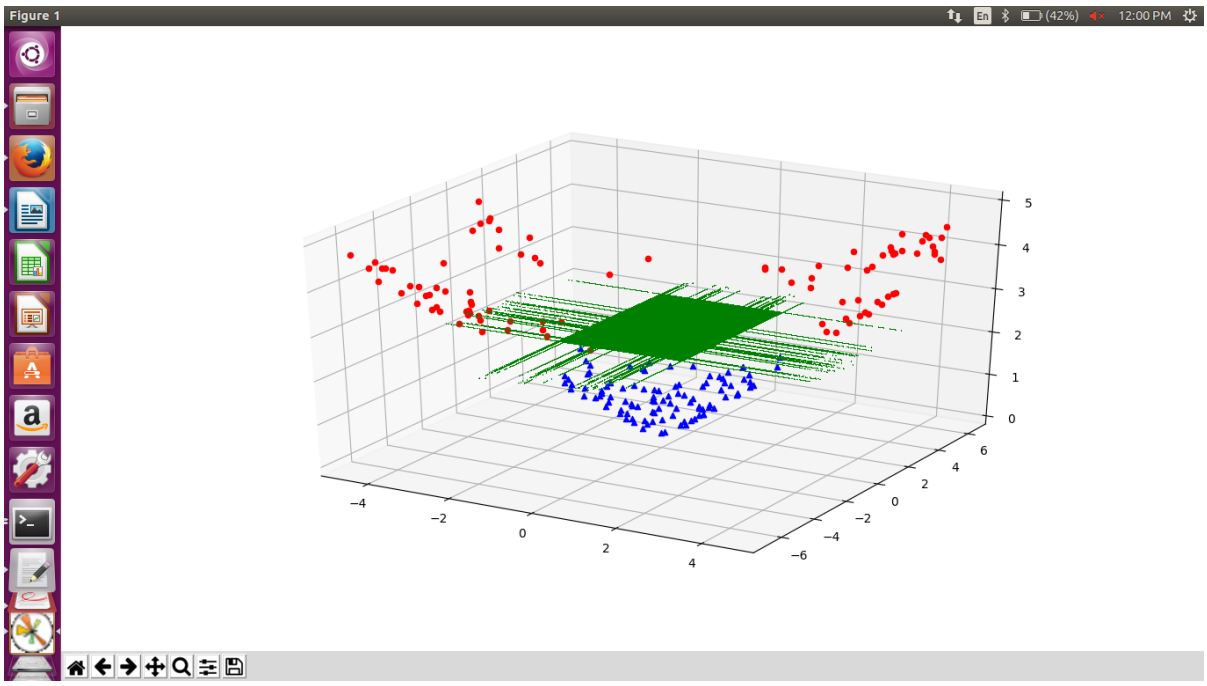
Kernel 2:

The second kernel used maps a given 2d input (x1,x2) to a 3d input given by (1.3*x1, x1+x2, sqrt($x1^2$+$x2^2$)). The data is linearly separable when mapped using this relation and therefore the training accuracy, precision, recall and f1 scores are all 1.0000.

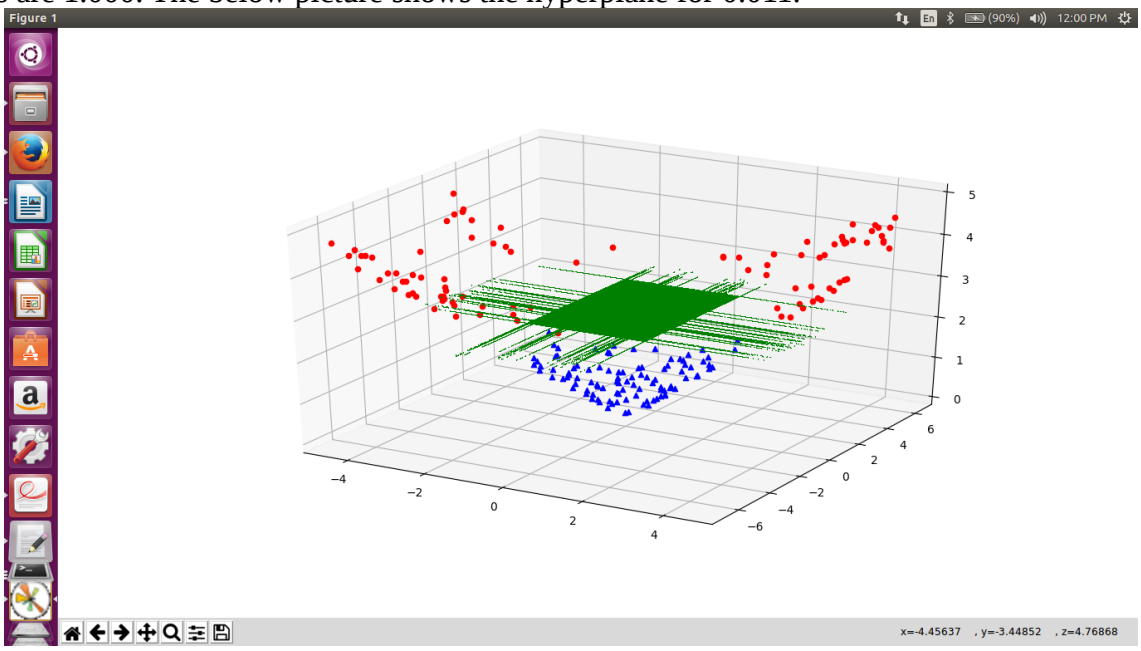
The perceptron model used below uses an L1 penalty with alpha = 0.01. The below picture shows the data mapped to 3d using the above kernel. We can see that the data is now linearly separable as the red and blue points can be separated by a hyperplane.



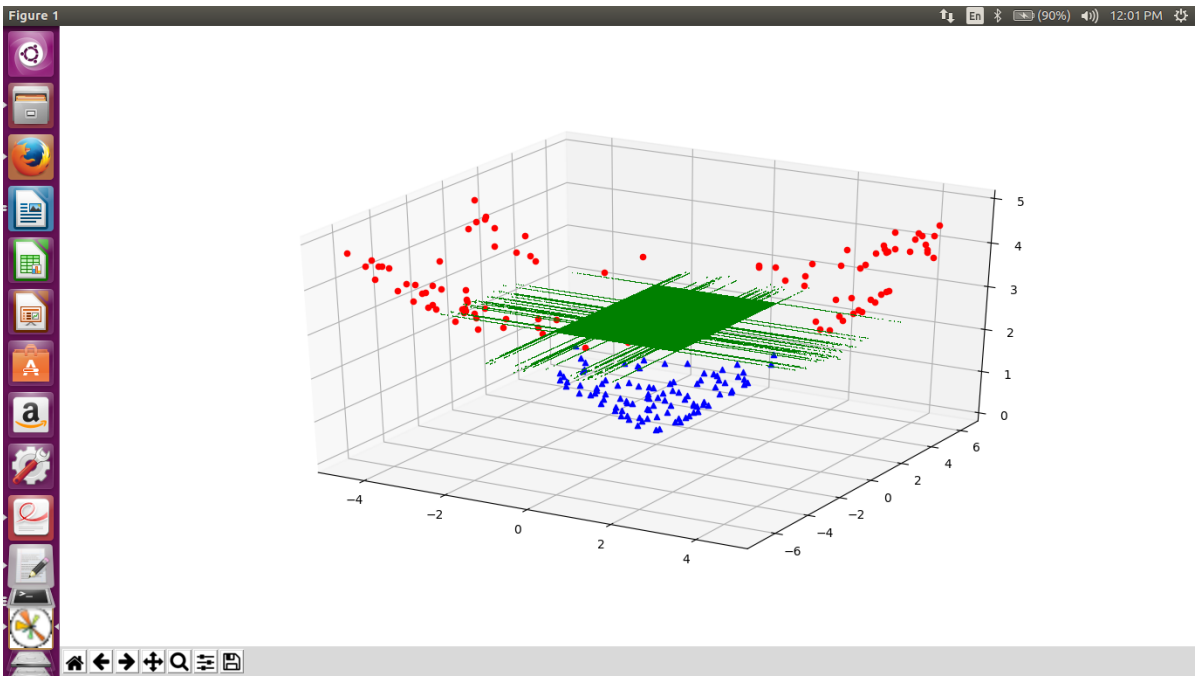
It can be easily seen that the data can be separated by a 3d decision surface (the green hyperplane).



On using L1 penalty, we note that any alpha less than 0.012 will give a hyperplane which classifies the data correctly. Up until alpha = 0.011, the training accuracy, precision, recall and f1 scores are 1.000. The below picture shows the hyperplane for 0.011.

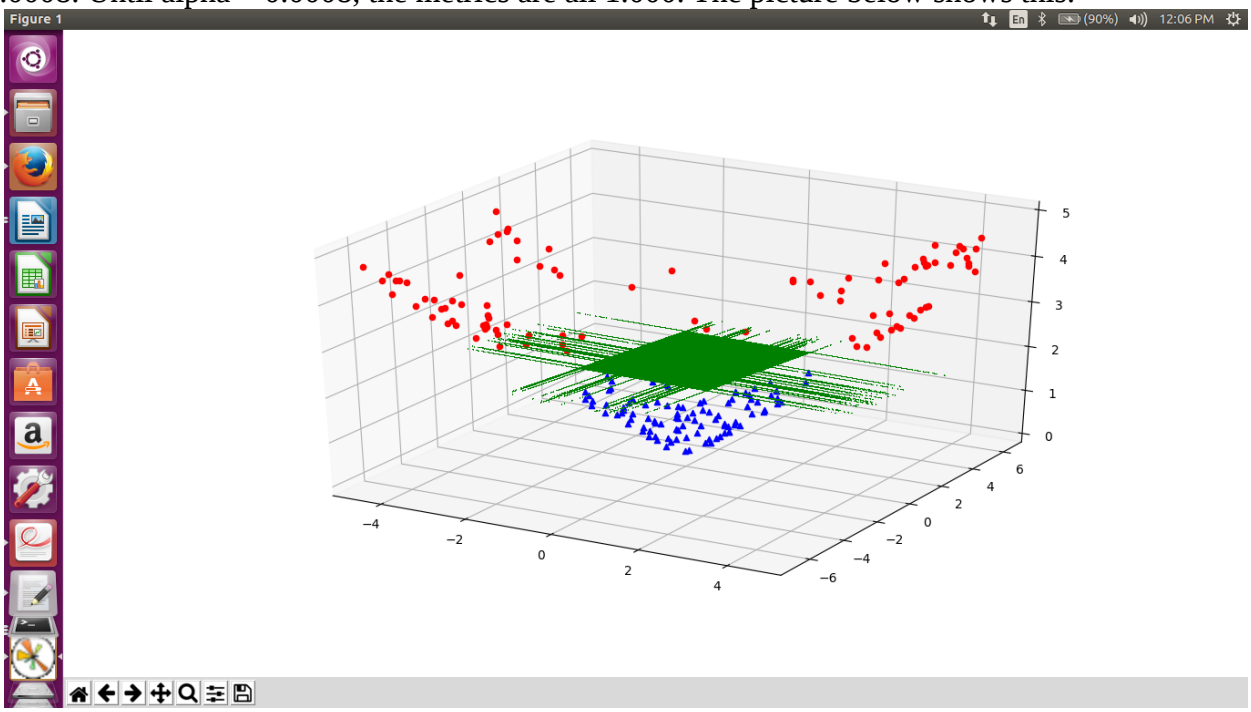


From alpha = 0.012, the kernel no longer classifies the data 100% correctly. The metrics become 0.9900, 0.9800, 1.0000 and 0.9899. The picture below depicts this.

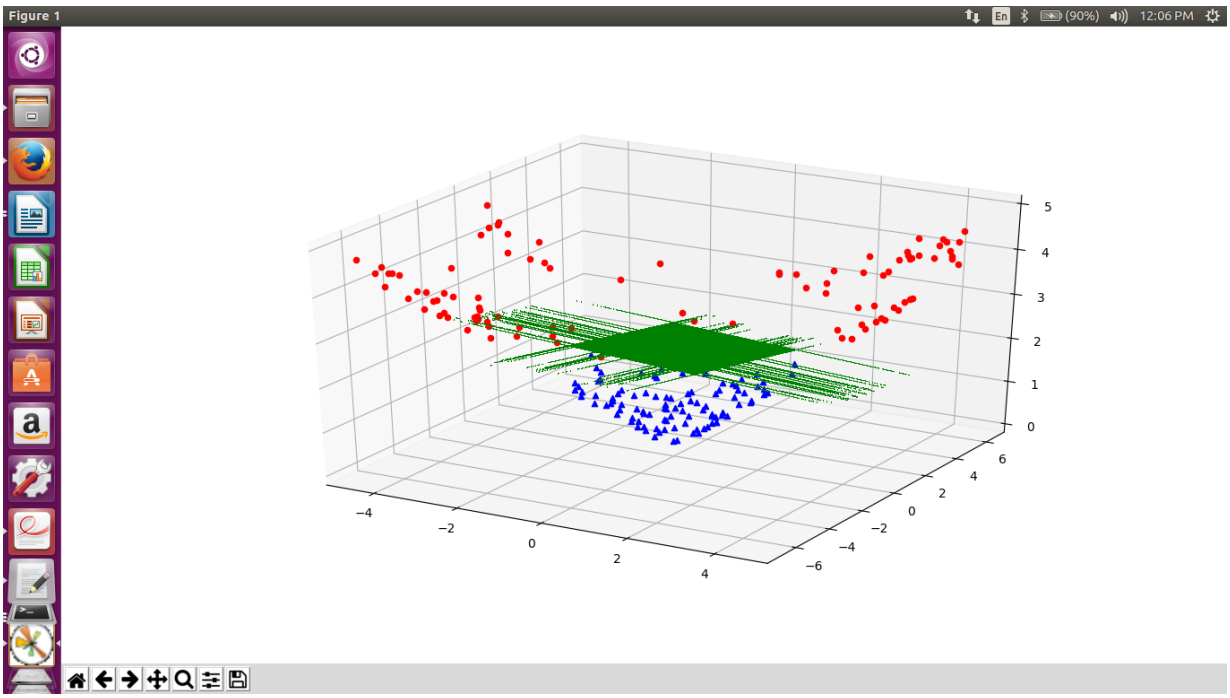


We can easily see that the hyperplane has risen upwards on increasing alpha.

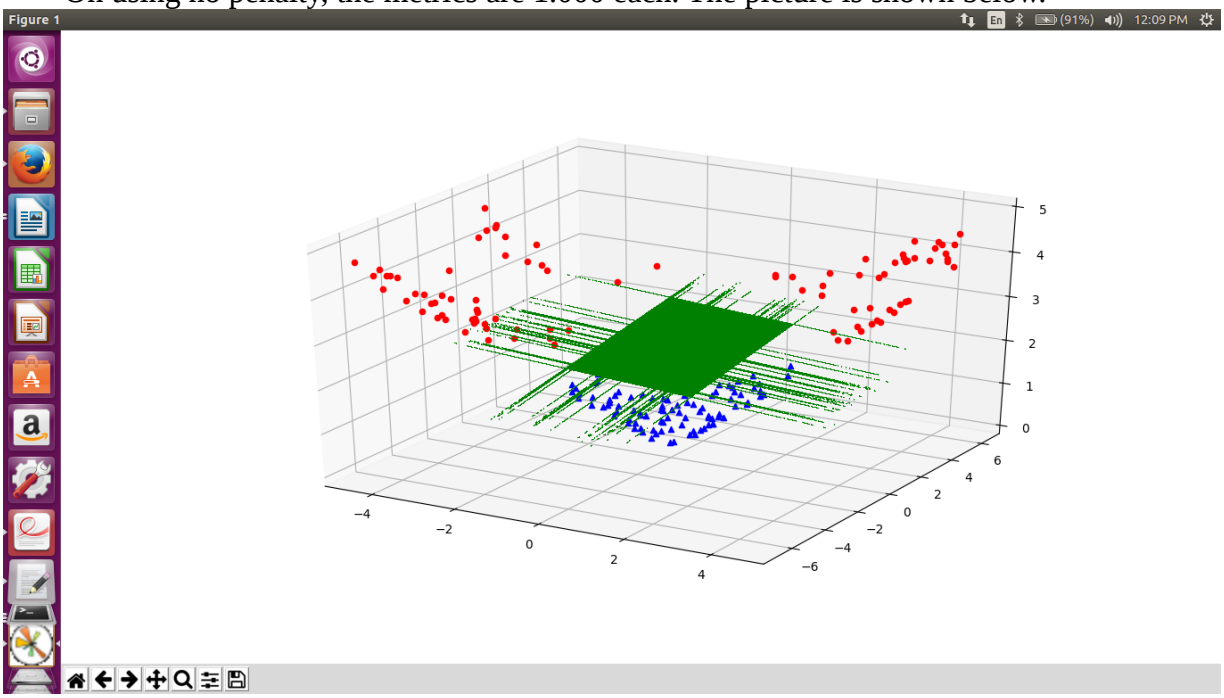
On using a L2 penalty, the hyperplane successfully separates the data up until alpha = 0.0008. Until alpha = 0.0008, the metrics are all 1.000. The picture below shows this.



For alpha = 0.0008, the metrics become 0.9950, 0.9900, 1.0000 and 0.9950. The hyperplane becomes a bit skewed in orientation as can be seen below.



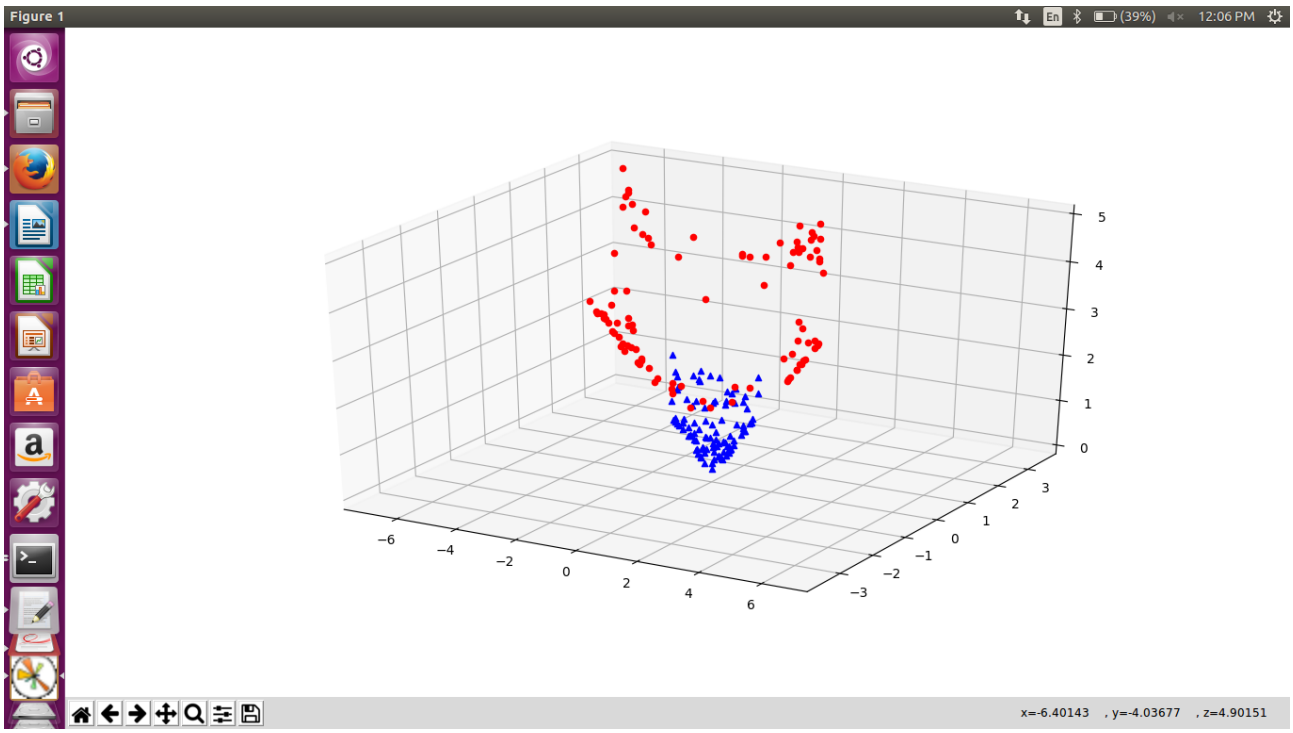
On using no penalty, the metrics are 1.000 each. The picture is shown below.



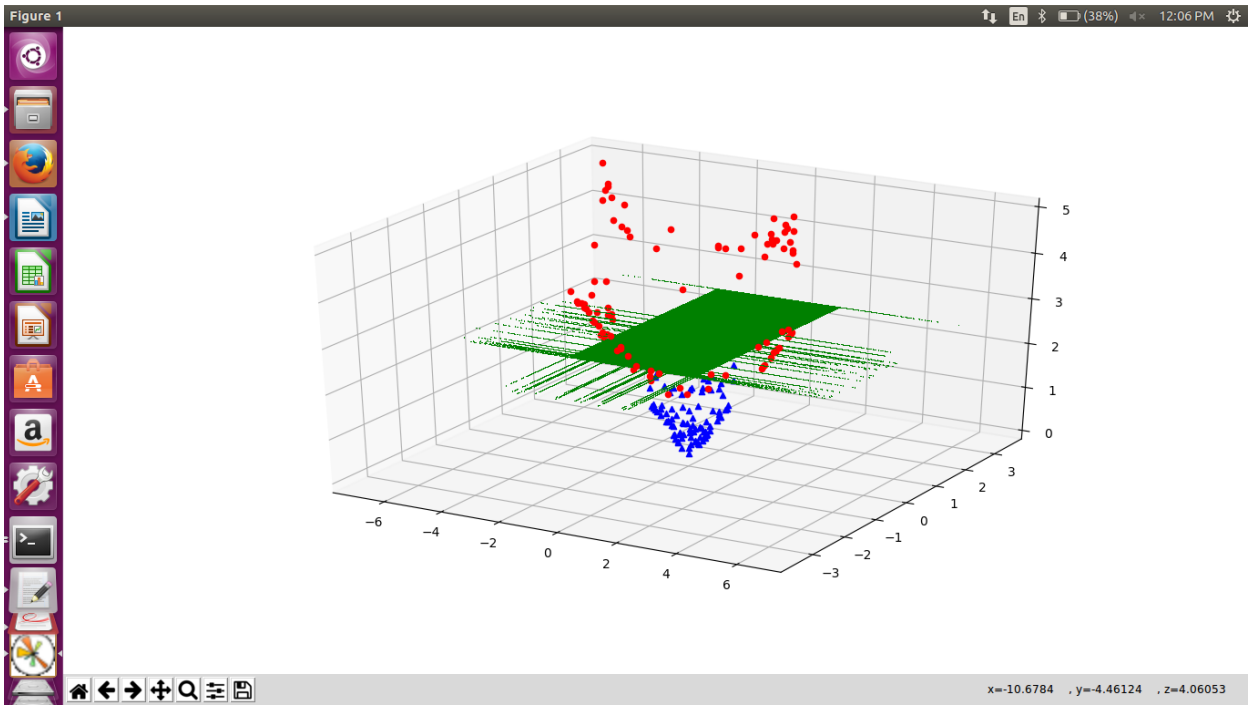
Kernel 3:

The third kernel used also maps a 2d data input (x1, x2) to 3 dimensions by using the relation (x1-x2, x2, sqrt($x1^2$+ $x2^2$)). The data becomes linearly separable when this kernel is used and therefore the training accuracy, precision, recall and f1 score become 1.000.

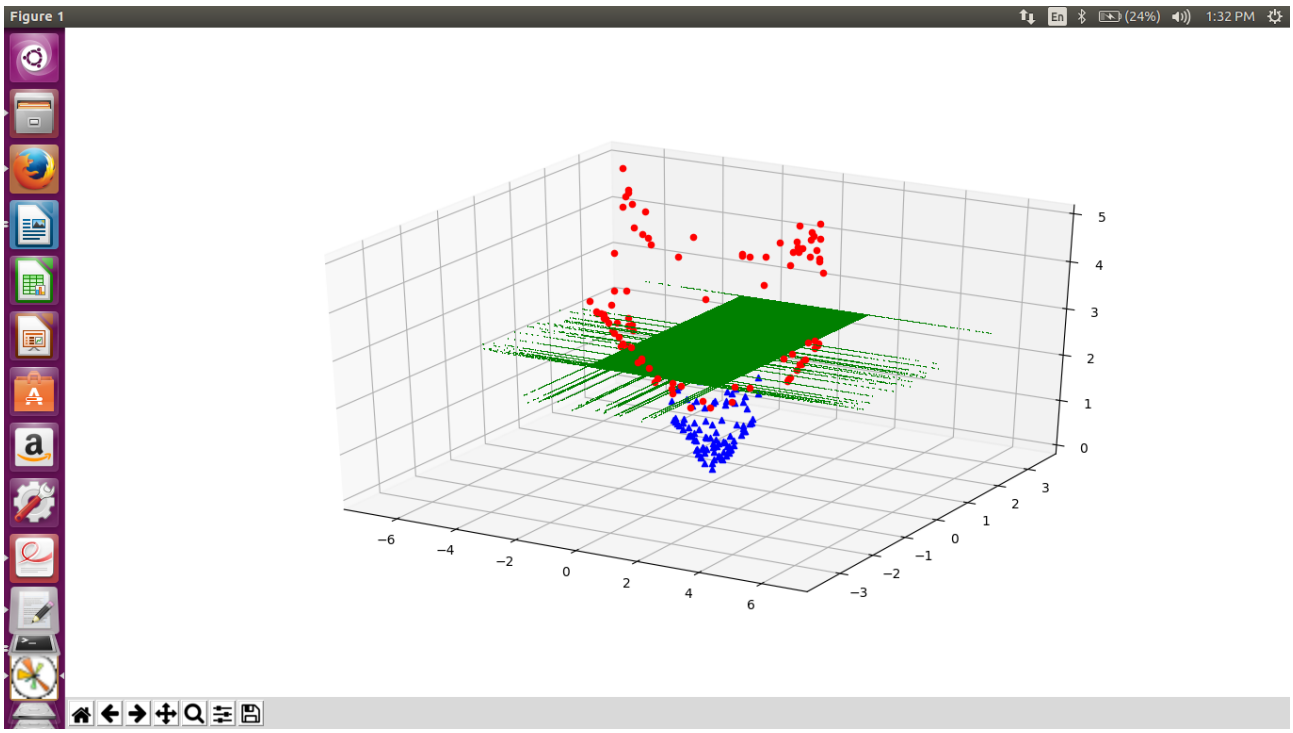
The perceptron model used below uses an L1 penalty with alpha = 0.01. The below picture shows the data mapped to 3 dimensions.



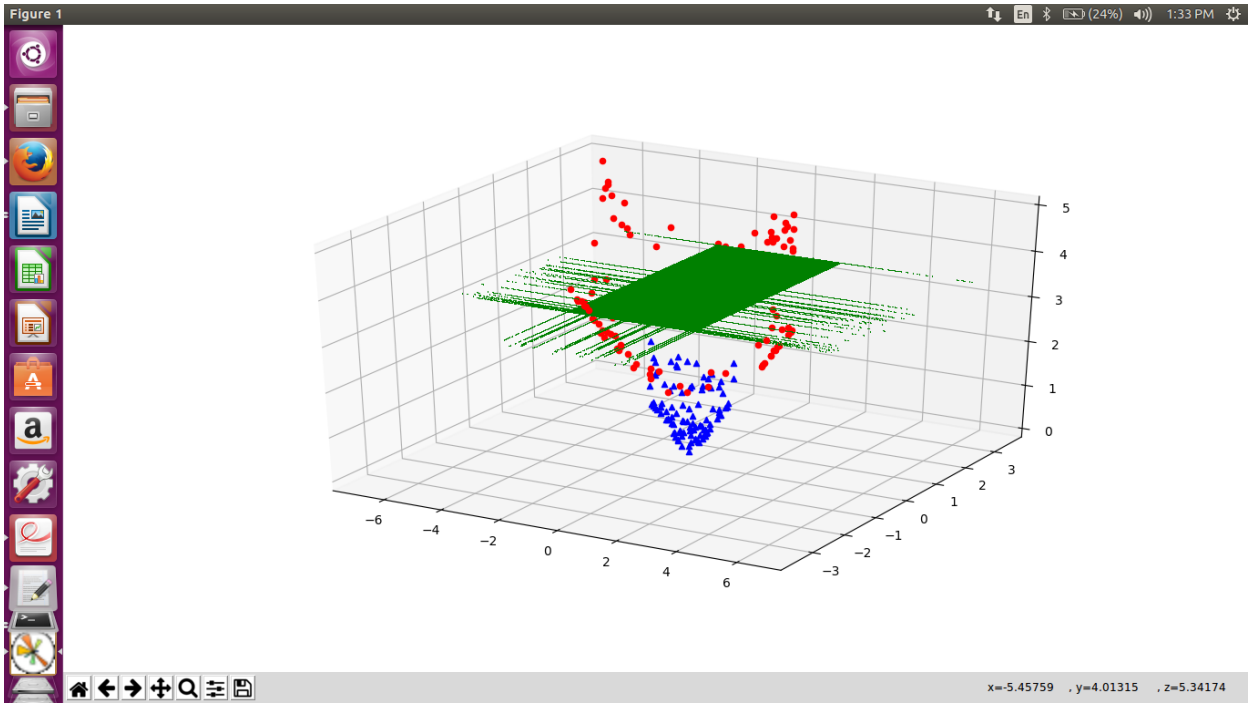
The red and blue points can be separated by a linear hyperplane which is why the data is said to be linearly separable. The hyperplane separating the two points is shown.



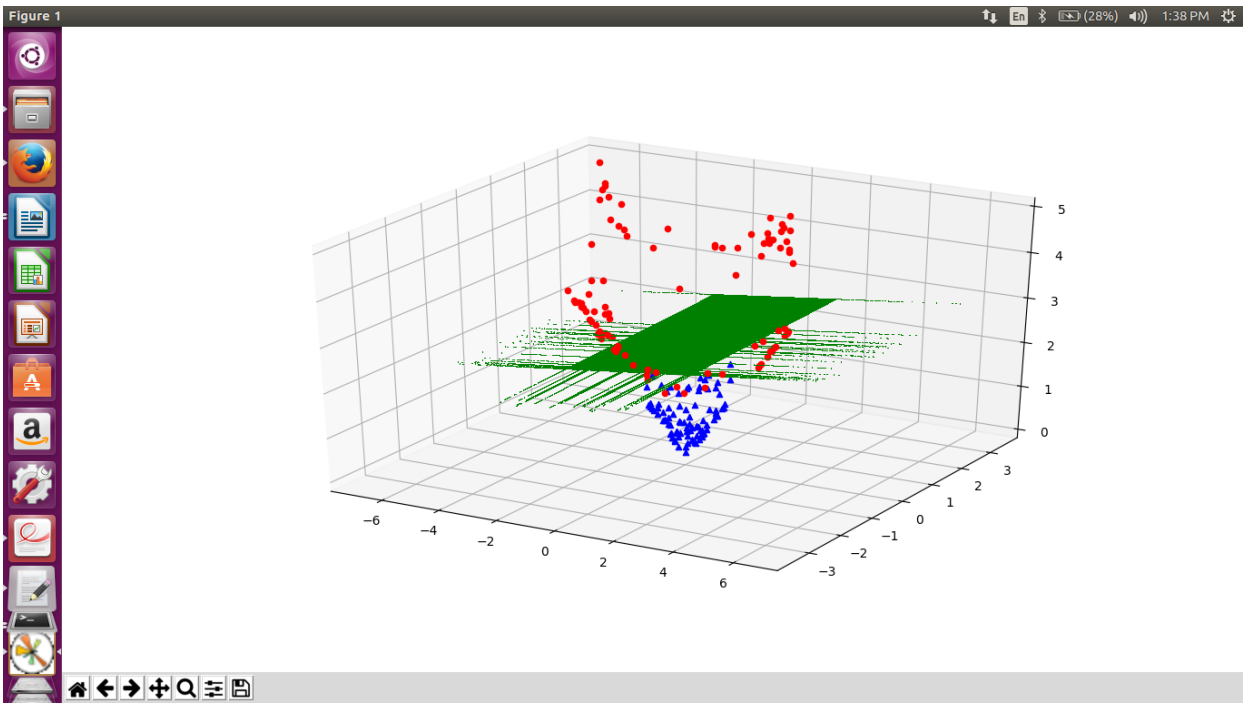
For L1 penalty, the data is linearly separable as long as alpha is below 0.07. For 0.07, the picture is shown below.



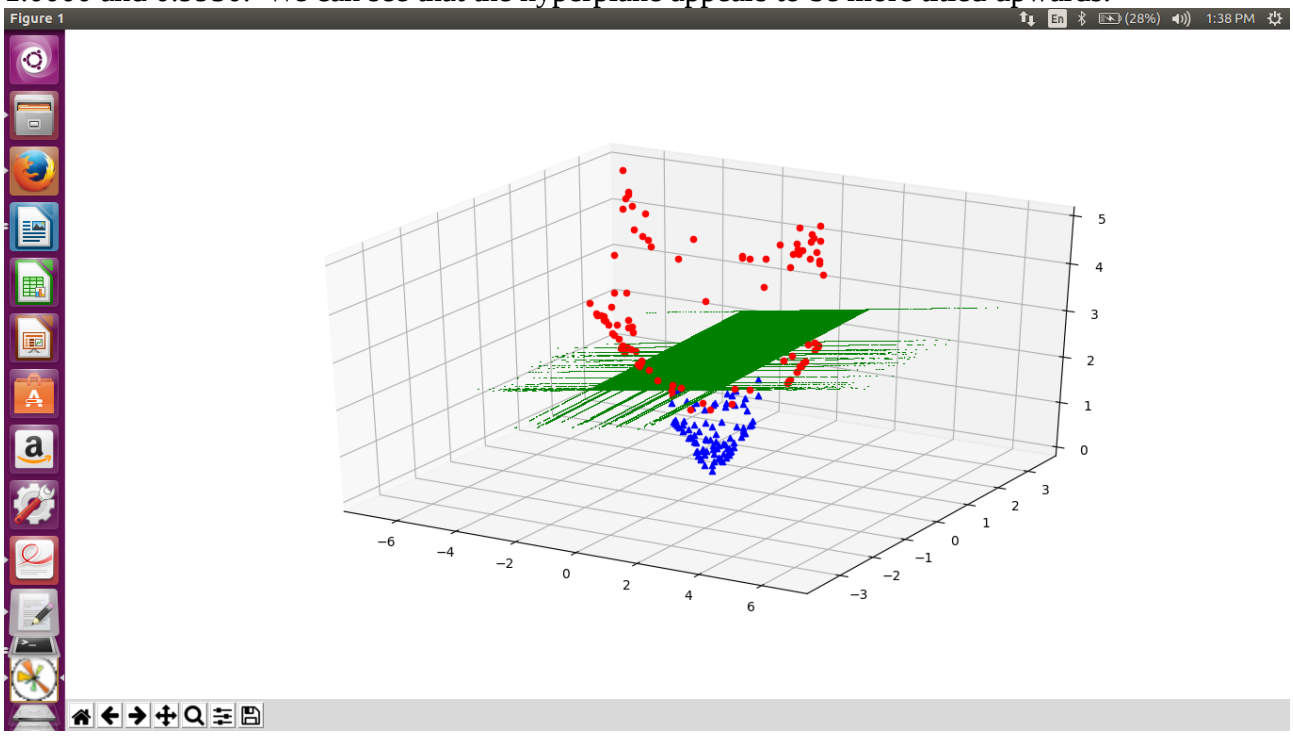
For alpha = 0.08, the below picture is shown. We can see that the hyperplane rises which causes it to misclassify. Note that for alpha = 0.08, the performance metrics are 0.8250, 0.6500, 1.0000 and 0.7879.



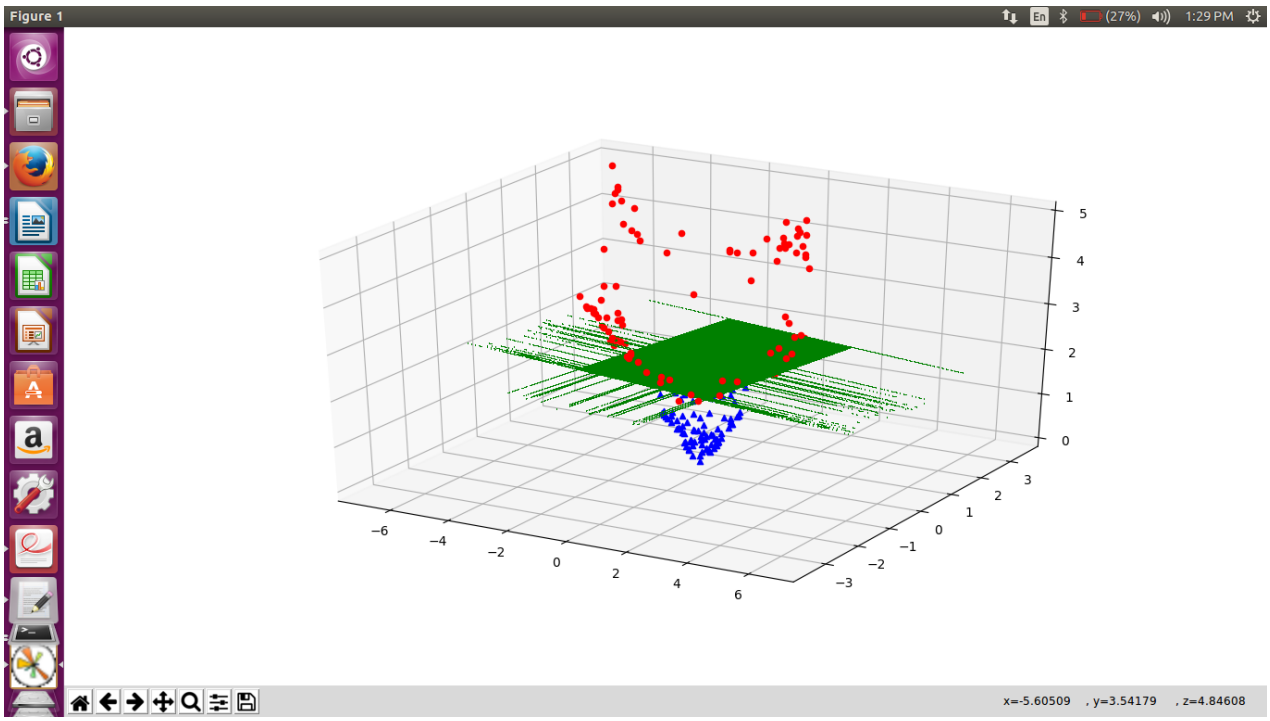
For the L2 penalty, the data continues to be linearly separable as long as alpha is less than 0.00005. For alpha = 0.00004, the picture is shown below. The performance metrics are all 1.000.



For alpha = 0.00005, the picture, is shown below. The metrics become 0.9950, 0.9900, 1.0000 and 0.9950. We can see that the hyperplane appears to be more titled upwards.



For no penalty, the data always happens to be linearly separable. The picture is shown below.



We can see that out of all the kernels I have used, the second one seems to be better at mapping the points as the red points and blue points seem to be separated from one another by a large margin. Since a large margin is desired in order to increase performance, I have included the second kernel in the submission file.

b) Letter Classification

Case 1: Linear kernel

On using a linear kernel with C (penalty parameter) = 1 and max_iter = -1, the accuracy, precision, recall and f1 score for the data is 0.8573999999999994, 0.8560267578621209, 0.8588460114569673 7 and 0.8560334614444487, respectively.

On increasing C to 1.5 while keeping the other parameters constant, the metrics become 0.8568000000000012, 0.8551290026361702, 0.8579200781602388 5 and 0.8551173994333932 7. We can see that all the metrics have decreased slightly on increasing C.

On reducing C to 0.5 and other parameters constant, the metrics become 0.8507000000000001, 0.8491696501295653 6, 0.8526121856410572 3 and 0.8495749109239174 6. We can see that decreasing C has a worse effect when compared to increasing C.

On keeping C to 0.75 and other parameters constant, the metrics become 0.8579000000000011, 0.8566195579248056, 0.8606039418874539 3 and 0.8570543938430276 2. We can see that this performance is better than the one for C = 1.5 but is still inferior to that for C = 1.

On keeping C to 1.1, the metrics become 0.8560999999999997, 0.8546277250519299 7, 0.858100820693782 8 and 0.854902035486476 7.

Best case:

On keeping C to 1.07, the metrics become 0.8574999999999993, 0.8561913241275768 1, 0.8593449416603899 9 and 0.8567634651746057 8.

Observations:

I noticed that on increasing C slightly better performance was observed. I also noticed that in general a larger C gives a better performance than a smaller C when those two are equidistant from 1 (such as C = 1.5 is superior to C =0.5).

Case 2: RBF

On using a rbf kernel with C = 1 and gamma = ‘auto’, the accuracy, precision, recall and f1 scores were 0.94009999999999994, 0.9396806240735931, 0.9426517009073393 and 0.94030679867755629, respectively.

On increasing C to 1.2 while keeping the other parameters same, the metrics become 0.94889999999999985, 0.94850866928665734, 0.95057219601217857 and 0.94888007357546322. We can see that thee is an improvement in the performance.

On further increasing C to 1.5, the performance is improved further as seen by the metrics 0.9516, 0.95114677899984501, 0.95315784733040443 and 0.95150653101877913.

For C = 10, the metrics become 0.97059999999999991, 0.97030379923808585, 0.97094243715466244 and 0.97039925721528086.

For C = 13, the performance reaches its peak with the metrics being 0.97110000000000007, 0.97086389637002646, 0.97171207521222647 and 0.97099302993176251.

On keeping C = 13 and making gamma = 0.1 (other parameters are constant), the metrics show an increase by becoming 0.9766999999999999, 0.97648986742389587, 0.97706369362842993 and 0.97659770762015918.

On making gamma = 0.01 (other parameters are constant), the metrics reduce to 0.9265000000000001, 0.92594118354904364, 0.92881940715761746 and 0.92640783788381265.

On increasing gamma to 0.5, the metrics again experience a drop (although not that drastic) and become 0.96469999999999989, 0.96446999496887253, 0.96539164206879469 and 0.96466270891428285.

On decreasing gamma to 0.11, the metrics become 0.97349999999999992, 0.97333829513077885, 0.97383331987446853 and 0.97340393505861744 showing that the peak occurs around gamma = 0.1.

Best Case:

The best case is obtained at C = 13 and gamma = 0.1, with values 0.9766999999999999, 0.97648986742389587, 0.97706369362842993 and 0.97659770762015918.

Observations:

I have observed that as C increases upto 13, the performance continues to rise after which it falls. Also, when we set gamma around 0.1, the performance rises when compared to that when gamma is ‘auto’.

Case 3: Sigmoid Kernel

On using a sigmoid kernel with C = 1, gamma = ‘auto’ and coef0 = 0, the metrics become 0.47490000000000004, 0.47299132613418804, 0.50747534464251187 and 0.47717310916056316.

On reducing C to 0.1 and keeping other parameters constant, the metrics become 0.6026999999999999, 0.60045039870961858, 0.61258243480105457 and 0.59925003525510745. We can see that performance has increased.

On further reducing C to 0.05, the metrics become 0.64249999999999996, 0.63996719191866624, 0.64906763143178781 and 0.63562436865009297.

On increasing C to 1.2, the metrics become 0.47119999999999995, 0.46978962546171193, 0.50377568423524188 and 0.47304105108624722. We can see that all things remaining constant, if C is reduced, the performance becomes better.

On using C = 0.05 and gamma = 0.1, the metrics become .0.48570000000000002, 0.48362427551589759, 0.50307505775510108, and 0.48706970591335796 We can see that the performance has worsened.

On keeping gamma = 0.01, the metrics become 0.52280000000000004, 0.51743893972254018, 0.60711643684471939 and 0.49944928134152866. We can see that on decreasing gamma, the performance increases.

On reducing gamma to 0.007, the metrics decrease to 0.41529999999999995, 0.40924567698735786, 0.47903860115011143 and 0.38792656653565744.

On increasing gamma to 0.5, the metrics reduce to 0.1086, 0.10853440122036787, 0.15907011099396104 and 0.11080977549031301.

We can see that using gamma only has been reducing our performance.

On using C = 0.05, gamma = ‘auto’ and coef0 = 0.01, the metrics become 0.62890000000000001, 0.62662519104856984, 0.63755583432361651 and 0.62299965323683892. We can see a slight decrease.

On increasing coef0 to 0.1, we get 0.61699999999999999, 0.61466244876339127, 0.62399262352574658 and 0.61041129596597055. We can see a decrease.

On decreasing coef0 to 0.0001, the metrics improve to give 0.64329999999999998, 0.6407429540700138, 0.64929439516417176 and 0.63620906018069279.

On decreasing coef0 to 0.000001, the metrics reach their peak to become 0.64490000000000003, 0.64239920228641578, 0.65311737750333188 and 0.63821269333805808.

Best Case:

For C = 0.05, gamma = ‘auto’ and coef0 = 0.000001 with values 0.64490000000000003, 0.64239920228641578, 0.65311737750333188 and 0.63821269333805808.

Observations:

We can see that as C reduces (uptil 0.05), the performance increases. We can see that on using gamma, the performance decreases. Also, for smaller positive values of coef0, the performance improves slightly.

On checking which two features are most discriminative:

On using the SelectKBest function with the method being f_classif, we get that the two most discriminative features are the 7^{th} and 11^{th} ones. On evaluating, I found out that these are ybar and x2ybar.

I had also used just two features at a time in order to see which two features worked the best.

On using the linear kernel with C = 1.07 and the first two features of X_data (hpos and vpos), the metrics are 0.86069999999999991, 0.85930920583485304, 0.86256459302854382 and 0.85945980066591121.

On using the same kernel but with the second and third features of X_data (vpos and w), the metrics become 0.85319999999999996, 0.85160918816777653, 0.85520316989536593 and 0.85176825347934704.

On using the third and fourth features of X_data (w and h), the metrics become 0.85239999999999994, 0.85090945601760326, 0.85388342380120652 and 0.85100735435591202.

On using the fourth and fifth features of X_data (h,pix) , the metrics become 0.85250000000000004, 0.85103103845298578, 0.85445665594489983 and 0.85140284151052048.

On using the fifth and sixth features (pix, xbar), the metrics become 0.85299999999999998, 0.85179448533937341, 0.85562259929813089 and 0.85218953143296938.

On using the sixth and seventh features (xbar, ybar), the metrics become 0.85720000000000007, 0.85555196152753266, 0.85899448041241155 and 0.85587987770256846.

On using the seventh and eighth features (ybar, x2bar), the metrics become 0.86039999999999994, 0.85899532347309182, 0.86188832024821971 and 0.85927503511955816.

On using the eighth and ninth features (x2bar, y2bar), the metrics become 0.85039999999999993, 0.84870114767641469, 0.85209123915069573 and 0.84911827569798748.

On using the ninth and tenth features (y2bar, xybar), the metrics become 0.8538, 0.85229885861598353, 0.85568611649150061 and 0.85257822041839904.

On using the tenth and eleventh features (xybar, x2ybar), the metrics become 0.85619999999999996, 0.85475238345007987, 0.85694102798051675 and 0.85472048893181773.

On using the eleventh and twelveth features (x2ybr,xy2br), the metrics become 0.85120000000000007, 0.8497130309077543, 0.85281053157994768 and 0.84972671569698621.

On using the twelvth and thirteenth features (xy2br, xedge), the metrics become 0.84589999999999999, 0.8442744527440359, 0.84768479116285567 and 0.84456024628489978.

On using the thirteenth and fourteenth features (xedge, xegvy), the metrics become 0.85640000000000005, 0.85521115171954298, 0.85818928970533859 and 0.85552763312281199.

On using the fourteenth and fifteenth features (xegvy, yegde), the metrics become 0.85329999999999995, 0.85203373244763758, 0.85677884694501283 and 0.85268946876818619.

On using the last two features (yedge, yegvx), the metrics become 0.85320000000000018, 0.85176583497628489, 0.8552651880881541 and 0.85209422140893787.

We can see that among these combinations, the best two features to use would be the first two ones which are hpos and vpos.