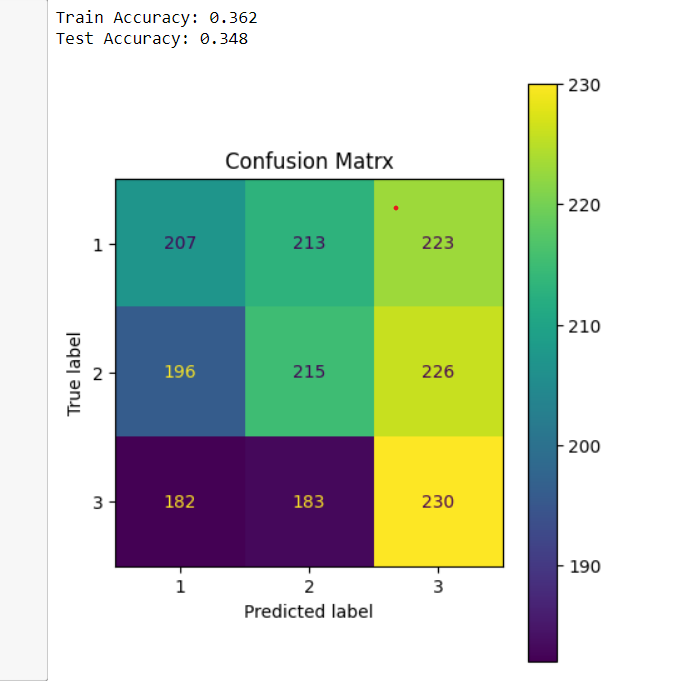
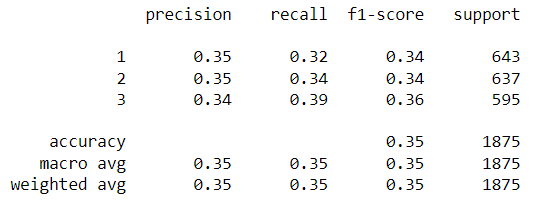
### **Task 0: Naïve Logistic Regression**

Data provided has been classifed into more than 2 classes. Multinomial Logistic Regression is used to classify data which belongs to more than two classes. The loss function will be cross entropy for such scenario. Task 0 will evaluate the original data. A error matrix is visualized alogn with accuracy values.



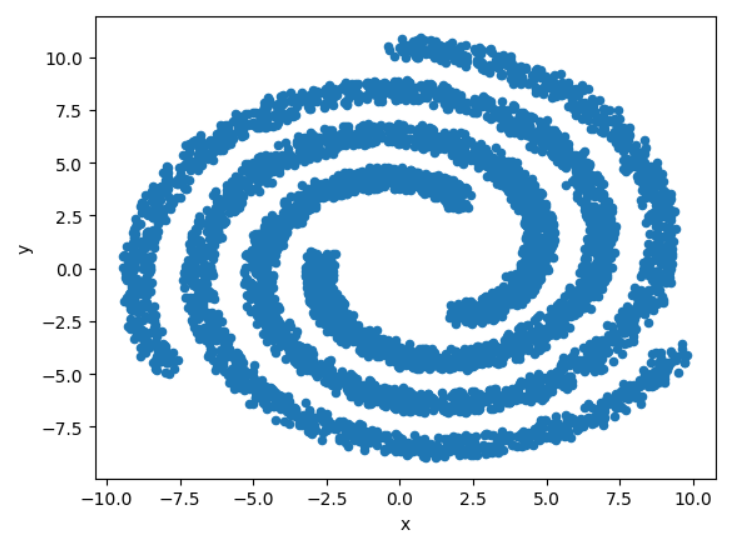
Classification Report:



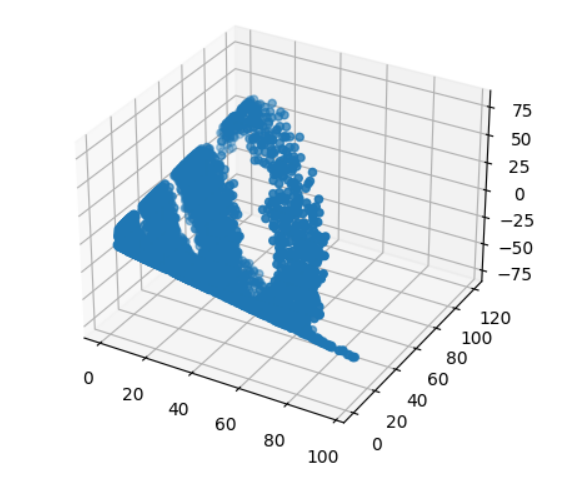
# **Task 1 : Train Data Transformation**

We transform the orginal data into feature space using polynomial Kernel. Kernels transform the non-linea data into linear data. This could also be done using PCA. We need to divide the sprial data set into 4 quadrents to make the data linear. It is a hectic process. So we go with kernel. Here, **we use degree 2 polynomial kernel (Kernel 1).**

## **Visualize the original data**



Visualization of Data after transformation.



## **4 Assumptions of logistic regression**

To be able apply linear models, the data should obey few assumptions. These assumptions were verified below for train data that was transformed into new space.

1. The outcome should be either binomial or multinomial classes

Unique classes : 3

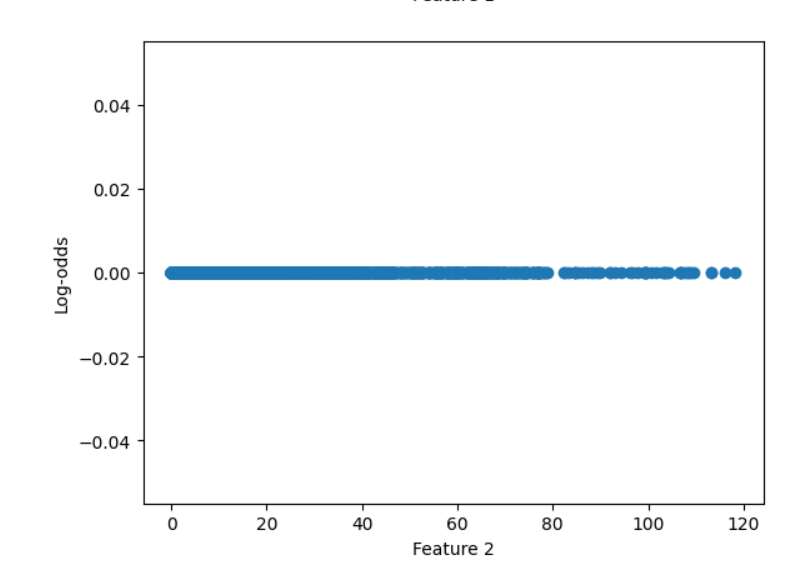
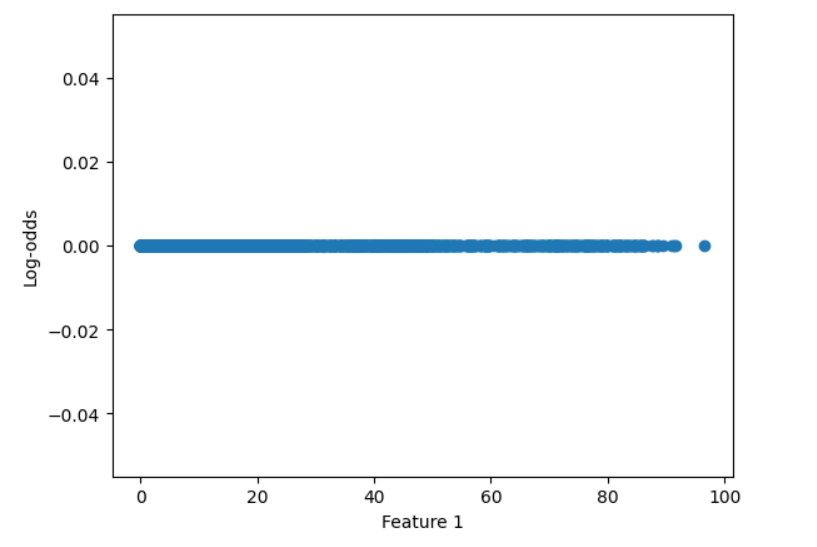
1. The length of the dataset should exceed 500

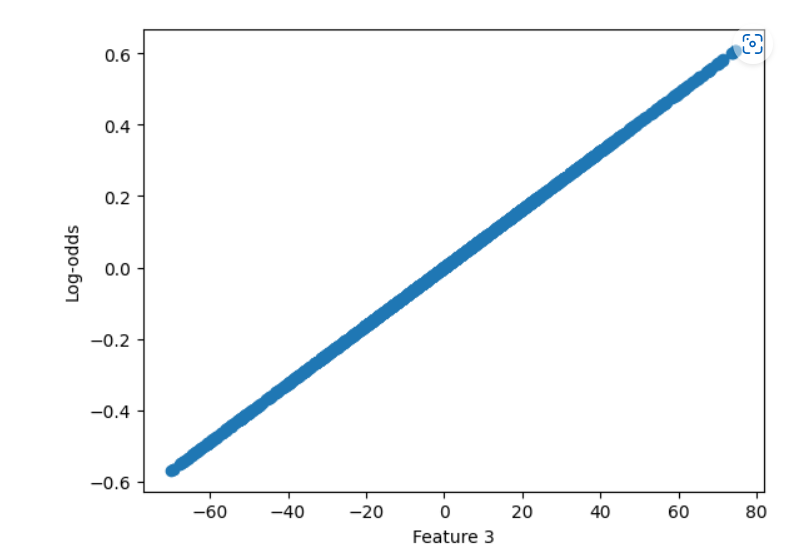
Length of Dataset : 5625

1. Absence of multicolinearity. This can be verified using variance inflation factor.

Number of features having problematic amount of multicolinearity : 0

1. Linearity of independent variables. We use log odds to verfiy this assumption.



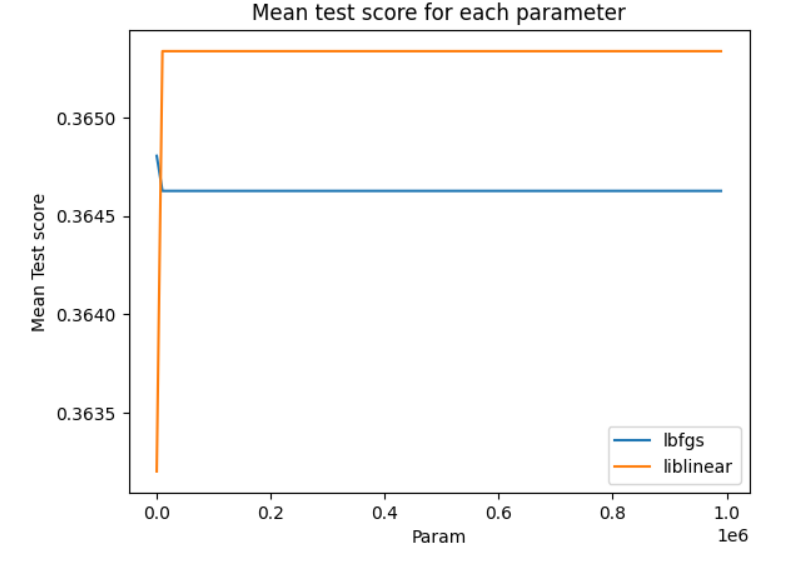


# **Task 2 : Linear Parametric Classification**

We use scikit-learn logistic regression method to evaluate the degree 2 polynomial kernel transfromed data.We find it coefficients and then we use gridSearchCV to tune the hyperparameters. We will take 100 different coefficents ranging between 10e-5 to 10e-5. We will also evaulate on limited memory BFGS, liblinear solvers and observe there performences. We pick the best mobel after hyper tuning, fit the data with it and evaluate using test data.

Weights of the untuned model:  
[[-0.00422421 0.00193641 -0.00610417]  
 [ 0.00077619 0.00222582 0.00709119]  
 [ 0.00344802 -0.00416222 -0.00098702]]  
  
Test data accuracy of the untuned model: 0.3589333333333333  
  
Train data accuracy of best model from grid search cv: 0.365

Test data evaluation of best fit model from grid search cv: 0.359



# **Task 3 : Transformation using Kernel Method**

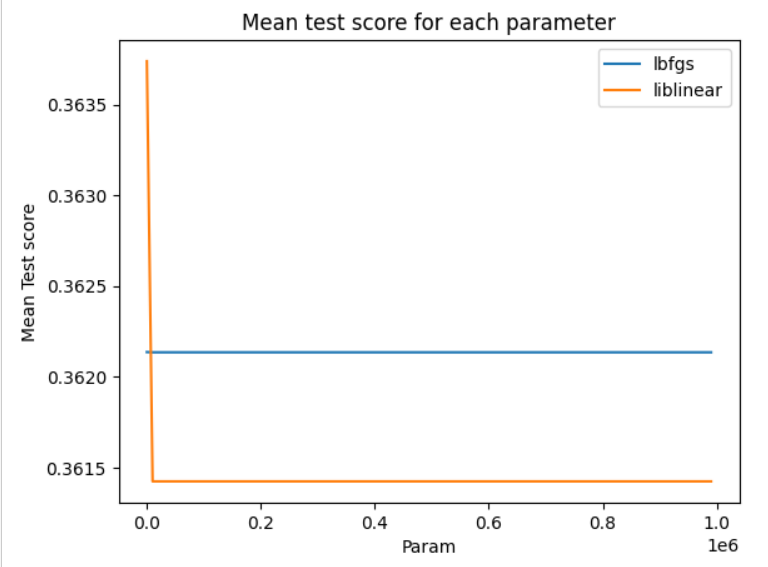
We have used one kernel already to make data linear. We will implement 4 other kernels and evaulate the respective transformed data using the Task 2 code i.e we will evaluate the data with and without hypertuning on logistic regression parameters. We will pick the best model in the collection of hypertuned models and evalute it with test data.

## **Kernel 2 : Linear Kernel**

We will try exponenting the very basic linear kernel. The mathamatial equation of this kernel is k(x,y) = (x,y)

Weights of the untuned model:  
[[-0.0086941 -0.046578 ]  
 [ 0.04352109 0.01751435]  
 [-0.03482698 0.02906365]]  
  
Test data accuracy of the untuned model: 0.34773333333333334  
  
Train data accuracy of best model from grid search cv: 0.364

Test data evaluation with best fit model give accuracy 0.344

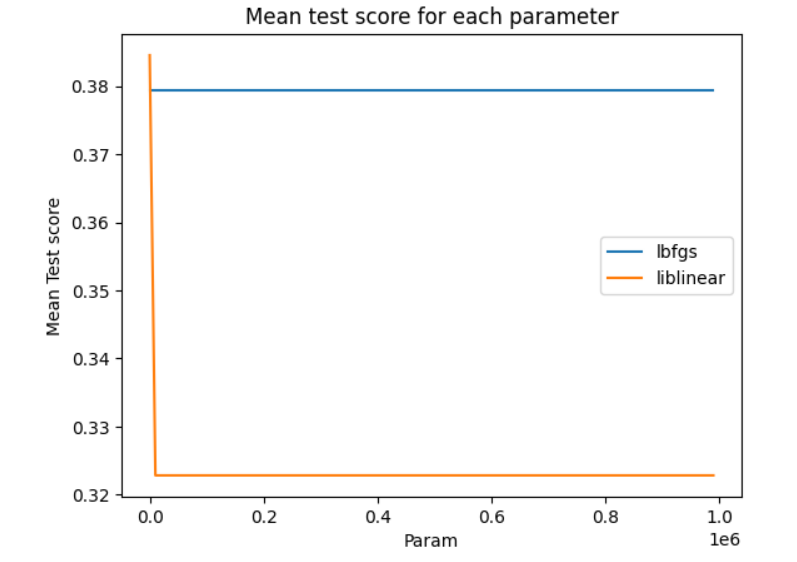


## **Kernel 3 : 4 degree polynomial kernel**

We will use 4 degree polynomial kernel. The mathamatical equation of it is k(x,y) = (x^4, 2x^3y, sqrt(3)x^2y^2, 2xy^3, y^4).

Weights of the untuned model:  
[[-1.53326590e-04 -8.61074388e-05 1.18128290e-04 -9.58456581e-05  
 -3.46852341e-06]  
 [ 5.84036820e-05 3.53787719e-05 -6.31327311e-05 1.91887289e-04  
 7.86741788e-06]  
 [ 9.49229084e-05 5.07286669e-05 -5.49955590e-05 -9.60416308e-05  
 -4.39889446e-06]]  
  
Test data accuracy of the untuned model: 0.3808  
  
Train data accuracy of best model from grid search cv: 0.385

Test data evaluation with best fit model give accuracy 0.384

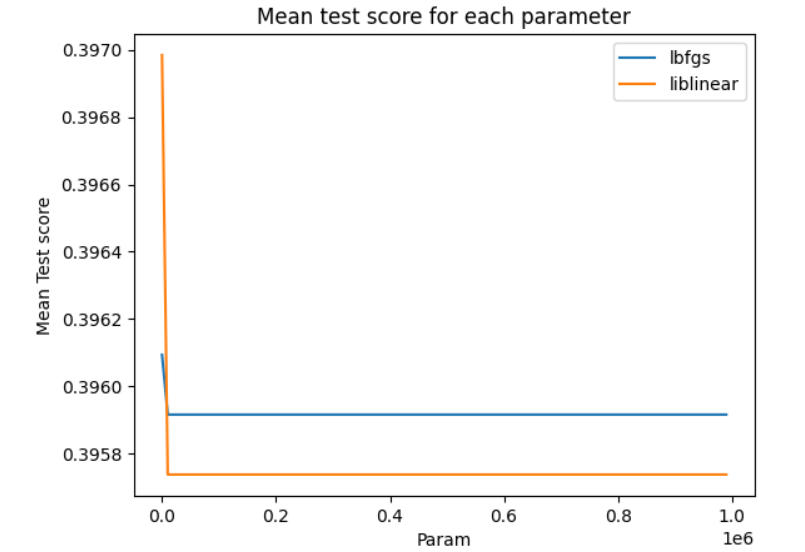


## **Kenrel 4 : Hyperbolic Tangent Kernel Function**[**¶**](http://localhost:8888/notebooks/ML_Project1.ipynb#Kenrel-4-:-Hyperbolic-Tangent-Kernel-Function)

This kernel is generally used in neural networks. Yet, this is a valid function. The mathematical function is k(x,y) = tanh(kxy+c) or k(x,y) = tanh((sqrt(k)x,sqrt(k)y))

Weights of the untuned model:  
[[ 0.05874668 -0.08273913]  
 [ 0.07524915 0.1232782 ]  
 [-0.13399582 -0.04053907]]  
  
Test data accuracy of the untuned model: 0.37546666666666667  
  
Train data accuracy of best model from grid search cv: 0.397

Test data evaluation with best fit model give accuracy 0.375



## **Kernel 5 : Guassan RBF Kernel**

Guassina RBF kernel assumes, that the data is normally distributed. The transformed data will also be in gassian or normally distributed.

Observation : Guassian kernel is taking very long time while executing gridsearchcv for 100 values

Weights of the untuned guassian model:  
[[ 2.14112082 -3.56260938 -0.07611596 -5.26402193 1.69754648 -0.20770251  
 -0.78169269 0.52293503 0.62714759 -4.10141584 2.5365963 -3.99866883  
 -0.09879497 3.48424394 0.28213444 -0.04900083 -1.3212572 0.20979561  
 -0.22701847 2.5993769 -0.48800217 0.12527617 -1.55638012 1.1987337  
 -0.33182206 -2.84457545 -2.81121316 0.2663857 -0.87791916 1.9043999  
 2.80728921 2.48253211 -0.73100713 0.56683876 -0.48369233 0.92737849  
 -0.90442559 0.78596976 -1.06824748 0.0455107 0.02390953 -0.43704832  
 0.33289179 0.96987753 -0.29535673 -0.1556 1.89747311 4.17723451  
 2.45756156 -1.40145931 -0.14822953 0.5655483 -0.50880838 -2.58402962  
 2.17190765 -0.38453188 -0.1062103 0.46159572 -1.01730944 4.48120226  
 1.08399128 2.45569316 0.78623244 0.30454151 -0.03847359 3.12047516  
 -0.86117346 -0.69170422 -0.24234666 0.56092399 1.3235071 0.59872155  
 -0.58852177 -2.45100044 1.82760405 0.8887377 -2.86720559 -0.36032119  
 -1.33751656 1.27536151 -0.36520782 -0.74894414 -1.63193587 1.5327079  
 -0.34612527 1.46155132 -1.15541347 3.97550052 0.07825006 -0.99463303  
 2.50635445 -0.9874835 1.50861402 2.2018459 1.79293059 1.85230213  
 2.76809801 -4.06793217 -1.2783229 -1.58890076]

[ 1.45934955 6.65984203 -0.01019066 2.97313215 2.584637 0.0978013  
 0.32562423 -0.67235637 1.81603106 -0.58201793 -3.3708671 -1.21588325  
 -2.21719226 0.3464408 -0.0406521 -0.1809373 -0.59869385 -0.32969304  
 0.44177087 -0.48067258 0.35294037 -1.47994686 -2.82526912 -0.48820829  
 0.62657945 1.09875474 2.68712993 -0.48101828 0.81100344 -6.64562852  
 0.43477873 -0.6019919 -3.98468939 -1.77154725 0.32859464 -0.117257  
 0.77016223 -0.98363773 -2.9474391 -0.20250817 0.30610566 -0.10843367  
 -0.87296016 1.73648627 0.38886237 0.38973506 -1.78560511 -2.1785142  
 -3.84754524 -0.85606424 0.18463599 -1.54457165 1.24653626 3.07387283  
 -1.28502427 -0.03276833 -0.40671448 -0.39194951 1.18766688 -4.41951004  
 -0.20382186 -2.4928903 2.3138433 0.51532142 -2.97804653 -4.80296892  
 2.00174861 1.27062382 -0.17864027 -1.06359337 -0.80143681 -0.40130602  
 0.3904597 3.39156871 -0.46214837 -0.47140232 -0.13445609 -0.07949099  
 0.24370544 2.53967529 0.34854531 -0.77474987 0.51274416 0.09385727  
 -0.19490696 0.85988194 -0.20659063 -8.85059868 0.35997852 -2.05161957  
 -2.92565865 0.29995752 -2.39068345 0.80979873 0.91430403 0.05521151  
 3.73587648 4.84331845 3.36336805 2.66843238]

[-3.60047037 -3.09723265 0.08630661 2.29088977 -4.28218348 0.10990121  
 0.45606846 0.14942134 -2.44317865 4.68343377 0.8342708 5.21455207  
 2.31598723 -3.83068474 -0.24148234 0.22993813 1.91995105 0.11989743  
 -0.2147524 -2.11870432 0.1350618 1.35467069 4.38164924 -0.71052541  
 -0.2947574 1.74582071 0.12408323 0.21463257 0.06691572 4.74122861  
 -3.24206794 -1.88054021 4.71569651 1.20470849 0.15509769 -0.81012149  
 0.13426336 0.19766797 4.01568658 0.15699747 -0.33001519 0.54548199  
 0.54006837 -2.7063638 -0.09350564 -0.23413506 -0.111868 -1.99872031  
 1.38998369 2.25752355 -0.03640645 0.97902335 -0.73772789 -0.48984321  
 -0.88688337 0.4173002 0.51292478 -0.06964622 -0.17035744 -0.06169221  
 -0.88016942 0.03719713 -3.10007574 -0.81986293 3.01652012 1.68249376  
 -1.14057515 -0.57891961 0.42098693 0.50266938 -0.52207029 -0.19741553  
 0.19806207 -0.94056827 -1.36545568 -0.41733538 3.00166168 0.43981218  
 1.09381112 -3.8150368 0.01666251 1.52369402 1.1191917 -1.62656518  
 0.54103224 -2.32143325 1.3620041 4.87509816 -0.43822858 3.0462526  
 0.4193042 0.68752598 0.88206942 -3.01164463 -2.70723463 -1.90751364  
 -6.50397449 -0.77538627 -2.08504515 -1.07953161]]

Test data accuracy of the untuned guassian model 0.944

# **Task 4: Non-parametric KNN Classification**

We will K Nearest Neighbours algorithm to classify the original and transformed datasets from different kernels. We will visualize performance of each dataset. Number of Neighbours will be between 1 to 200.

