In [133]:
```
#lets see performance in test -set for n_estimators = 5 ie lets look at optimize
model_gb_o = GradientBoostingClassifier(random_state = 42, n_estimators = 5)
model_gb_o.fit(X_train_gb, y_train_gb)
y_pred_test = model_gb_o.predict(X_test_gb)
score_gb = accuracy_score(y_test_gb, y_pred_test)
print("test set accuracy of Gradient boosting trees for for n_estimator = 5 is ",
```
```
test set accuracy of Gradient boosting trees for for n_estimator = 5 is  1.0
```

In [134]:
```
#lets see performance in test -set for keeping n_estimators as default ie lets l
model_gb_d = GradientBoostingClassifier(random_state = 42)
model_gb_d.fit(X_train_gb, y_train_gb)
y_pred_test = model_gb_d.predict(X_test_gb)
score_gb = accuracy_score(y_test_gb, y_pred_test)
print("Test set accuracy of Gradient boosting trees for n_estimator as deafult i
```
```
Test set accuracy of Gradient boosting trees for n_estimator as deafult is  1.0
```

Analysis

```
=================================================================================
```

Q : Why do we split dataset into training and test dataset ?

Ans : We split dataset into train and test set so that we can design our model based on some data, and can measure its performance on unseen data. The dataset used for designing our model is called training set and dataset used for testing the model is called test set. Since, we are interested in performace of our model for unforeseen /new data, test data should not be part of model designing phase. Also, if we do not split the data, we would not know if our algorithm is facing high variance (overfitting) or not, ie algorithm might overfit the given data and may not perform well on new data. Therefore we usually split the data into parts, one part for test set and another part for training set

```
=================================================================================
```

Q : Explain why when finding the best parameters for KNN you didn't evaluate directly on the test set and had to use a validation test.

Ans : Validation set is used for tuning the hyper-parameters so that model can be improved using the tuned parameter and thus can be deployed to perform on unseen data. If we would have used test set for finding the best parameters, we would not have a way to know how the model would perform or measure its performance on unknown data. Also by doing, the model might have overfitted the test data here, and might result in lower accuracy performance on new/unseen data.

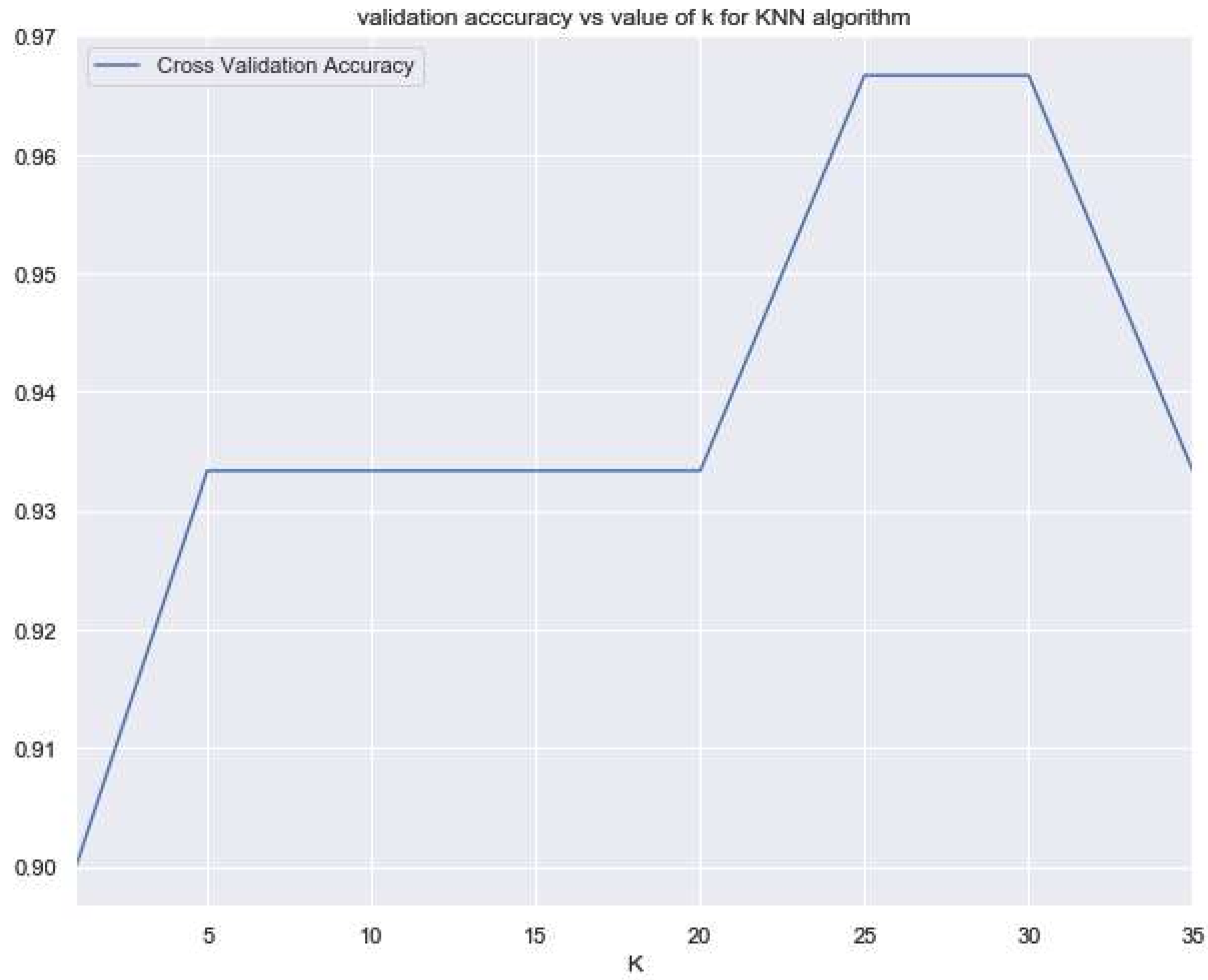
Thus validation set is used to ensure that we still have a dataset (i.e. test dataset) to measure the performance of model on new data(other than training data and validation data) and parameters that makes algorithm perform well on validation dataset would be used as final parameters for the model, and then this model would be used to predict performance on test data (unseen/unknown data).

===

Q :What was the effect of changing k for KNN. Was the accuracy always affected the same way with an increase of k? Why do you think this happened?

Ans : With increase in values of K the accuracy first increased and then decreased, lets see the analysis below

lets look at the validation set performance vs value of K graph and table storing the Cross validation accuracies for various K values



In [69]:
```
K_val_data
```

Out[69]:

	K	Cross Validation Accuracy
0	1	0.900000
1	5	0.933333
2	10	0.933333
3	15	0.933333
4	20	0.933333
5	25	0.966667
6	30	0.966667
7	35	0.933333

With increase in value of K the Cross Validation accuracy intially improved as K increased (using manhy neigbhors instead of few neighbors would be better to classify the data) ,till K increased to 25 to 30 , as Value of K >= 30 , the cross validation accuracy started to decrease with increase in value of K.

From the Graph and data, it can be seen, the performance accuracy on validation is highest for k = 25 and 30,also following observations and explainations can be made :

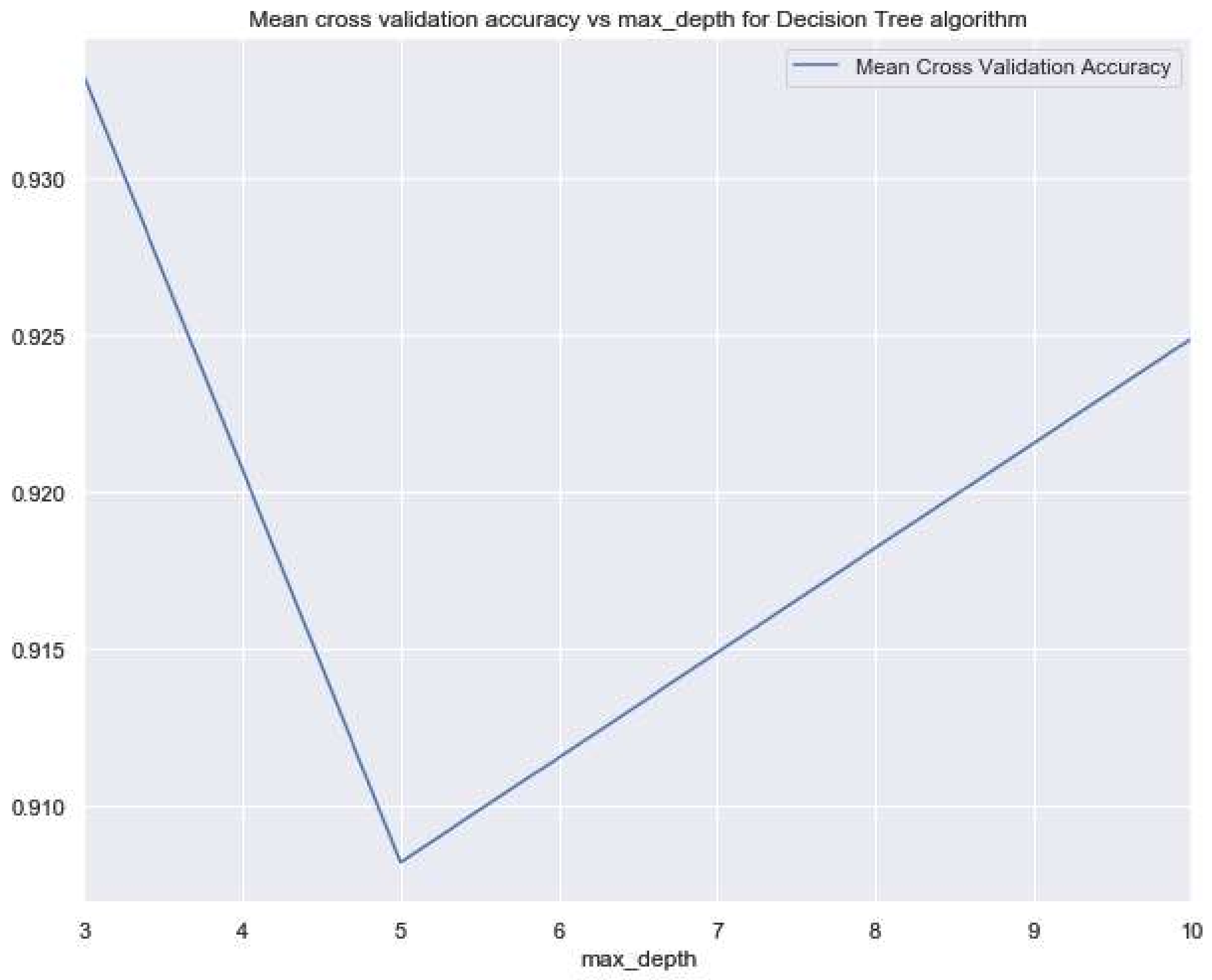
1. When K is small, the algorithm is looking only for few neighbors around and hence is performing mediore which can be attributed to overfitting.
2. From the graph, we can see KNN algorithm perfromance on validation set increases as value of K increases upto a certain value of K.(these values of K seem optimized values)
3. After A certain value of K, increase in K did not result in increase in validation accuracy ,(here K =30). It seems like model is having difficulty diffrentiating among classes, cause of having too many neighbours to decide from, thus providing an effect similar to under-fitting.

==

Q : What was the relative effect of changing the max depths for decision tree and random forests? Explain the reason for this.

Ans : The best performance for decision tree and random forest models is achieved with max_depth = 3, i.e., as we increase the max_depth in the Decision Tree and Random_forests models their performance on 10-fold validation set started to decrease.

Lets look at Decision tree mean validation accuracy vs max_depth plot and the corresponding data



In [70]:
```
DT_df
```

Out[70]:

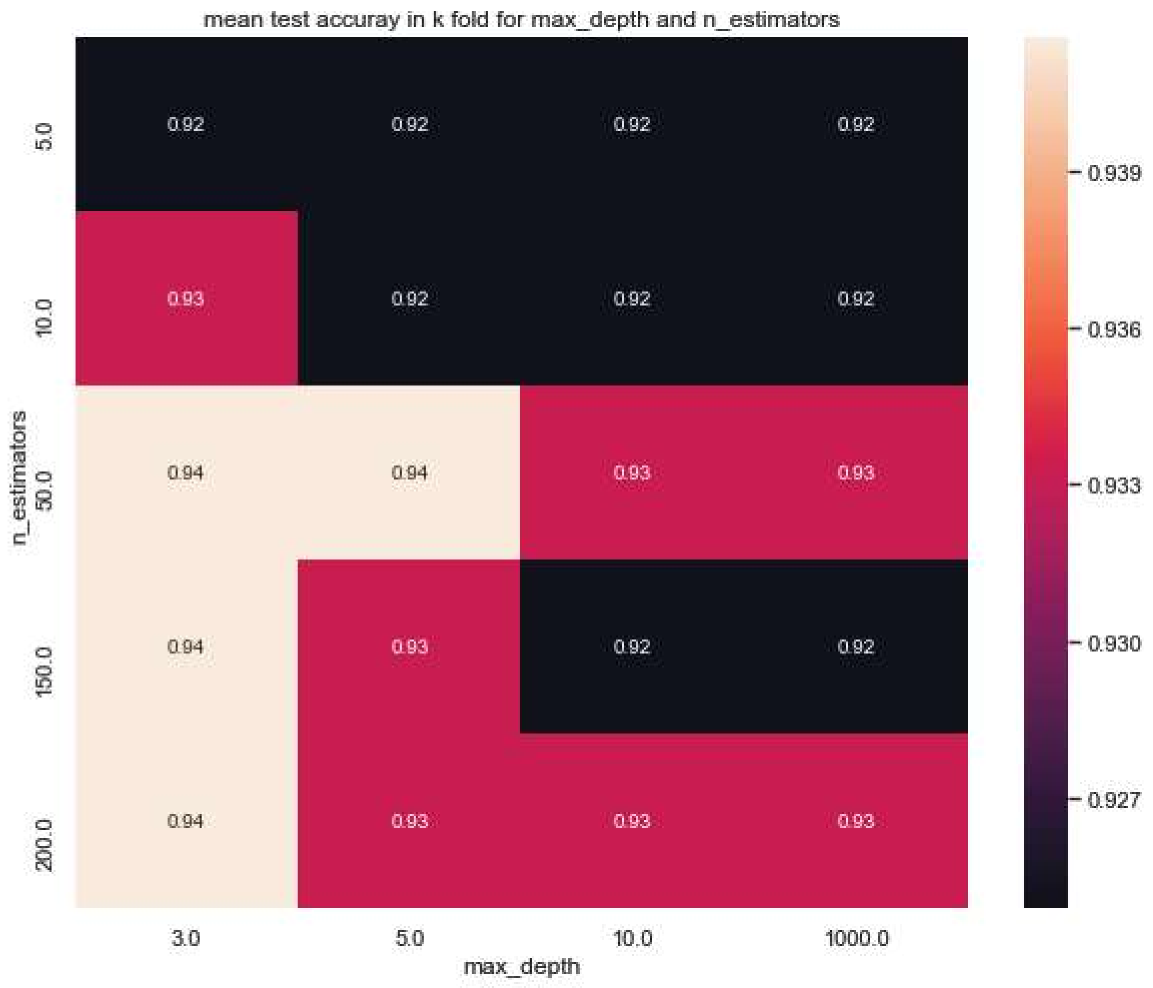
	max_depth	Mean Cross Validation Accuracy
0	3.0	0.933217
1	5.0	0.908217
2	10.0	0.924883
3	NaN	0.924883

- By looking at the plot (just above) and data for decision tree model we can see, as max_depth increases the mean accuracy in the cross validdation folds decreases/ gets constant.
- In plot, the accuarcy first decreases as max_depth values varies from 3 to 5 and then increases as max_depth values varies from 5 to 10 (or more). I think this is cause : **the maximum depth required to completely separate the training data into three clases is somewhere between 5 and 10.**
- I believe the more depth we allow, more it overfits the data. Also, for this dataset after a certain depth in decision tree model, probably max_depth <10, the model depth is not actually growing thus we are achieving same accuracy for max_depth =10 and max_depth = None

Also, getting 10-fold mean accuracy better on max_depth = 5 than on mean accuracy at max_depth = 10 is somewhat unexpected, I believe it has something to do with dataset, and how data being split at training validation fold splits and test set for the decison

Tree(algorithm). I think the target 1.0 and 2.0 are very close to each other, as can be seen through pairplots plotted above, which is making it difficult for decision trees to accurately seggregate the classes based on the data provided on training-validation split.

Lets look at Random forest mean validation accuracy vs max_depth plot heatplot and corresponding data



In [71]:
```
RF_df
```

Out[71]:

	max_depth	n_estimators	mean test accuray in k fold	approx time taken
0	3.0	5.0	0.924883	0.049866
1	3.0	10.0	0.933217	0.074770
2	3.0	50.0	0.941550	0.281278
3	3.0	150.0	0.941550	0.809256
4	3.0	200.0	0.941550	1.023935
5	5.0	5.0	0.924883	0.050869
6	5.0	10.0	0.924883	0.077796
7	5.0	50.0	0.941550	0.294118
8	5.0	150.0	0.933217	0.845376
9	5.0	200.0	0.933217	1.034411
10	10.0	5.0	0.924883	0.050863
11	10.0	10.0	0.924883	0.079264
12	10.0	50.0	0.933217	0.320659
13	10.0	150.0	0.924883	0.859764
14	10.0	200.0	0.933217	1.092116
15	1000.0	5.0	0.924883	0.052859
16	1000.0	10.0	0.924883	0.078822
17	1000.0	50.0	0.933217	0.275232
18	1000.0	150.0	0.924883	0.917716
19	1000.0	200.0	0.933217	1.116195

here n_estimators = None is plotted as n_estimators = 1000

- From looking at Heatmap for Random_forest, We can see max_depth is increased while keeping the n_estimators i.e. number of trees fixed, we can observe the performance of random-forest in mean-10 fold validation accuracy decreases as we increase the max_depth.
- I belive, this is again because of over-fitting being reciprocated for increased max-depths in all of the random-forest trees made with the different size of n_estimators.

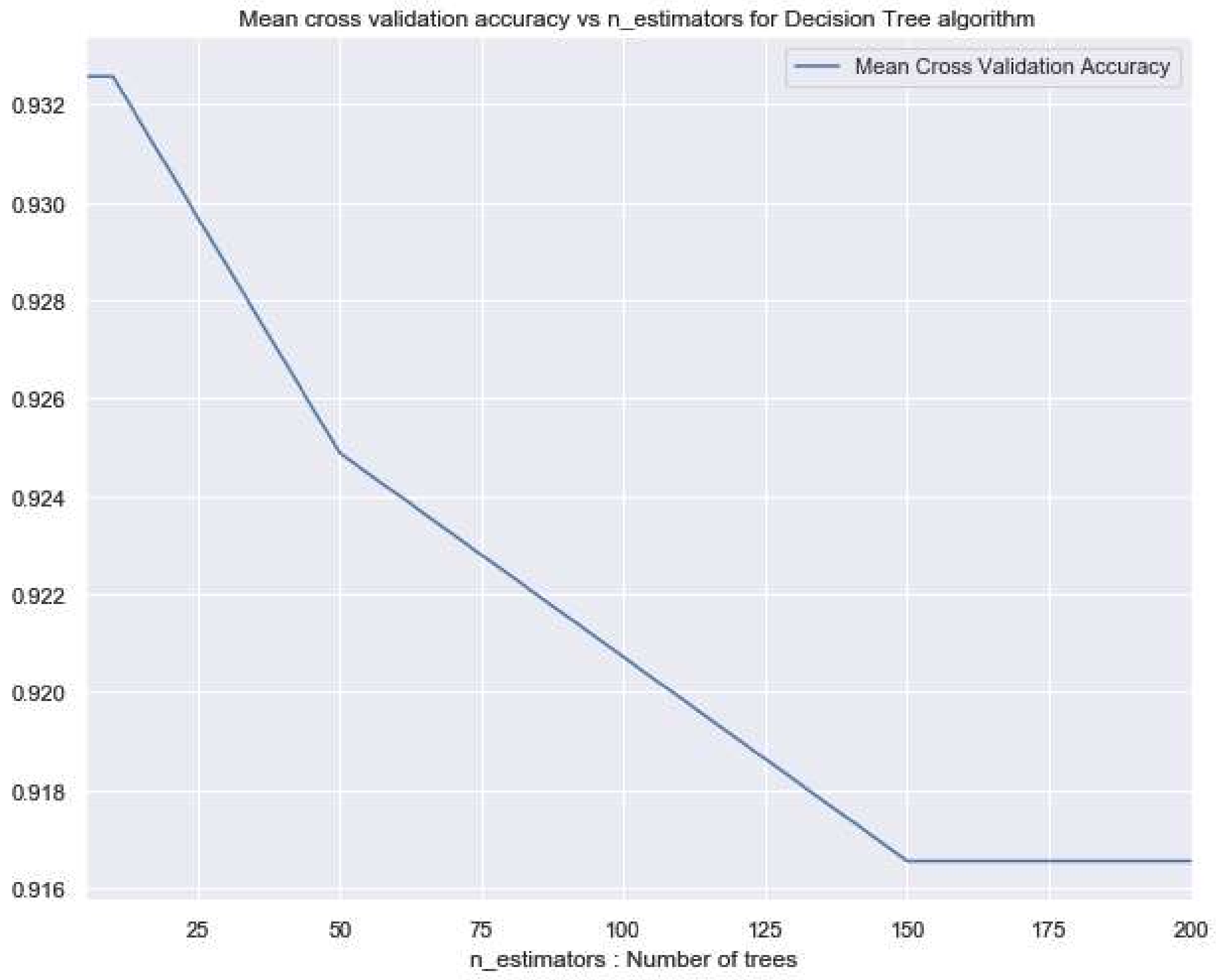
Thus we can conclude, increasing the max_depth might result in overfitting for decision trees and random forests, and should be done iterativey on validation set to tune it for getting a good performance on unseen data(test set).

==

Q : Comment on the effect of the number of estimators for Gradient TreeBoosting and what was the relative effect performance of gradient boosting compared with random forest. Explain the reason for this.

Ans: The performance for Random Forest and gradient boosting trees in mean 10-fold validation set usually first increases as we increase the number of trees in the model, and then after a certain number of increase in value of n_estimators (number of trees)as we increase the number of trees, the performance starts to decrease.

Lets look at Gradient Boosting tree mean validation accuracy vs n_estimators plot



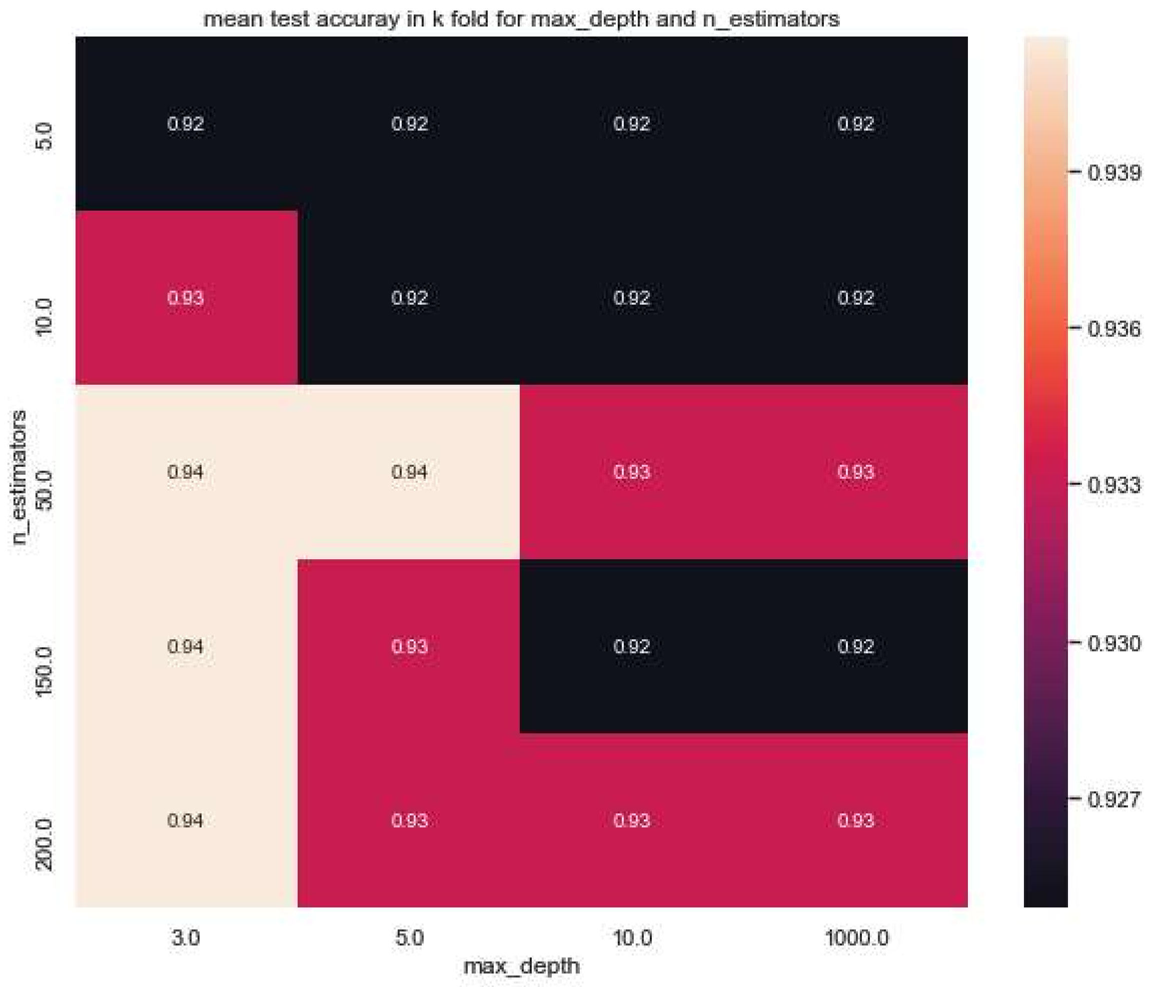
```
In [72]: GB_df
```

Out[72]:

	n_estimators : Number of trees	Mean Cross Validation Accuracy
0	5	0.932576
1	10	0.932576
2	50	0.924883
3	150	0.916550
4	200	0.916550

- By looking on the above plot , we can see the performance on Gradient boosting on validation set starts to decrease as as we increase the number of trees (n_estimators) in the model.
- I believe it is cause, the model with more number of trees fits the training data well, but can result in overfitting, resulting in poor validation accuracy.
- optimal performace is achieved for values of n_estimatiors = 5.
- Also, more number of trees would require larger training time resulting in slow prediction and fit time for this machine learning algorithm.

lets look at Random forest mean validation accuracy vs n_estimators plot heatplot and corresponding data



here n_estimators = None is plotted as n_estimators = 1000

In [73]:
```
RF_df
```

Out[73]:

	max_depth	n_estimators	mean test accuray in k fold	approx time taken
0	3.0	5.0	0.924883	0.049866
1	3.0	10.0	0.933217	0.074770
2	3.0	50.0	0.941550	0.281278
3	3.0	150.0	0.941550	0.809256
4	3.0	200.0	0.941550	1.023935
5	5.0	5.0	0.924883	0.050869
6	5.0	10.0	0.924883	0.077796
7	5.0	50.0	0.941550	0.294118
8	5.0	150.0	0.933217	0.845376
9	5.0	200.0	0.933217	1.034411
10	10.0	5.0	0.924883	0.050863
11	10.0	10.0	0.924883	0.079264
12	10.0	50.0	0.933217	0.320659
13	10.0	150.0	0.924883	0.859764
14	10.0	200.0	0.933217	1.092116
15	1000.0	5.0	0.924883	0.052859
16	1000.0	10.0	0.924883	0.078822
17	1000.0	50.0	0.933217	0.275232
18	1000.0	150.0	0.924883	0.917716
19	1000.0	200.0	0.933217	1.116195

Analysing above heatmap and data(for random forest) we can conclude following points :

- The best accuracy (0.941550) is achieved with max_depth = 3, with n_estimators = 50,150,200 and max_depth = 4, n_estimators = 50
- For fixed number of max_depth (i.e. depth of the tree) as number of trees increases, the approx time taken by algorithm increases.
- In most of the cases,for a fixed number of max_depth (i.e. depth of the tree) as number of trees increases in random forest algorithm, the approx mean accuracy in 10 fold increases. **Since, Random forest is based on bagging of trees with generated form random features, it is likely to get better accuracy with more number of trees**. Though there is still possibility of decrease in accuracy after an increase of certain number of trees in random forest (as can be seen from heatmap for max_depth = 5,10, None and for n_estimators =50,150,200). Thus it is a wise choice to go through the dataset and tune the model to find the best fit parameters.

Relative effect for change in n_estimator for Gradient boosting Tree and Random Forest

Looking at both performances we can see the best performance on mean validation accuracy observed for Gradient boosting tree is 93.276 %, On the other hand best performance observed for Random forests is 94.155%. The best accuracy achieved for gradient boosting for n-estimators(maximum number of trees) = 5 while for random forests it is achieved for n_estimators(maximum number of trees) = 50.

- **when we increase n_estimators for Gradient boosting algorithm**, it starts to fit well the training data, but increasing n_estimators more than a certain value might result in overfitting as explained above.
- Also, Gradient tree boosting uses all features that are provided to the model, and iterates over the trees to form best model to fit the data.

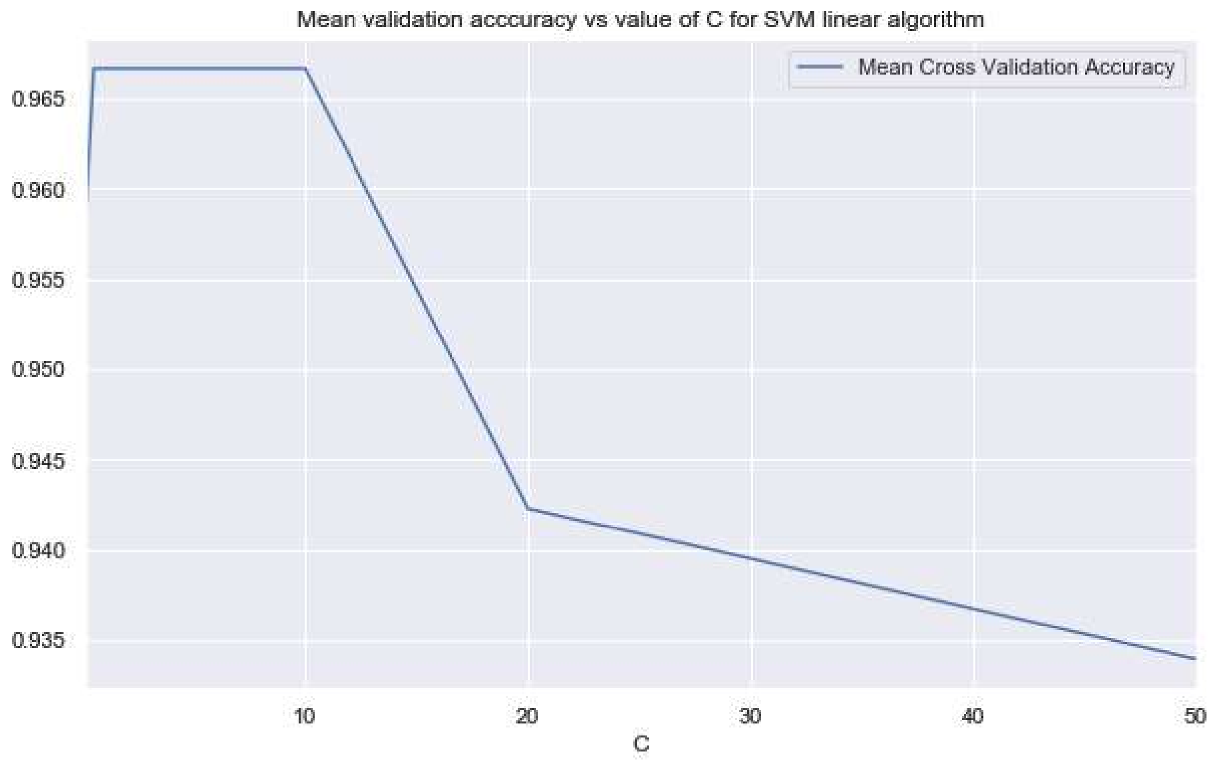
- **While in the case of random forest algorithm**, features are selected randomly and thus depends on how large each tree was constructed and what features were used while generating the individual trees.
- Thus if the depth of trees were kept small, chances of overfitting with increase in number of trees would be small, on the other hand if max_depth per tree is allowed to be large the chances of decrease in mean accuracy with increase in number of trees would be more.

```
============================================================================
```

Q : What does the parameter C define in the SVM classifier? What effect did you observe and why do you think this happened?

Ans : With increase in value of C the accuracy first increased in the 10 fold validation set and then after a certain value it started to decrease.

lets look at SVM mean validation accuracy vs C plot and corresponding data



In [74]:
```
SVM_df
```

Out[74]:

	C	Mean Cross Validation Accuracy
0	0.1	0.957576
1	0.5	0.966667
2	1.0	0.966667
3	2.0	0.966667
4	5.0	0.966667
5	10.0	0.966667
6	20.0	0.942308
7	50.0	0.933974

for C = 0.5,1,2,5,10 , we are getting high mean accuracy using 10 fold validation , but for **C= 2** we are getting small approx running time and hence should use C=2 for test set

Also from the graphs we can see the following observation :

1. When C is small (between 0 to 0.5), the model is getting small accuracy on the validation set, it looks like it underfitted the data.
2. when C value is between (0.5 to 10), the model seem to have good variance and good bias and thus have good accuracy on 10-fold validation.
3. when C is large (more than 10 , in this case), it seems to overfit the training data resulting in comparatively bad performance in validation folds.

From, the above data collected these things can be understood about the optimization function of

SVM classifier

- for small values of C it underfits the data thus allowing more points to be inside the large margin gutter, causing high bias thus model does not perform well on the training as well validation/test sets.
- When value of C is optimal, i.e. not too large or not too low (needs to be tuned usually), the model do not show large bias and variance, thus neither overfits nor underfits the data.
- As value of C is increased to a very large number, it starts to overfits the data, resulting in low-margin fit in the training data thus performs poorly in validation set/ test set

```
In [ ]:
```