In [1]:
```
#importing basic libararies
import numpy as np
import pandas as pd
from sklearn import datasets
```

In [2]:
```
#importing iris dataset from sklearn dataset
iris_data = datasets.load_iris()
```

In [3]:
```
# lets have a look at the dataset
iris_data
```

```
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
```

In [4]:
```
#checking the type of dataset
type(iris_data)
```

Out[4]:
```
sklearn.utils.Bunch
```

In [5]:
```
#looking at the keys of the sklearn iris dataset
dir(iris_data)
```

Out[5]:
```
['DESCR', 'data', 'feature_names', 'filename', 'target', 'target_names']
```

In [6]:
```
#converting iris dataset into iris data frame called 'iris_df'
iris_df = pd.DataFrame(data = np.c_[iris_data['data'],iris_data['target']], colu
# I looked at this source : https://stackoverflow.com/questions/38105539/how-to-
```

Doing basic Exploratory data analysis on the given data

In [7]:
```
#seeing how data looks like
iris_df.head()
```

Out[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

So, we have features as 'sepal length (cm)', 'sepal width (cm)', 'petal length(cm)' and 'petal width(cm)'.
The output is either '0.0', '1.0' or '2.0' which is stored in 'target' column of our dataset.
Form the printed 'iris_data' above we can see : target '0.0' corresponds to class 'setosa', target '1.0' corresponds to class 'versicolor'and target '2.0' corresponds to class 'virginica'.

In [8]:
```
# describing the data
iris_df.describe().T
```

Out[8]:

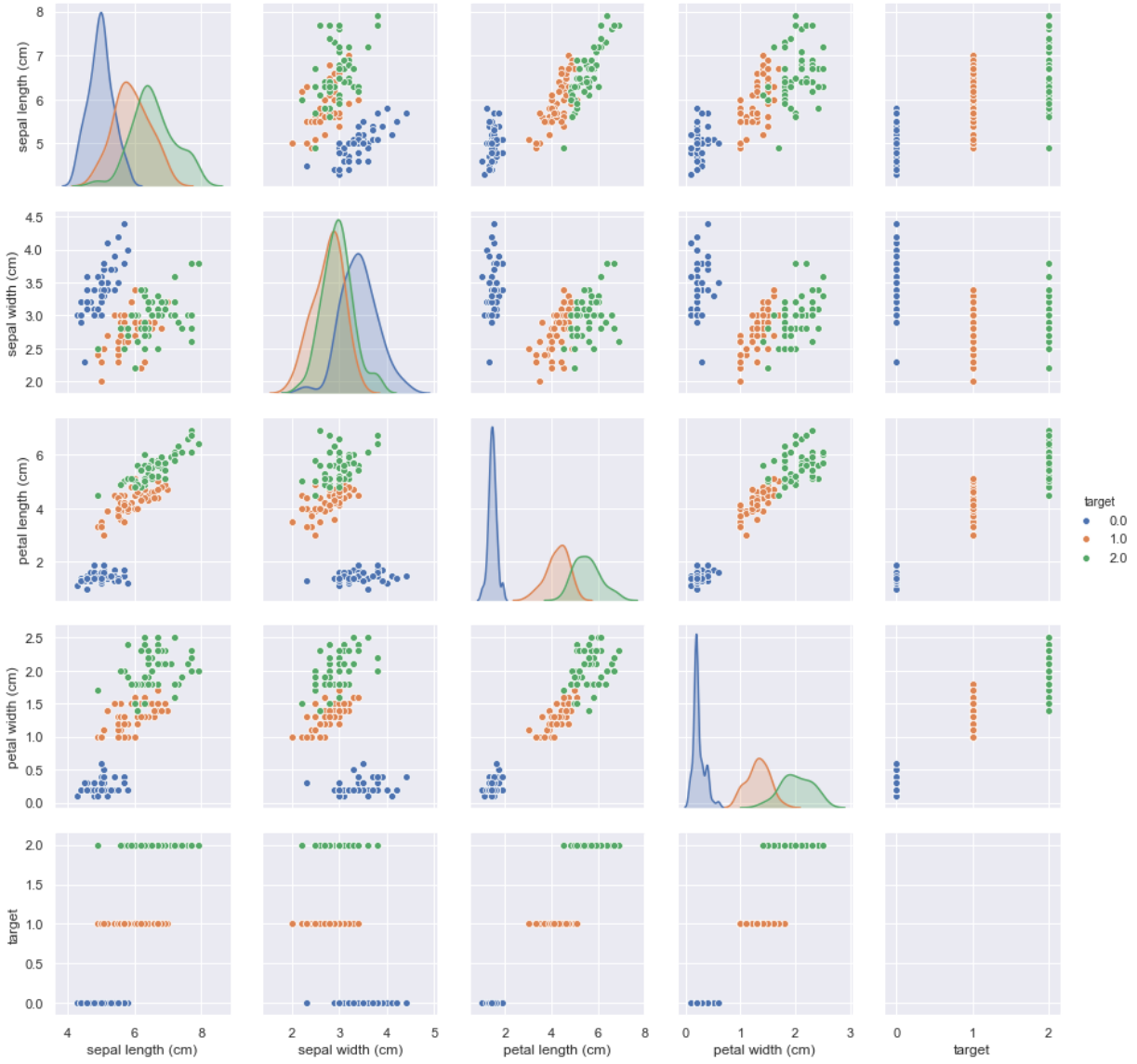
	count	mean	std	min	25%	50%	75%	max
sepal length (cm)	150.0	5.843333	0.828066	4.3	5.1	5.80	6.4	7.9
sepal width (cm)	150.0	3.057333	0.435866	2.0	2.8	3.00	3.3	4.4
petal length (cm)	150.0	3.758000	1.765298	1.0	1.6	4.35	5.1	6.9
petal width (cm)	150.0	1.199333	0.762238	0.1	0.3	1.30	1.8	2.5
target	150.0	1.000000	0.819232	0.0	0.0	1.00	2.0	2.0

In [9]:
```
# let us understand the dataframe information
#, lets see if there is any missing data and understand datatype
iris_df.info(verbose =True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
target               150 non-null float64
dtypes: float64(5)
memory usage: 5.9 KB
```

In [10]:
```
#importing libraries for plotting the graphs
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [11]:
```
#plotting scatter plot among all the features and targets
#p = sns.pairplot(iris_df)
p = sns.pairplot(iris_df, hue = "target")
#g = sns.pairplot(iris_df, hue = "target", vars = iris_df.columns[:-1] )
```



The above graphs statistics tell us :

1. target 0.0 i.e. 'setosa' flower has smallest value for mean sepal length, mean petal length and mean petal width also it has largest mean for sepal width compared to other targets in the dataset.

2. target 1.0 i.e. 'versicolor' flower has avergae value for mean sepal length, mean petal length and petal width compared to other targets in the dataset and smallest sepal width among all targets.
3. target 2.0 i.e. 'virginica' flower has largest value for mean sepal length, mean petal length and mean petal width.

In [12]:
```
#making a copy of dataset to train the model
iris_df2 = iris_df.copy(deep =True)
```

In [13]:
```
#importing useful libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [14]:
```
#preprocessing data
X = iris_df2.drop(['target'], axis =1)
y = iris_df2.target
print ("shape of X is :", X.shape)
print("shape of y is :", y.shape)
```

```
shape of X is : (150, 4)
shape of y is : (150,)
```

KNN Classifier

In [15]:
```
#splitting data into training, validation and test sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(X, y, test_size = 0.
X_train_k, X_valid_k, y_train_k, y_valid_k = train_test_split(X_train_k, y_train_
print( "X_train_k.shape is :", X_train_k.shape)
print( "X_valid_k.shape is :", X_valid_k.shape)
print( "X_valid_k.shape is :", X_valid_k.shape)
print( "y_train_k.shape is :", y_train_k.shape)
print( "y_valid_k.shape is :", y_valid_k.shape)
print( "y_valid_k.shape is :", y_valid_k.shape)
```

```
X_train_k.shape is : (90, 4)
X_valid_k.shape is : (30, 4)
X_valid_k.shape is : (30, 4)
y_train_k.shape is : (90,)
y_valid_k.shape is : (30,)
y_valid_k.shape is : (30,)
```

In [16]:
```
# lets have a look at X_trian_k X_test_k and X_valid_k to make sure that they are
print("\n")
print("Description of training set feature values ")
print(X_train_k.describe().T)
print("\n")
print("Description of validation set feature values ")
print(X_valid_k.describe().T)
print("\n")
print("Description of testing set feature values ")
print(X_test_k.describe().T)
```

```
Description of training set feature values
                   count      mean       std  min  25%  50%  75%  max
sepal length (cm)   90.0  5.846667  0.835074  4.3  5.1  5.8  6.4  7.7
sepal width (cm)    90.0  3.112222  0.462000  2.0  2.8  3.0  3.4  4.4
petal length (cm)   90.0  3.727778  1.797753  1.1  1.5  4.3  5.1  6.7
petal width (cm)    90.0  1.188889  0.769568  0.1  0.3  1.3  1.8  2.5


Description of validation set feature values
                   count      mean       std  min    25%   50%    75%  max
sepal length (cm)   30.0  5.696667  0.791978  4.4  5.200  5.70  6.225  7.7
sepal width (cm)    30.0  2.910000  0.375408  2.2  2.625  3.00  3.100  3.6
petal length (cm)   30.0  3.723333  1.637636  1.0  1.750  4.10  4.975  6.1
petal width (cm)    30.0  1.166667  0.710189  0.2  0.225  1.35  1.750  2.3


Description of testing set feature values
                   count      mean       std  min    25%   50%    75%  max
sepal length (cm)   30.0  5.980000  0.845026  4.7  5.425  6.05  6.500  7.9
sepal width (cm)    30.0  3.040000  0.384708  2.2  2.800  3.00  3.200  3.8
petal length (cm)   30.0  3.883333  1.841305  1.3  1.600  4.50  5.175  6.9
petal width (cm)    30.0  1.263333  0.810910  0.1  0.325  1.35  2.000  2.3
```

In [17]:
```
#lets see performance of KNN classifier, default case on this dataset
Knn_default = KNeighborsClassifier()
Knn_default.fit(X_train_k, y_train_k)
y_pred_test = Knn_default.predict(X_test_k)
score_default = accuracy_score(y_test_k, y_pred_test)
print(" accuracy on default Knn is : ", score_default , sep ="\t")
```

```
 accuracy on default Knn is :   0.9666666666666667
```

The KNN on default case gives an approximate accuracy of 0.967 on the test set

In [18]:
```
def K_performance(k):
    """
    this function takes input k, ie which is fed to KNN as numbers of neighbours
    account while modelling the model
    it returns the accuracy on the validation set of the model
    """
    Knn_model = KNeighborsClassifier(k)
    Knn_model.fit(X_train_k, y_train_k)
    y_pred_cv = Knn_model.predict(X_valid_k)
    score = accuracy_score(y_valid_k,y_pred_cv)
    return score
```

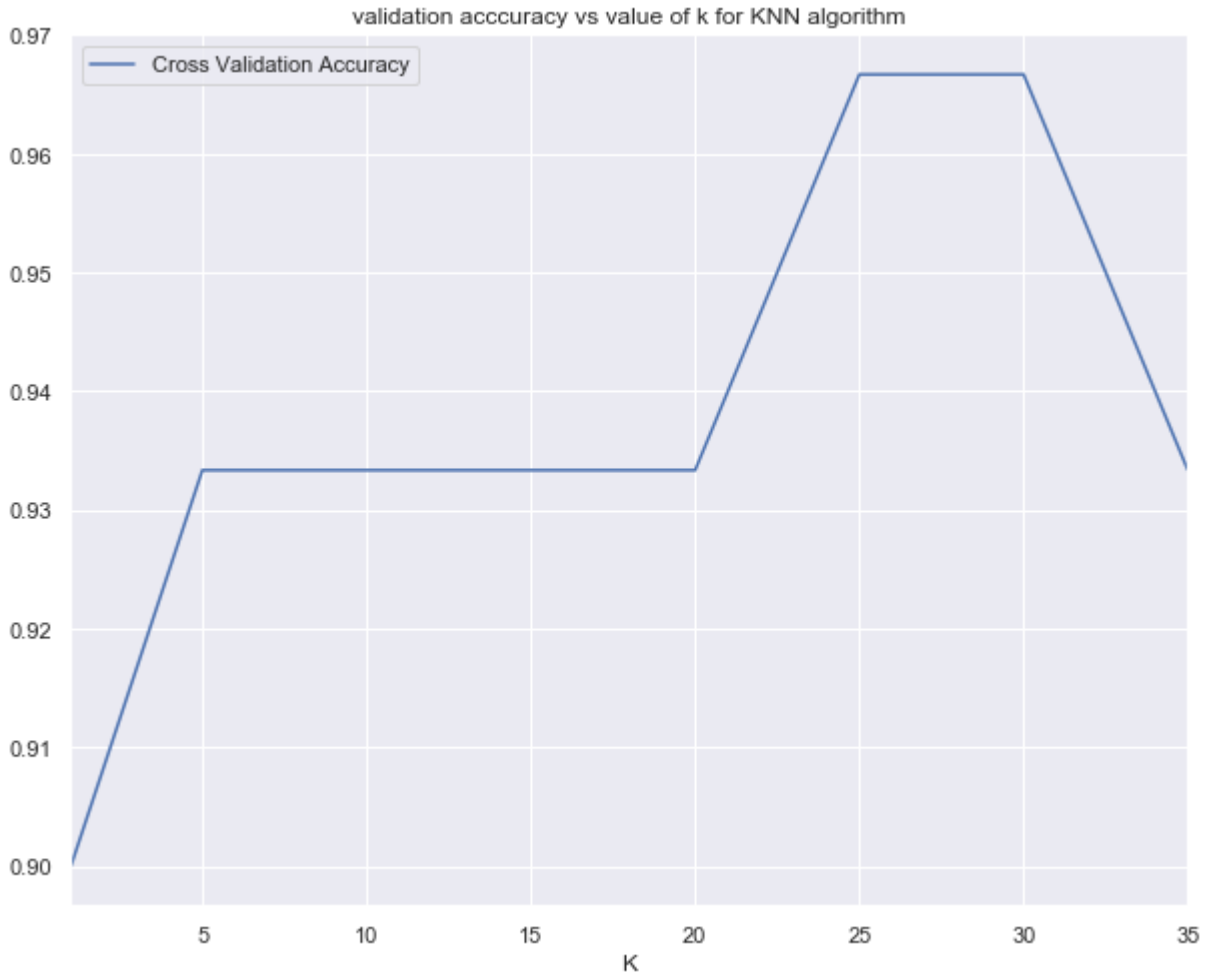
In [19]:
```
# storing KNN model performance (i.e. accuracy score) for various
#k values in the dictionary dict_k
K_vals = [1,5,10,15,20,25,30,35]
dict_k ={}
for k in K_vals:
    temp = K_performance(k)
    dict_k[k] = temp

print ("The dictionary storing accuracy for different values of K is")
print(dict_k)
```

```
The dictionary storing accuracy for different values of K is
{1: 0.9, 5: 0.9333333333333333, 10: 0.9333333333333333, 15: 0.9333333333333333,
20: 0.9333333333333333, 25: 0.9666666666666667, 30: 0.9666666666666667, 35: 0.9
333333333333333}
```

In [20]:
```
#lets see the graph for k vs performance in the validation set
#(I would convert dictionary to dataframe, so as to plot it)
K_val_data = pd.DataFrame(data = list(dict_k.items()))
K_val_data.rename(columns = {0 : "K", 1: "Cross Validation Accuracy"}, inplace =
K_val_data.plot(x= "K", y="Cross Validation Accuracy", title = "validation acccu
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1fdb084b940>



From the Graph, it can be seen, the performance accuracy on validation is highest for k = 25 and 30 ,

1. When K is small, the algorithm is looking only for few neighbors around and hence is performing mediore on validation set, this can be thought as overfitting.
2. From the graph, we can see KNN algorithm perfromance on validation set increases as value of K increase 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 accuracy (here K = 30), seems like it does provide an overly smoothening effect .

In [46]:
```
# writing a function to see performance of Hypertuned model on test set
def K_performance_test(k):
    """
    this function takes input k, ie which is fed to KNN as numbers of neighbours
    account while modelling the model
    it returns the accuracy on the test set of the model
    """
    Knn_model = KNeighborsClassifier(k)
    Knn_model.fit(X_train_k, y_train_k)
    y_pred_test = Knn_model.predict(X_test_k)
    score = accuracy_score(y_test_k,y_pred_test)
    return score
```

In [47]:
```
# lets see performance of KNN for various values of K
K_test = [25,30]
dict_k_t ={}
for k in K_test:
    temp = K_performance_test(k)
    dict_k_t[k] = temp

print("test set accuracy dictionary with key values as k and accuracy as values
```

```
test set accuracy dictionary with key values as k and accuracy as values  {25:
1.0, 30: 1.0}
```

In [65]:
```
import time
K_time = [25,30]
dict_time_t ={}
for k in K_time:
    start = time.time()
    K_performance_test(k)
    stop = time.time()
    dict_time_t[k] = stop - start
print("test set accuracy dictionary with key values as k and approx fit time (in
```

```
test set accuracy dictionary with key values as k and approx fit time (in secon
ds) as values  {25: 0.002956390380859375, 30: 0.004007816314697266}
```

Both for K= 25 and k =30 the model is able to achieve 100% accuracy on the test set. Since K =30 will take more time and would be computationally expensive compared to the case for K = 25. Hence we choose K = 25. Also, K =30 which is even, which is not preffered over an alternative which is odd (K=25).
Thus the best K parameter to consider would be K=25

SVM classifier

In [68]:
```
#importing useful libraries
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
```

In [69]:
```
#preprocessing the dataset,printing shapes of data to be fed
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(X,y,test_size = 0.20
print( "X_train_s.shape is :", X_train_s.shape)
print( "X_test_s.shape is :", X_test_s.shape)
print( "y_train_s.shape is :", y_train_s.shape)
print( "y_test_s.shape is :", y_test_s.shape)
```

```
X_train_s.shape is : (120, 4)
X_test_s.shape is : (30, 4)
y_train_s.shape is : (120,)
y_test_s.shape is : (30,)
```

In [70]:
```
#defining parameters for which are to be optimized
parameters_SVM = {'C':[0.1, 0.5, 1, 2, 5, 10, 20, 50]}
```

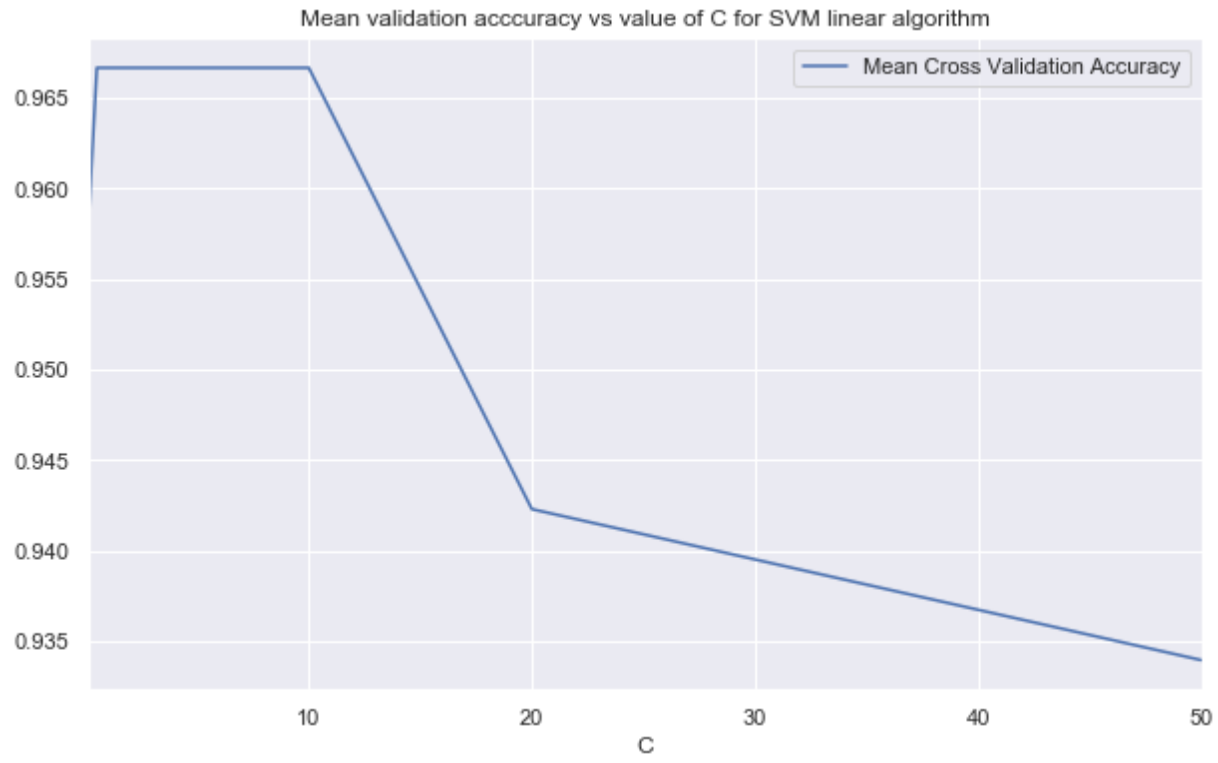
In [145]:
```
#writing function that takes C values as input and returns mean accuracy on 10 fo
def SVM_performance(C):
    """
    this function takes input C, ie which is fed to SVC parameter C to take into
    account while modelling the model
    it returns the mean accuracy of 10-fold validation sets of the model
    """
    SVM_model = SVC(C=C, kernel = 'linear', random_state = 42)
    scores_SVM = cross_val_score(SVM_model, X_train_s, y_train_s, cv=10)
    return scores_SVM.mean()
```

In [146]:
```
#creating dictionary for values, the key of the dictionary stores value of C and
#accuracy on 10 fold cross validation set for the SVM model
dict_SVM = {}
for C_vals in parameters_SVM.values():
    for c in C_vals:
        dict_SVM[c] = SVM_performance(c)
print("The dictionary containing data SVM parameter C as Key and mean accuracy or
print(dict_SVM)
```

```
The dictionary containing data SVM parameter C as Key and mean accuracy on 10 f
old cross validation set for the SVM model
{0.1: 0.9575757575757576, 0.5: 0.9666666666666666, 1: 0.9666666666666666, 2: 0.
9666666666666668, 5: 0.9666666666666668, 10: 0.9666666666666668, 20: 0.94230769
23076923, 50: 0.933974358974359}
```

In [147]:
```
#Plotting C vs Mean Cross Validation Accuracy for SVM model
SVM_df = pd.DataFrame.from_dict(data = list(dict_SVM.items()))
SVM_df.rename(columns = {0 : "C", 1: "Mean Cross Validation Accuracy"}, inplace =
SVM_df.plot(x= "C", y="Mean Cross Validation Accuracy",title = "Mean validation a
           figsize = (10,6))
```

Out[147]: <matplotlib.axes._subplots.AxesSubplot at 0x25400fb6c88>



In [60]:
```
#just for checking, Approximately how long does it takes for the PC to run the SV
C_acc = [0.5,1,2,5,10]
import time
C_time ={}
for c in C_acc :
    start = time.time()
    SVM_performance(c)
    end = time.time()
    C_time[c] = end -start
SVM_df_time = pd.DataFrame.from_dict(data = list(C_time.items()))
print("dataframe containing approx time for SVM model to fit ")
SVM_df_time.rename(columns = {0 : "C", 1: "Approx time to fit"}, inplace =True)
SVM_df_time
```

```
dataframe containing approx time for SVM model to fit
```

Out[60]:

	C	Approx time to fit
0	0.5	0.025409
1	1.0	0.023936
2	2.0	0.021941
3	5.0	0.022938
4	10.0	0.021942

for C = 0.5,1,2,5,10 , we are getting high mean accuracy(0.9666666666667) on 10 fold cross validation , but for **C= 2** we are getting small approx fitting time and hence should use C= 2 for test set

Also from the graph "Mean validation acccuracy vs value of C for SVM linear algorithm" 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.

In [74]:
```
def C_performance_test(C):
    """
    this function takes input C, ie which is fed to SVC
    it returns the accuracy on the test set of the model
    """
    svc_model = SVC(C=C, kernel ='linear')
    svc_model.fit(X_train_s, y_train_s)
    y_pred_test = svc_model.predict(X_test_s)
    score = accuracy_score(y_test_s,y_pred_test)
    return score
```

The model gets high 10 fold accuracy of for C = 0.5, C = 1 and C=2 Since, we got samllest approx fit time for C=2 while fittig the model on 10-fold cross validation, best value for the parameter would be **C=2**

In [118]:
```
C_test = [2]
dict_C_t ={}
for C in C_test:
    temp = C_performance_test(C)
    dict_C_t[C] = temp
print("test set accuracy dictionary with key values as C for SVM model and accura
```

```
test set accuracy dictionary with key values as C for SVM model and accuracy as
values  {2: 1.0}
```

In [38]:
```
#Lets check SVM performance for the default case :
svc_model_d = SVC(kernel ='linear')
svc_model_d.fit(X_train_s, y_train_s)
y_pred_test = svc_model_d.predict(X_test_s)
score = accuracy_score(y_test_s,y_pred_test)

print("Model accuracy for deafult case is ", score)
```

```
Model accuracy for deafult case is  1.0
```

Decision Tree model

In [80]:
```
#importing useful libraries
from sklearn.tree import DecisionTreeClassifier
```

In [177]:
```
#preprocessing the dataset, printing shapes of data to be fed
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(X,y,test_size = (
print( "X_train_dt.shape is :", X_train_dt.shape)
print( "X_test_dt.shape is :", X_test_dt.shape)
print( "y_train_dt.shape is :", y_train_dt.shape)
print( "y_test_dt.shape is :", y_test_dt.shape)
```

```
X_train_dt.shape is : (120, 4)
X_test_dt.shape is : (30, 4)
y_train_dt.shape is : (120,)
y_test_dt.shape is : (30,)
```

In [178]:
```
def DT_performance(max_depth):
    """
    this function takes input max_depth, ie which is fed to  DecisionTreeClassifi
    account while modelling the model
    it returns the mean accuracy of 10-fold validation sets of the model
    """
    DT_model = DecisionTreeClassifier(random_state =42 , max_depth = max_depth)
    scores_DT = cross_val_score(DT_model, X_train_dt, y_train_dt, cv=10)
    return scores_DT.mean()
```

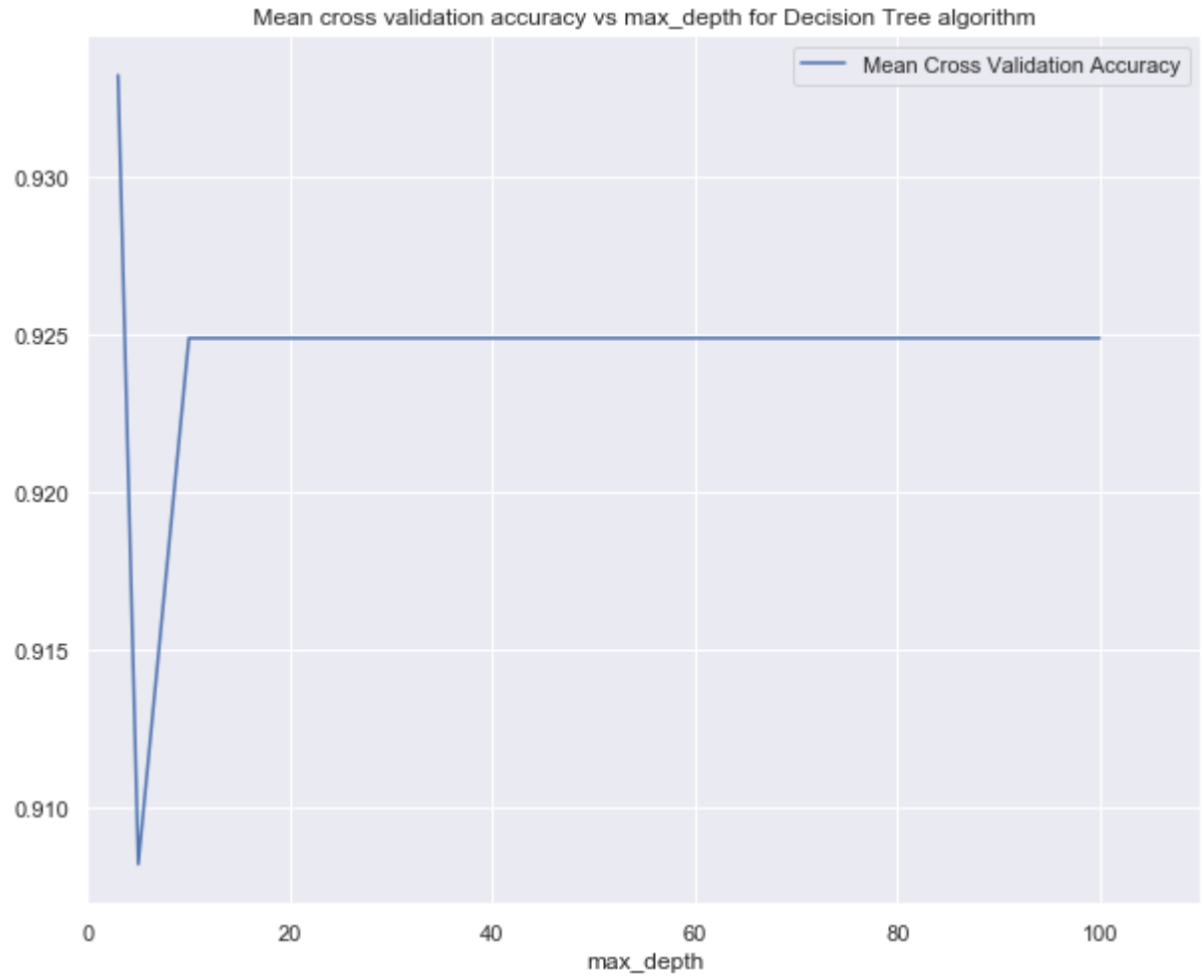
In [179]:
```
parameters_DT = {'max_depth' : [3,5,10,None]}
```

In [180]:
```
#creating dictionary for values
dict_DT = {}
for max_depth_vals in parameters_DT.values():
    for Max_depth in max_depth_vals:
        dict_DT[Max_depth] = DT_performance(Max_depth)
print("Dictionary  : Decision tree score for variou max_Depth values on 10 -fold
```

```
Dictionary  : Decision tree score for variou max_Depth values on 10 -fold valdi
ation set is :  {3: 0.9332167832167831, 5: 0.908216783216783, 10: 0.92488344988
34497, None: 0.9248834498834497}
```

In [175]:
```
#checking the results of hyperparameter tuning
# for plotting purpose none is changed to 100 (none : plotting depth till we ach
#none can be replace with 100)
Dict_DT1 = {3: 0.9332167832167831, 5: 0.908216783216783, 10: 0.9248834498834497,
DT_df = pd.DataFrame.from_dict(data = list(Dict_DT1.items()))
DT_df.rename(columns = {0 : "max_depth", 1: "Mean Cross Validation Accuracy"}, i
DT_df.plot(x= "max_depth", y="Mean Cross Validation Accuracy", title = " Mean cr
          , figsize = (10,8), xlim =(0,110))
```

Out[175]: <matplotlib.axes._subplots.AxesSubplot at 0x2540103f9e8>



We are getting highest mean accuarcy at max_depth =3 , gives an idea that there would be some overfiting, as the max_depth of tree is allowed to be more. By looking at the plot we can say, The best parameter would be max_depth = 3, also the dataset is small (150 samples total), thus there are more possibilities of data to be overfitted as we increase the max_depth.

In [181]:
```
#Lets see performance in test -set for max_depth = 3 ie lets look at optimized de
model_dt_o = DecisionTreeClassifier(max_depth = 3, random_state= 42)
model_dt_o.fit(X_train_dt, y_train_dt)
y_pred_test = model_dt_o.predict(X_test_dt)
score_dt = accuracy_score(y_test_dt, y_pred_test)
print("Test set Accuracy of decision Tree for max_depth = 3  is ", score_dt)
```

```
Test set Accuracy of decision Tree for max_depth = 3  is  1.0
```

In [182]:
```
#Lets see performance in test -set for max_depth = 3 ie lets look at optimized de
model_dt_d = DecisionTreeClassifier(random_state = 42 )
model_dt_d.fit(X_train_dt, y_train_dt)
y_pred_test = model_dt_d.predict(X_test_dt)
score_dt = accuracy_score(y_test_dt, y_pred_test)
print("Test set accuracy of decision Tree for max_depth as default is ", score_d
```

```
Test set accuracy of decision Tree for max_depth as default is  1.0
```

Random Forest

In [106]:
```
#importing useful libraries
from sklearn.ensemble import RandomForestClassifier
```

In [107]:
```
#preprocessing the dataset and printing shapes of data to be fed
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X,y,test_size = (
print( "X_train_rf.shape is :", X_train_rf.shape)
print( "X_test_rf.shape is :", X_test_rf.shape)
print( "y_train_rf.shape is :", y_train_rf.shape)
print( "y_test_rf.shape is :", y_test_rf.shape)
```

```
X_train_rf.shape is : (120, 4)
X_test_rf.shape is : (30, 4)
y_train_rf.shape is : (120,)
y_test_rf.shape is : (30,)
```

In [108]:
```
def RF_performance(max_depth, n_estimators):
    """
    this function takes input max_depth, ie which is fed to  DecisionTreeClassifi
    account while modelling the model
    it returns the mean accuracy of 10-fold validation sets of the model
    """
    RF_model = RandomForestClassifier(random_state = 42, max_depth = max_depth, r
    scores_RF = cross_val_score(RF_model, X_train_rf, y_train_rf, cv=10)
    return scores_RF.mean()
```

In [109]:
```
#fitting the dataset with multiple models created by multiple hyperparameters
max_depth_list = [3,5,10,None]
n_estimators_list = [5,10,50,150,200]

# creating a pair list for parameters
mega_list = []
for Max_depth in max_depth_list:
    for N_estimator in n_estimators_list:
        mega_list.append([Max_depth, N_estimator])

# creating a dictionary for better record_keeping
mega_dict = {}
for i in range(len(mega_list)):
    mega_dict[i] = mega_list[i]

# Saving accuracy scores in the dictionary
for element_pair in mega_dict:
    mega_dict[element_pair].append(RF_performance(mega_dict[element_pair][0], meg

# Saving Time taken to fit in the dictionary

for element_trio in mega_dict:
    start = time.time()
    RF_performance(mega_dict[element_trio][0], mega_dict[element_trio][1])
    end = time.time()
    mega_dict[element_trio].append(end-start)
```

In [110]:
```
# converting None values to 1000 values for plotting purposes
for triple_element in mega_dict.values():
    if triple_element[0] == None:
        triple_element[0] =1000
```

In [111]:
```
RF_df = pd.DataFrame.from_dict(data = mega_dict)
```

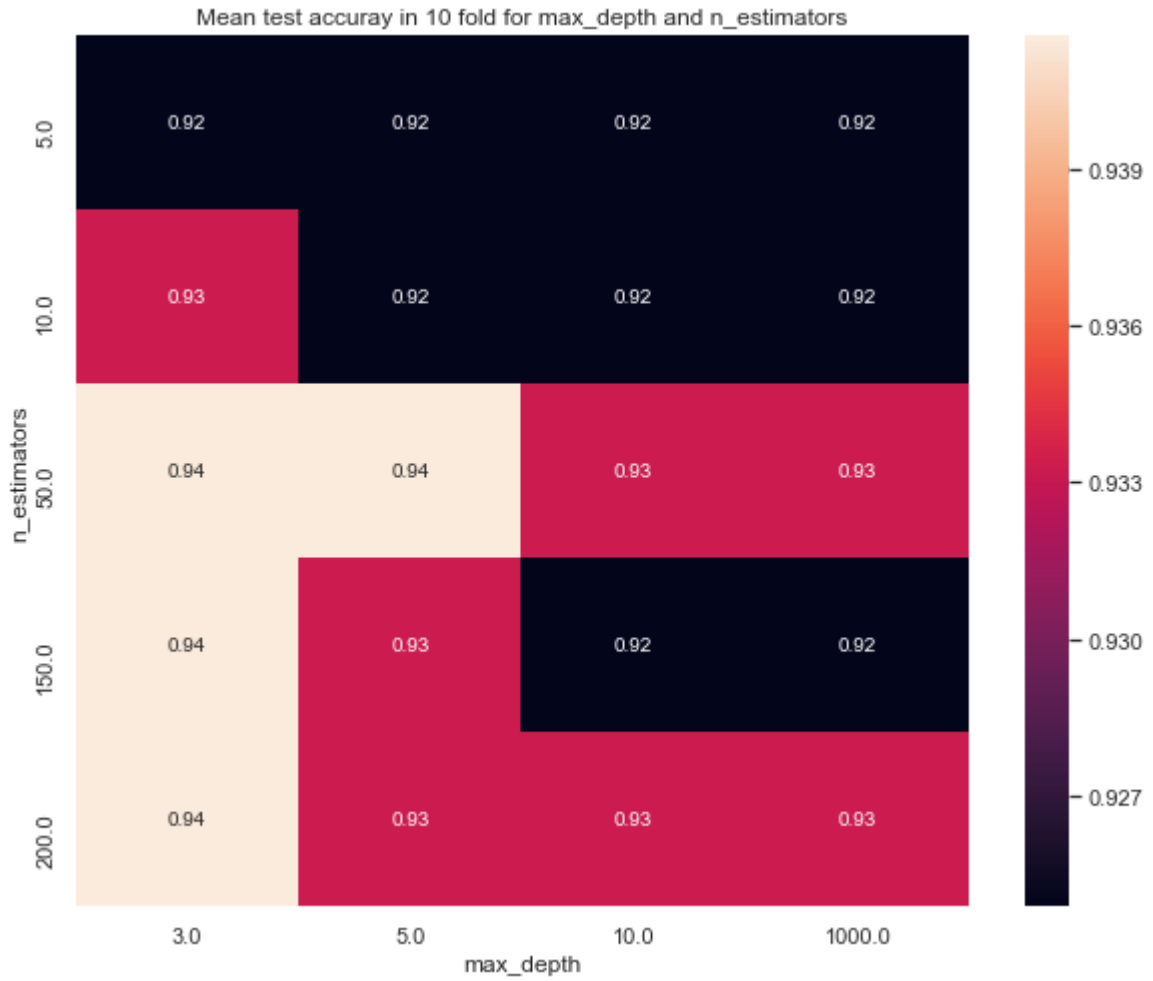
In [112]:
```
RF_df = RF_df.T
```

In [113]:
```
RF_df.rename(columns = {0: "max_depth", 1: "n_estimators", 2: " mean test accuray
```

In [137]:
```
RF_df1 = RF_df.drop(['approx time taken'], axis =1)
RF_df2 = RF_df.drop([' mean test accuray in k fold'], axis = 1)
```

In [138]:
```
plt.figure(figsize=(10,8))
plt.title("Mean test accuray in 10 fold for max_depth and n_estimators")
RF_df1 = RF_df1.pivot('n_estimators','max_depth',' mean test accuray in k fold')
#plt.imshow(RF_df1, annot = True)
sns.heatmap(data = RF_df1, annot=True)
#RF_df1
```

Out[138]: <matplotlib.axes._subplots.AxesSubplot at 0x2540056cba8>



When number of estimators ie number of treees are large and max_depth is more than 3, it seems like most of formed trees face problem of overfitting. for max_depth = 3, an increase in number of estimators increase mean accuracy in the validation set

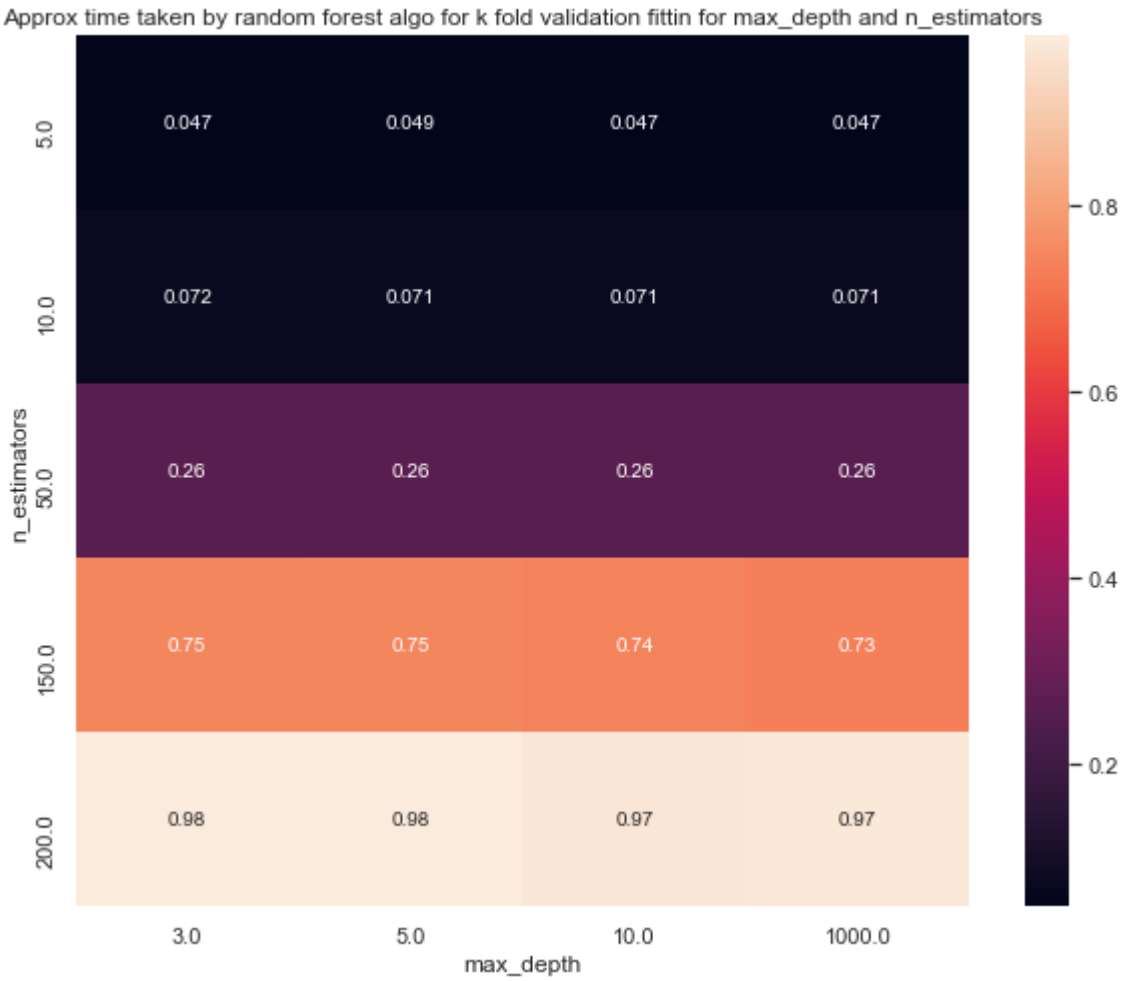
The best accuracy on 10 fold- validation (0.94) is obtained for following pair of paramaeters :

1. n estimators : 50, max depth : 3

2. n_estimators : 150, max_depth : 3
3. n_estimators : 200, max_depth : 3
4. n_estimators : 50, max_depth : 5

In [139]:
```
plt.figure(figsize=(10,8))
plt.title("Approx time taken by random forest algo for k fold validation fittin
RF_df2 = RF_df2.pivot('n_estimators','max_depth','approx time taken')
sns.heatmap(data = RF_df2, annot=True)
```

Out[139]: <matplotlib.axes._subplots.AxesSubplot at 0x254005d4c18>



By looking at the above heat map representing "approx time taken by random forest algo for k fold validation fittin for max_depth and n_estimators"

Comparing two heat maps, we can see **n_estimator = 50.0 and max_depth =3 gives the most optimal performance in terms of time and accuracy** amongst the all tried combination of parameters.

In [120]:
```
#Calculating accuracy on test set performnace for various parameters for Random F
RF_model = RandomForestClassifier(random_state = 42, max_depth = 3, n_estimators
RF_model.fit(X_train_rf, y_train_rf)
y_pred_RF= RF_model.predict(X_test_rf)
accuracy_rf_best = accuracy_score (y_test_rf, y_pred_RF)
print(" The accuracy of Random forest model on test set, for tuned hyperparameter
```

```
 The accuracy of Random forest model on test set, for tuned hyperparameters max
_depth =3 and n_estimators = 50 is  1.0
```

Gradient Boosting Tree

In [124]:
```
#importing useful libraries
from sklearn.ensemble import GradientBoostingClassifier
```

In [125]:
```
#preprocessing the dataset, printing shapes of data to be fed
X_train_gb, X_test_gb, y_train_gb, y_test_gb = train_test_split(X,y,test_size = 0
print( "X_train_gb.shape is :", X_train_gb.shape)
print( "X_test_gb.shape is :", X_test_gb.shape)
print( "y_train_gb.shape is :", y_train_gb.shape)
print( "y_test_gb.shape is :", y_test_gb.shape)
```

```
X_train_gb.shape is : (120, 4)
X_test_gb.shape is : (30, 4)
y_train_gb.shape is : (120,)
y_test_gb.shape is : (30,)
```

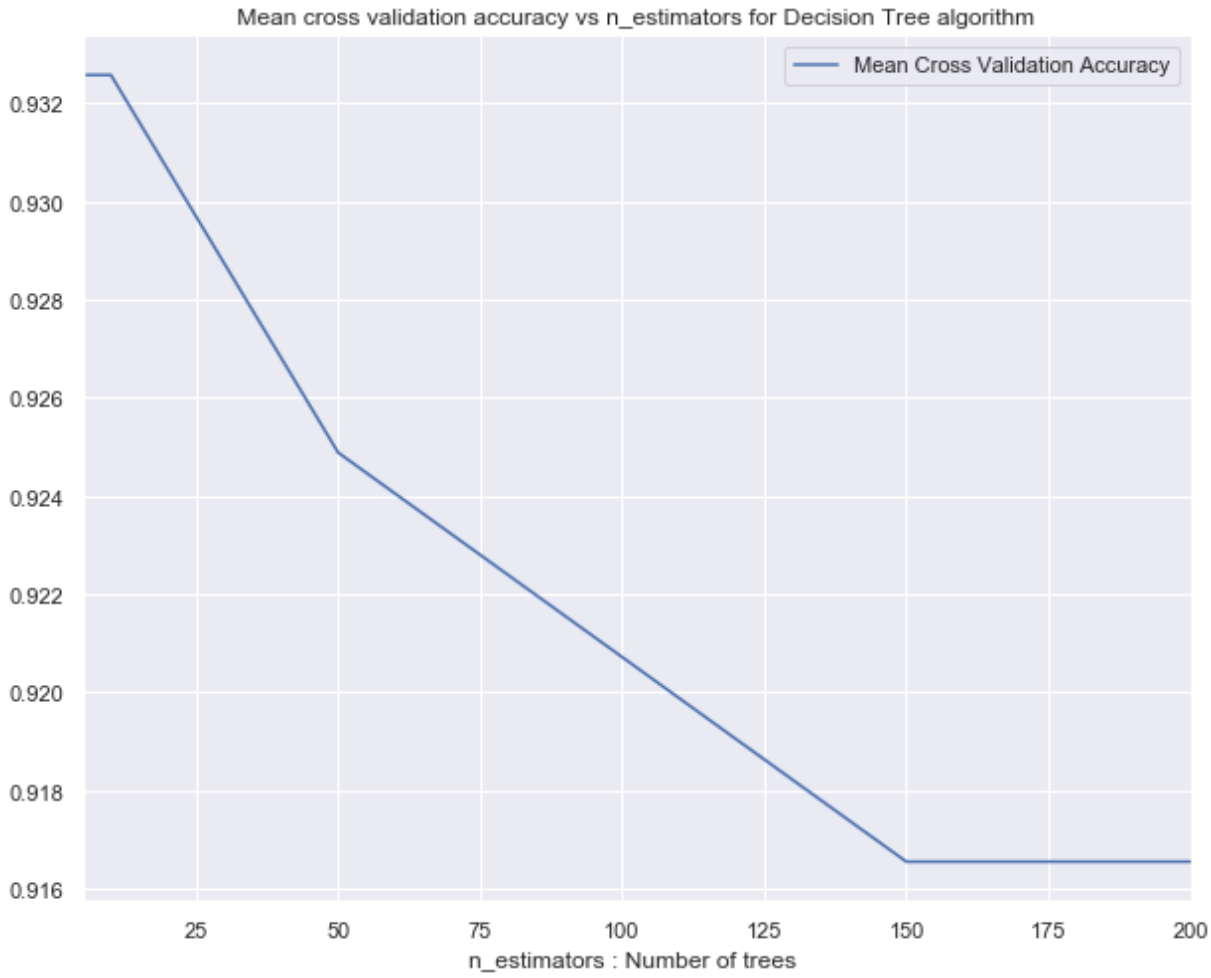
In [126]:
```
def GB_performance(n_estimators):
    """
    this function takes input n_estimators, ie which is fed to  GradientBoosting
    account while modelling the model
    it returns the mean accuracy of 10-fold validation sets of the model
    """
    GB_model = GradientBoostingClassifier(random_state = 42, n_estimators = n_est
    scores_GB = cross_val_score(GB_model, X_train_gb, y_train_gb, cv=10)
    #print(scores_GB)
    return scores_GB.mean()
```

In [127]:
```
#fitting the dataset with multiple models created by multiple hyperparameters
parameters_GB = {'n_estimators' : [5,10,50,150,200]}
```

In [128]:
```
#creating dictionary for values
dict_GB = {}
for n_estimators_vals in parameters_GB.values():
    for N_estimators in n_estimators_vals:
        dict_GB[N_estimators] = GB_performance(N_estimators)
```

In [129]:
```
#checking the results of hyperparameter tuning
GB_df = pd.DataFrame.from_dict(data = list(dict_GB.items()))
GB_df.rename(columns = {0 : "n_estimators : Number of trees ", 1: "Mean Cross Va
GB_df.plot(x= "n_estimators : Number of trees ", y= "Mean Cross Validation Accura
          , figsize = (10,8))
```

Out[129]: <matplotlib.axes._subplots.AxesSubplot at 0x2547c186198>



In [141]:
```
print("The dictionary representing 10-fold accuracy vs n_estimator with keys as n
```

```
The dictionary representing 10-fold accuracy vs n_estimator with keys as n_esti
mators and values as corresponding accuracy {5: 0.9325757575757574, 10: 0.93257
57575757574, 50: 0.9248834498834497, 150: 0.9165501165501164, 200: 0.9165501165
501164}
```

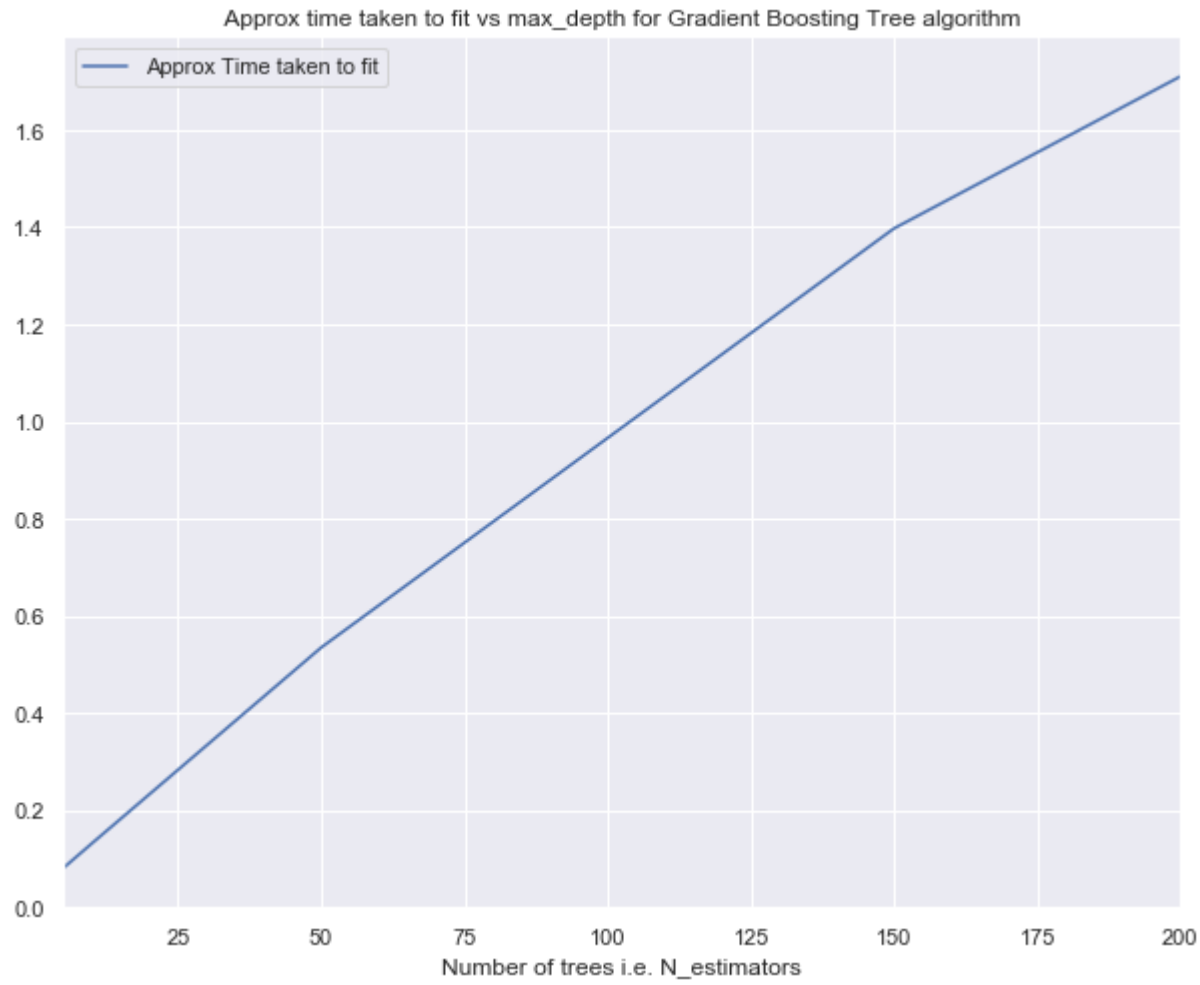
From the graph, it looks like increasing number of trees in gradient boosting trees increases fit in the training set and as it goes over a particular limit, it (n_estimators) starts to overfit and hence decreases performances in the validation set

Also accuaracy on 10 fold validation set for n_estimators =5 and n_estimators = 10 turns out exactly similar (equal to 0.93257), lets see the time taken by them

In [132]:
```
N_est_acc = [5,10,50,150,200]
N_est_time = {}
for n in N_est_acc :
    start = time.time()
    GB_performance(n)
    end = time.time()
    N_est_time[n] = end - start

GB_df_time = pd.DataFrame.from_dict(data = list(N_est_time.items()))
GB_df_time.rename(columns = {0 : "Number of trees i.e. N_estimators", 1: "Approx
GB_df_time.plot(x= "Number of trees i.e. N_estimators", y="Approx Time taken to
          , figsize = (10,8))
print(N_est_time)
```

```
{5: 0.08078479766845703, 10: 0.13164806365966797, 50: 0.5346009731292725, 150:
1.3974201679229736, 200: 1.7103703022003174}
```



By looking at approx time taken by algorithm for various number of trees(refering above plot : Approx time taken to fit vs max_depth for Gradient Boosting Tree algorithm), we can say , it takes less time for less number of trees.

Thus the best parameter for this case would be n_estimators = 5

In [133]:
```
#Lets see performance in test -set for n_estimators = 5 ie lets look at optimized
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 is
```

```
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.

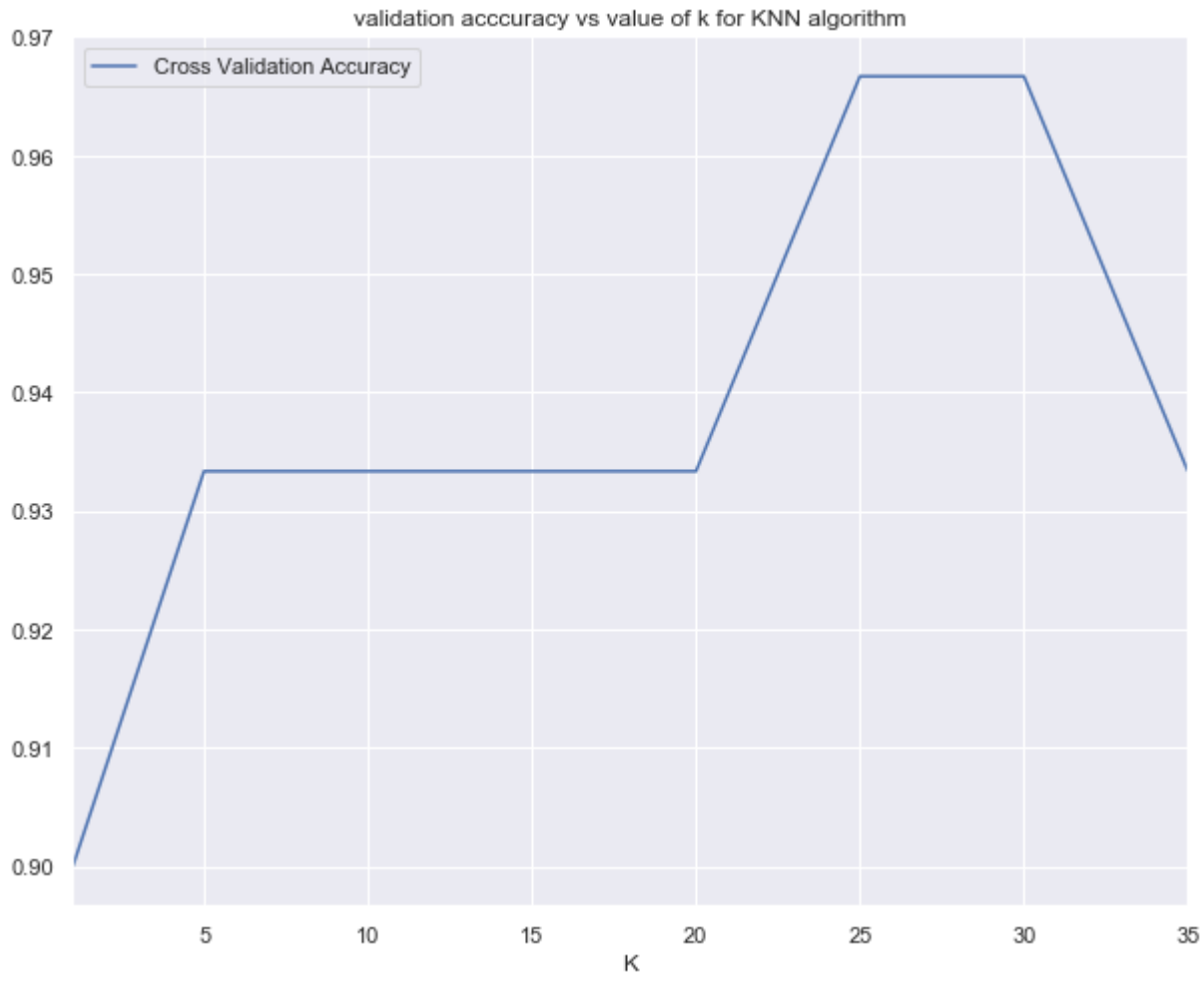
Thus validation set is used to ensure that we still have a dataset (i.e. test dataset) to measure the performance of model on new data(other than training data and validation data) and parameters that makes algorithm perform well on validation dataset would be used as final parameters for the model,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

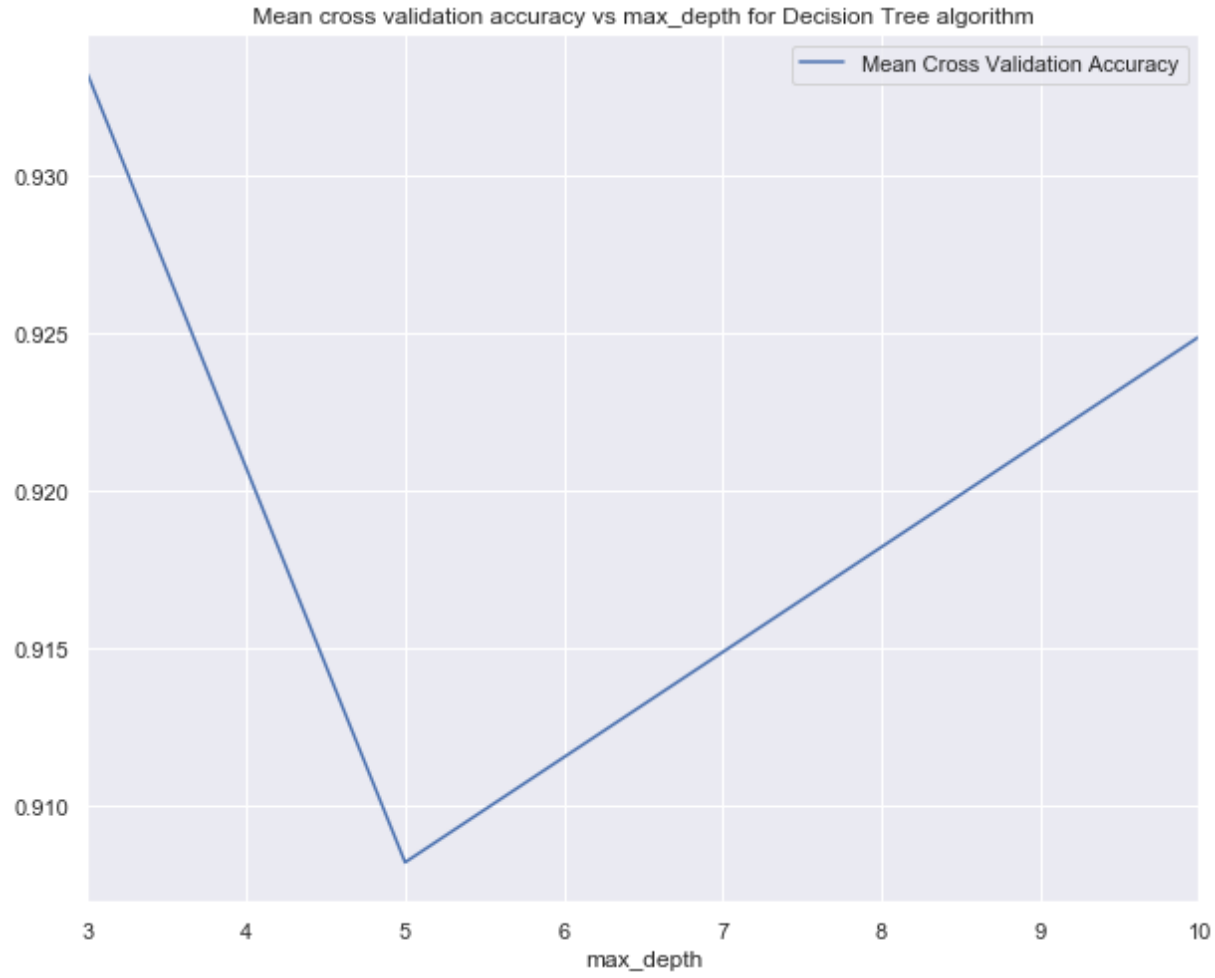
1. When K is small, the algorithm is looking only for few neighbors around and hence is performing mediore which can be attributed to overfitting.
2. From the graph, we can see KNN algorithm perfromance on validation set increases as value of K increases upto a certain value of K.(these values of K seem 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]:

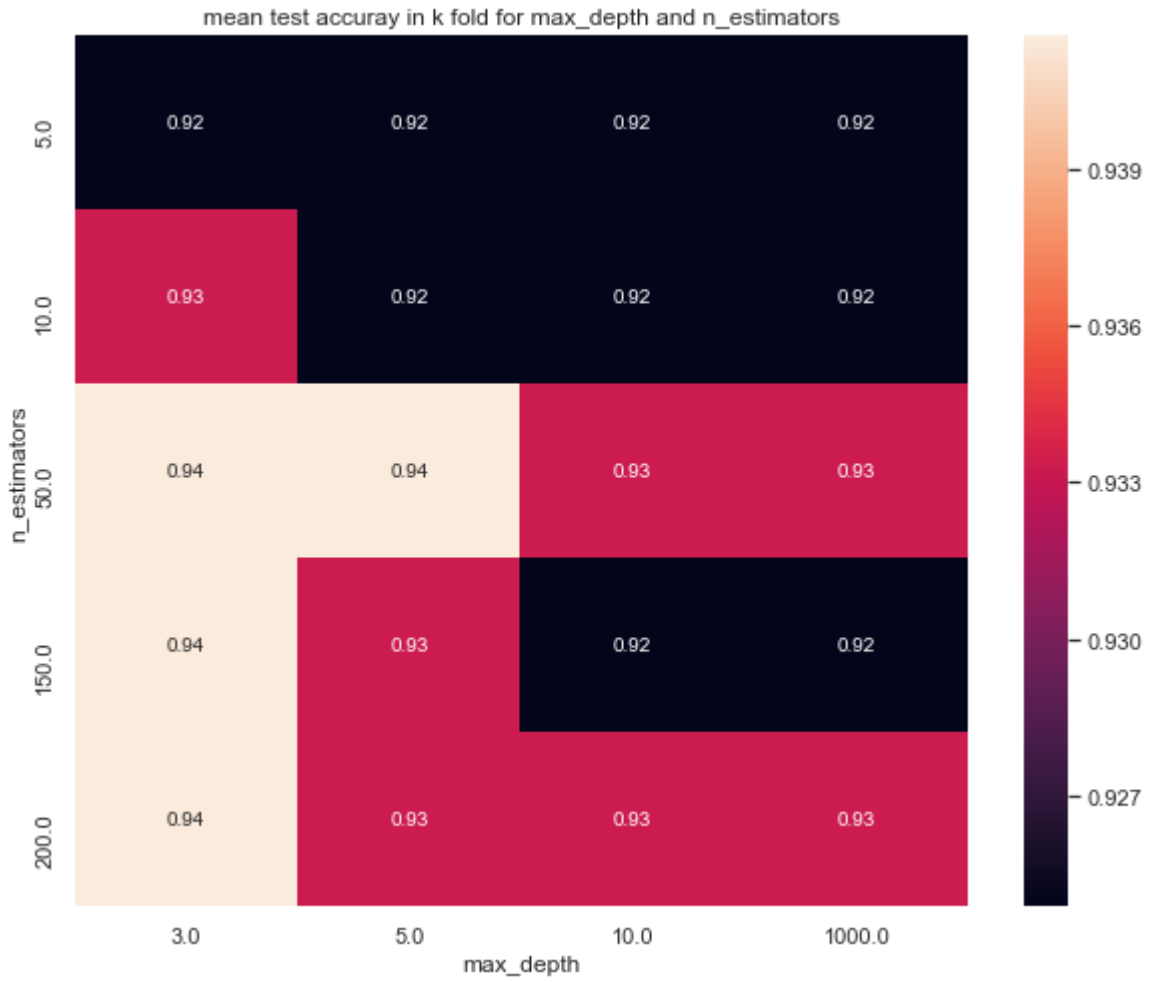
	max_depth	Mean Cross Validation Accuracy
0	3.0	0.933217
1	5.0	0.908217
2	10.0	0.924883
3	NaN	0.924883

- 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.

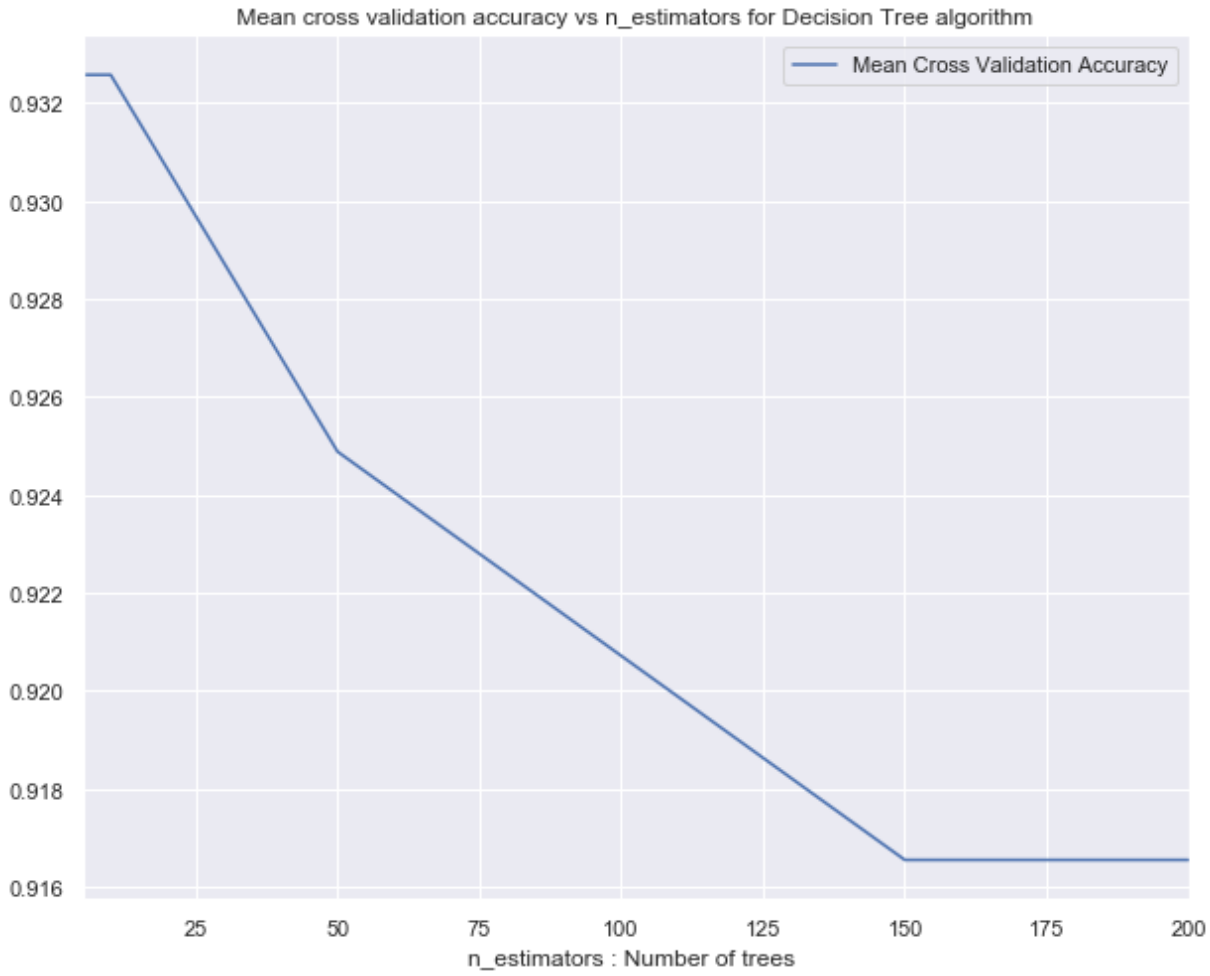
Thus we can conclude, increasing the max_depth might result in overfitting for decision trees and random forests, and should be done iterativey on validation set to tune it for getting a good performance on unseen data(test set).

==========

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



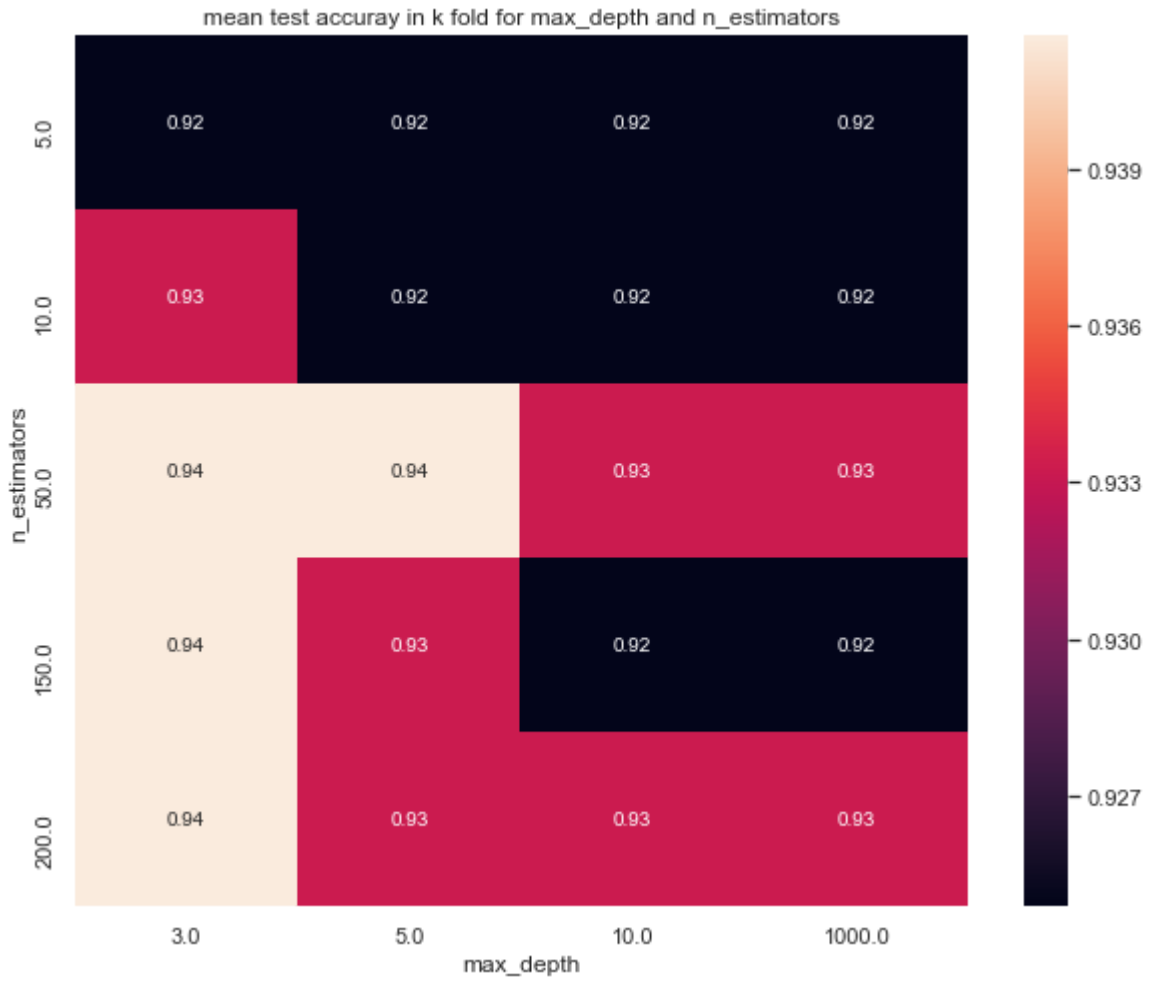
In [72]:
```
GB_df
```

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.

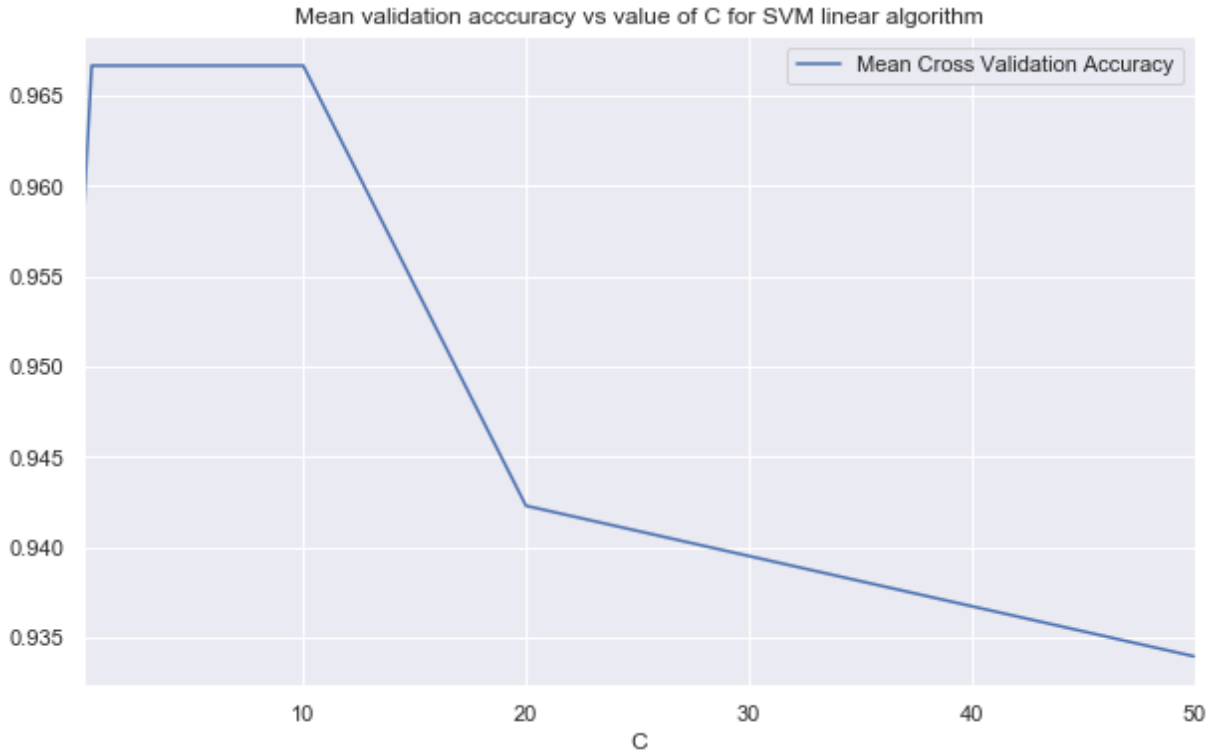
- **While in the case of random forest algorithm**, features are selected randomly and thus depends on how large each tree was constructed and what features were used while generating the individual trees.
- Thus if the depth of trees were kept small, chances of overfitting with increase in number of 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 [ ]: