Seeds Data Set

Measurements of geometrical properties of kernels belonging to three different varieties of wheat. A soft X-ray technique and GRAINS package were used to construct all seven, real-valued attributes.

Attribute Information:

1. area A,
2. perimeter P
3. Compactness
4. Length of Kernel
5. Width of Kernel
6. Asymmetry coefficient
7. Length of kernel groove.
8. Type

In [1]:

```
import numpy as np
import pandas as pd

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

In [2]:

```
df = pd.read_csv('/Users/rod/Documents/UCSD/COGS/COGS_118A/Project/Seeds/seeds_d
ataset.csv')#.astype(np.float32)
df.columns = ['Area','Perimeter','Compactness','Length','Width','Asymmetry','Gro
ove Length', 'Type']
#df
```

In [3]:

```
print('df type: ' + str(type(df)))
print('df size: ' + str(df.shape))
df.head()
```

```
df type: <class 'pandas.core.frame.DataFrame'>
df size: (211, 8)
```

Out[3]:

	Area	Perimeter	Compactness	Length	Width	Asymmetry	Groove Length	Type
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
1	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
2	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1
3	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1
4	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1

Oberving the Data Set

1. Checks for null values.
2. Sees how many classes/categories there are.
3. Counts the data points that belong to each category.

In [4]:

```
#Observing the Leaf Data Set.
print('Number of NULL values in df: ' + str(df.isnull().sum().sum()))

uniqueClasses = df['Type'].unique()
print('Number of unique classes in df: ' + str(uniqueClasses.shape))
uniqueClasses = np.sort(uniqueClasses)

for i in uniqueClasses:
    print('Class ' + str(i) + ' count: ' + str((df['Type']==i).sum()))
```

```
Number of NULL values in df: 0
Number of unique classes in df: (3,)
Class 1 count: 71
Class 2 count: 70
Class 3 count: 70
```

Shuffle Data Randomly

1. Saves the first random shuffle of the original df.
2. Saves the second random shuffle of the original df.
3. Saves the third random shuffle of the original df.

In [5]:

```
df_shuffle1 = df.sample(frac=1)
#df_shuffle1.head()
```

In [6]:

```
df_shuffle2 = df.sample(frac=1)
#df_shuffle2.head()
```

In [7]:

```
df_shuffle3 = df.sample(frac=1)
#df_shuffle3.head()
```

F(X) = Y

Separates data into X and Y (labels) to set up the rest of the supervised learning algos in the [F(X) = Y] format.

1. Sets up F(X1) = Y1 from the first random shuffle of the original df.
2. Sets up F(X2) = Y2 from the second random shuffle of the original df.
3. Sets up F(X3) = Y3 from the third random shuffle of the orignal df.

In [8]:

```
df_array1 = np.array(df_shuffle1)                 #Convert dataframe to array in or
der to slice into X and Y.

X1 = df_array1[:, 0:(df_array1.shape[1] - 1)]  #First Column to second before la
st column. All numerical Features.
Y1 = df_array1[:, (df_array1.shape[1] - 1)]    #Last column represents the class
es which are all numerical.
print('X1 shape: ' + str(X1.shape))
print('Y1 shape: ' + str(Y1.shape))
```

```
X1 shape: (211, 7)
Y1 shape: (211,)
```

In [9]:

```
df_array2 = np.array(df_shuffle2)                #Convert dataframe to array in or
der to slice into X and Y.

X2 = df_array2[:, 0:(df_array2.shape[1] - 1)]  #First Column to second before la
st column. All numerical Features.
Y2 = df_array2[:, (df_array2.shape[1] - 1)]    #Last column represents the class
es which are all numerical.
print('X2 shape: ' + str(X2.shape))
print('Y2 shape: ' + str(Y2.shape))
```

```
X2 shape: (211, 7)
Y2 shape: (211,)
```

In [10]:

```
df_array3 = np.array(df_shuffle3)                #Convert dataframe to array in or
der to slice into X and Y.

X3 = df_array3[:, 0:(df_array3.shape[1] - 1)]  #First Column to second before la
st column. All numerical Features.
Y3 = df_array3[:, (df_array3.shape[1] - 1)]    #Last column represents the class
es which are all numerical.
print('X3 shape: ' + str(X3.shape))
print('Y3 shape: ' + str(Y3.shape))
```

```
X3 shape: (211, 7)
Y3 shape: (211,)
```

Functions Used For All Classifiers

1. partitionData
2. viewSplit
3. draw_heatmap_linear
4. bestValue
5. ViewConfusionMatrix
6. displayAccuracies

In [11]:

```
#X: Features of df.
#Y: Labels of df.
#percent: The percentage given to the training_validation set.
def partitionData(X, Y, percent):
    X_train_val = X[:int(percent*len(X))] # Get features from train + val set.
    Y_train_val = Y[:int(percent*len(Y))] # Get labels from train + val set.
    X_test      = X[int(percent*len(X)):] # Get features from test set.
    Y_test      = Y[int(percent*len(Y)):] # Get labels from test set.

    return X_train_val, Y_train_val, X_test, Y_test
```

In [12]:

```
#PURPOSE: Used to see the dimensions of the data after being partioned.
#Prints the shape of X_train_val.
#Prints the shape of Y_train_val.
#Prints the shape of X_test.
#Prints the shape of Y_test.
#Prints num of UNIQUE classes in Y_train_val.
#Prints the num of data points that belong to each class/category.
#Prints num of UNIQUE classes in Y_test.
def viewSplit(X_train_val, Y_train_val, X_test, Y_test):
    print('X_train_val shape: ' + str(X_train_val.shape))
    print('Y_train_val shape: ' + str(Y_train_val.shape))
    print('X_test: ' + str(X_test.shape))
    print('Y_test: ' + str(Y_test.shape))

    uniqueClasses = df['Type'].unique()
    print('Number of unique classes in df: ' + str(uniqueClasses.shape))
    uniqueClasses = np.sort(uniqueClasses)

    uniqueClasses_Y_train_val = np.unique(Y_train_val)
    print('Number of unique classes in Y_train_val: ' + str(uniqueClasses_Y_trai
n_val.shape))
    for i in uniqueClasses:
        print('Class ' + str(i) + ' count: ' + str((Y_train_val[:]==i).sum()))
    uniqueClasses_Y_test = np.unique(Y_test)
    print('Number of unique classes in Y_test: ' + str(uniqueClasses_Y_test.shap
e))
```

In [13]:

```
import seaborn as sns
import matplotlib.pyplot as plt

#PURPOSE: Draw heatmaps for result of grid search and find best C for validation
set.
def draw_heatmap_linear(acc, acc_desc, C_list):
    plt.figure(figsize = (2,4))
    ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=C_list, xticklabels
=[])
    ax.collections[0].colorbar.set_label("accuracy")
    ax.set(ylabel='$C$')
    plt.title(acc_desc + ' w.r.t $C$')
    sns.set_style("whitegrid", {'axes.grid' : False})
    plt.show()
```

In [14]:

```
#PURPOSE: Searches for the highest value in accuracyValidation, then uses the in
dex of the highest value
#         to find what value in the list caused this.
def bestValue(accuracyValidation, valueList):
    max_value_of_accV = np.max(accuracyValidation)
    max_index_of_accV = np.argmax(accuracyValidation)
    print('Largest value in accuracyValidation is ' + str(max_value_of_accV) + '
from index ' + str(max_index_of_accV) + '.')
    return valueList[max_index_of_accV]
```

In [15]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

def ViewConfusionMatrix(Y_test, pred):
    print('Original labels:\n' + str(Y_test))
    print('Original Labels or Y_test shape: ' + str(Y_test.shape))
    print('Predicted labels:\n' + str(pred))

    #Note that the shape of the confusion matrix is not based on the shape of th
e Y_test or pred, but instead on
    #how many unique classes were inside of these.
    print('\nTest Accuracy Score: ' + str(accuracy_score(Y_test, pred)))
    print(classification_report(Y_test, pred))
    confusionMatrix = confusion_matrix(Y_test, pred)
    print('Confusion Matrix shape: ' + str(confusionMatrix.shape))
    print(confusionMatrix) #Remove because it takes up to much space.....
```

In [107]:

```
import matplotlib.pyplot as plt; plt.rcdefaults()
import matplotlib.pyplot as plt

def displayAccuracies(stringClfName, stringDataName, acc80_20, acc50_50, acc20_8
0):

    objects = ('80%train,20%test', '50%train,50%test', '20%train,80%test')
    y_pos = np.arange(len(objects))
    performance = [acc80_20,acc50_50,acc20_80]

    plt.bar(y_pos, performance, align='center', alpha=0.5)
    plt.xticks(y_pos, objects)
    plt.ylabel('Accuracy')
    plt.title(str(stringClfName) +'\'s Test Accuracies of 3 Partitions on ' + st
r(stringDataName))
    plt.grid() #new
    plt.show()
```

In [17]:

```
#PURPOSE to print out the accuracies of 3 trials for each of the 3 partitions.
def printAccuracies(stringClgName, list_80_20, list_50_50, list20_80):

    print('Accuracy of ' + str(stringClgName) +'\'s 3 trials on (80% train, 20%
test) partition :' + str(list_80_20))
    accuracyAverage_80_20 = np.mean(list_80_20)
    print('Mean Accuracy of ' + str(stringClgName) +' on (80% train, 20% test) p
artition: ' + str(accuracyAverage_80_20))

    print('\nAccuracy of ' + str(stringClgName) + '\'s 3 trials on (50% train, 5
0% test) partition :' + str(list_50_50))
    accuracyAverage_50_50 = np.mean(list_50_50)
    print('Mean Accuracy of ' + str(stringClgName) + ' on (50% train, 50% test)
partition: ' + str(accuracyAverage_50_50))

    print('\nAccuracy of ' + str(stringClgName) + '\'s 3 trials on (20% train, 8
0% test) partition:' + str(list20_80))
    accuracyAverage_20_80 = np.mean(list20_80)
    print('Mean Accuracy of ' + str(stringClgName) + ' on (20% train, 80% test)
partition: ' + str(accuracyAverage_20_80))
```

Global Variables

CV: The number of folds that happens in the cross validation produced by GridSearchCV.

In [18]:

```
#Recommend running final test with CV=10, but during staging use CV=3.
CV = 10
```

Support Vector Machine (SVM)

Its a supervised machine learning algorithm which can be used for both classification or regression problems. But it is usually used for classification. Given 2 or more labeled classes of data, it acts as a discriminative classifier, formally defined by an optimal hyperplane that separates all the classes.

In [19]:

```
#GLOBAL VARIABLES FOR SVM
C_list = [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 1, 10, 100]
SVM_accuracyTestList_80_20 = []
SVM_accuracyTestList_50_50 = []
SVM_accuracyTestList_20_80 = []
```

In [20]:

```
from sklearn import svm

#C_list: C hyperparameter.
#cv: Number of folds when doing cross validation.
def svmTrainValidation(X_train_val, Y_train_val, C_list, CV):

    svm_classifier = svm.SVC(kernel = 'linear')

    parameters = {'C':C_list}

    SVM_clfGridSearch = GridSearchCV(svm_classifier, param_grid=parameters, cv=C
V, return_train_score=True)
    SVM_clfGridSearch.fit(X_train_val, Y_train_val)

    #accuracyTrain = clfGridSearch.cv_results_['mean_train_score']
    #accuracyValidation = clfGridSearch.cv_results_['mean_test_score']
    return SVM_clfGridSearch
```

SVM on (80% train, 20% test)

Splits the 3 shuffled datasets into 2 parts:

1. (80% of all the data points) ---> Training set + Validation Set.
2. (20% of all the data points) ---> Test set.

In [21]:

```
X1_train_val, Y1_train_val, X1_test, Y1_test = partitionData(X1, Y1, 0.8)
X2_train_val, Y2_train_val, X2_test, Y2_test = partitionData(X2, Y2, 0.8)
X3_train_val, Y3_train_val, X3_test, Y3_test = partitionData(X3, Y3, 0.8)
```

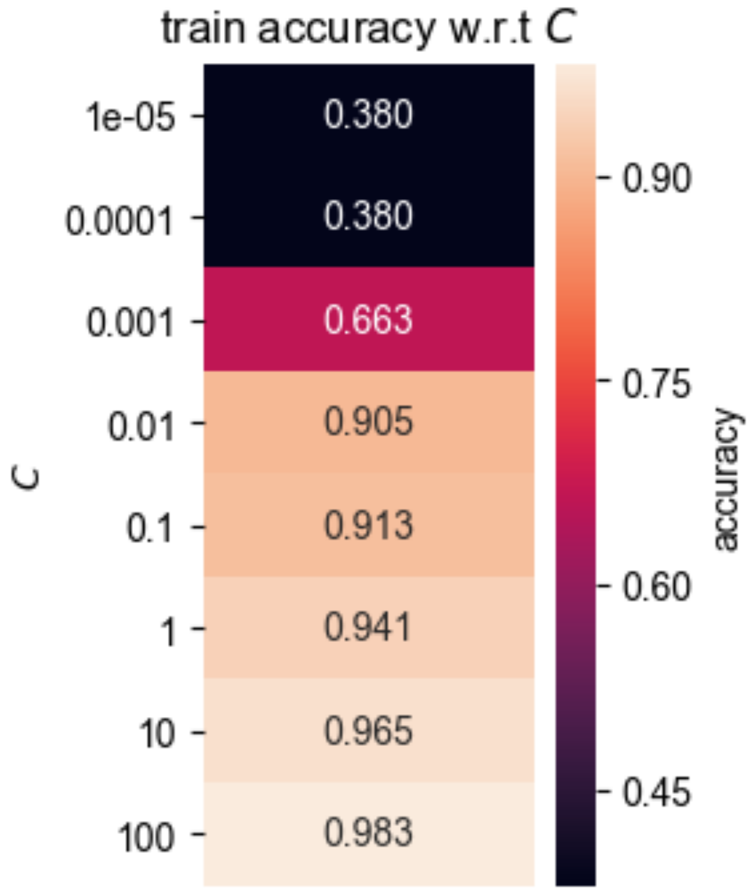
1st Run)

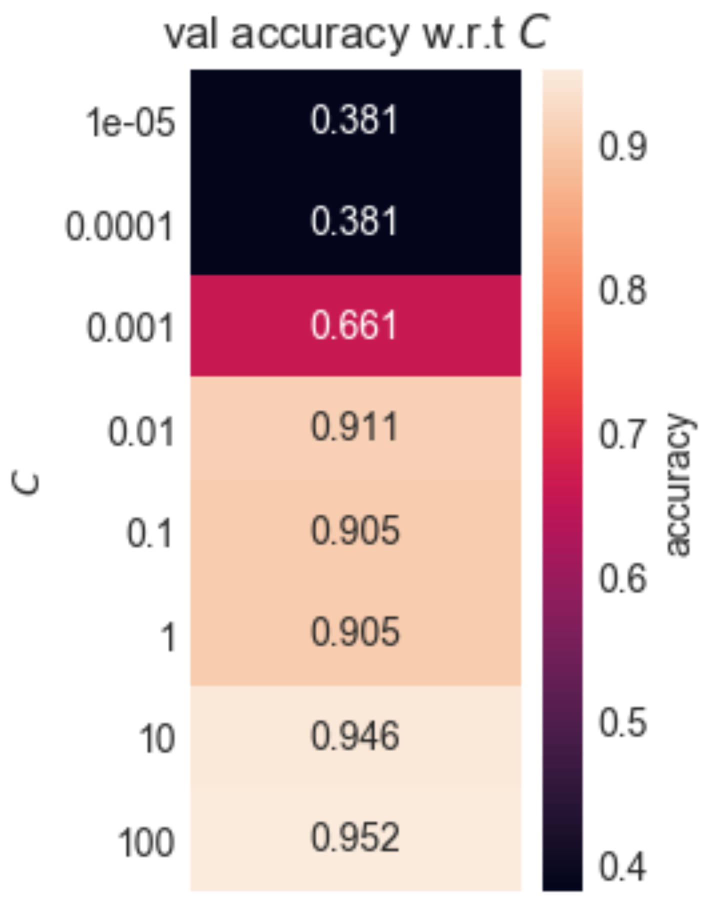
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [22]:

```
SVM_clfGridSearch = svmTrainValidation(X1_train_val, Y1_train_val, C_list, CV)
accuracyTrain = SVM_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = SVM_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', C_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', C_list)
```

```
[ 0.3801386   0.3801386   0.66269902  0.9047605   0.91270334  0.9411
5436
  0.9649479   0.98279854]
[ 0.38095238  0.38095238  0.66071429  0.91071429  0.9047619   0.9047
619
  0.94642857  0.95238095]
```





In [23]:

```
#Use the best C to calculate the test accuracy.
best_C = bestValue(accuracyValidation, C_list)
print('Best C: ' + str(best_C))

optimalClassifier = svm.SVC(kernel = 'linear', C = best_C).fit(X1_train_val, Y1_
train_val)
pred = optimalClassifier.predict(X1_test)
# correct = [(a==b) for (a,b) in zip(pred,Y1_test)]
# test_acc = sum(correct) * 1.0 / len(correct)
# print('Test Accuracy Score: ' + str(test_acc))

#accuracy(ORIGINAL_VALUES, PREDICTED_VALUES)
accuracyTest = accuracy_score(Y1_test, pred)
SVM_accuracyTestList_80_20.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.952380952381 from index 7.
Best C: 100
Test Accuracy Score: 0.976744186047
```

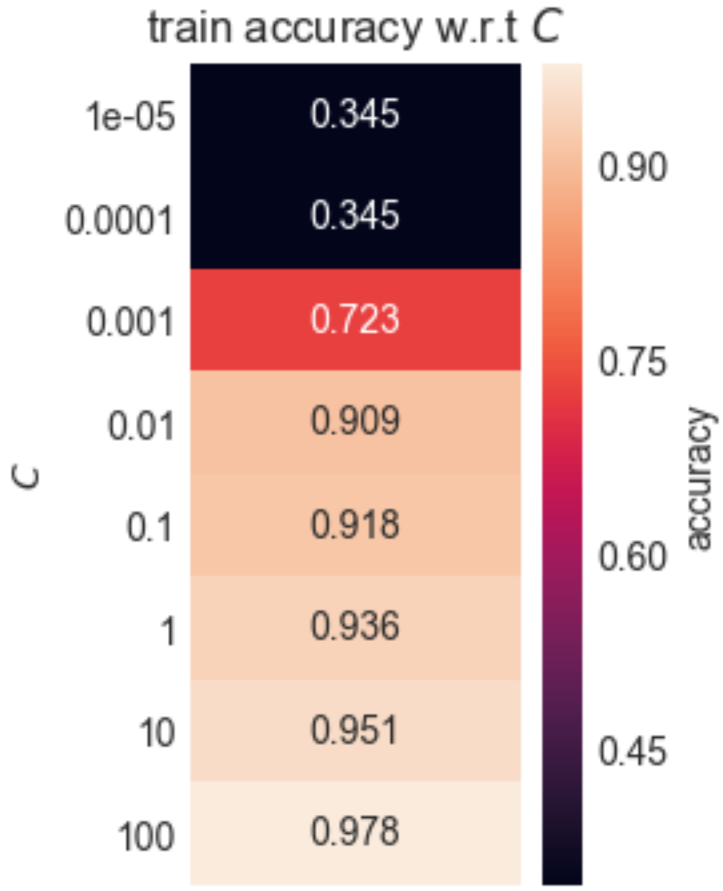
2nd Run)

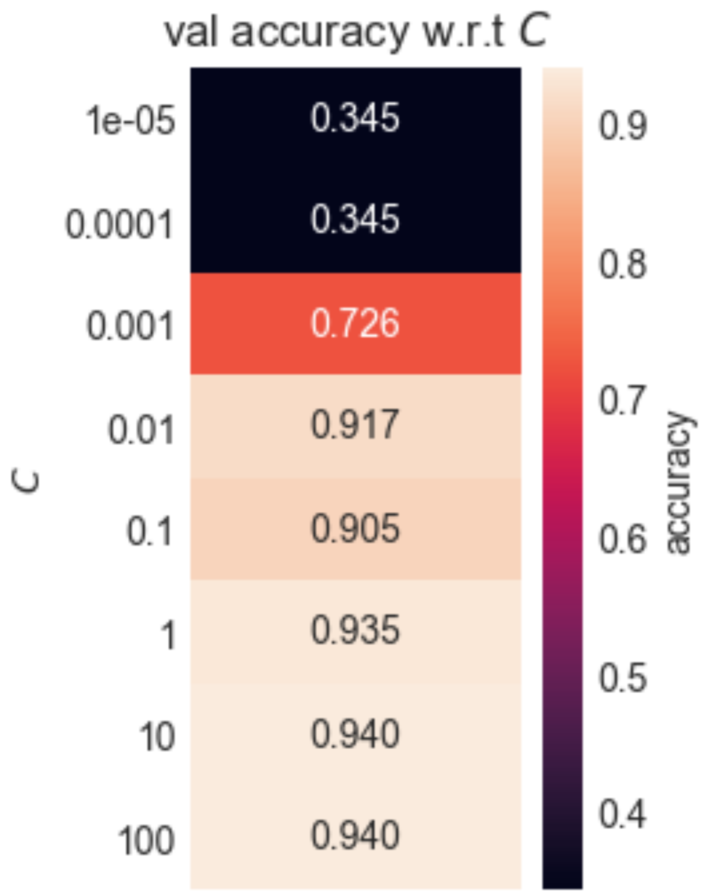
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [24]:

```
SVM_clfGridSearch = svmTrainValidation(X2_train_val, Y2_train_val, C_list, CV)
accuracyTrain = SVM_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = SVM_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', C_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', C_list)
```

```
[ 0.34524596  0.34524596  0.7234926   0.90874321  0.91799364  0.9364
8108
  0.95108204  0.97755624]
[ 0.3452381   0.3452381   0.72619048  0.91666667  0.9047619   0.9345
2381
  0.94047619  0.94047619]
```


train accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.345
0.345
0.723
0.909
0.918
0.936
0.951
0.978
0.90
0.75
0.60
0.45
accuracy


val accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.345
0.345
0.726
0.917
0.905
0.935
0.940
0.940
0.9
0.8
0.7
0.6
0.5
0.4
accuracy

In [25]:

```
best_C = bestValue(accuracyValidation, C_list)
print('Best C: ' + str(best_C))

optimalClassifier = svm.SVC(kernel = 'linear', C = best_C).fit(X2_train_val, Y2_
train_val)
pred = optimalClassifier.predict(X2_test)

accuracyTest = accuracy_score(Y2_test, pred)
SVM_accuracyTestList_80_20.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.940476190476 from index 6.
Best C: 10
Test Accuracy Score: 0.953488372093
```

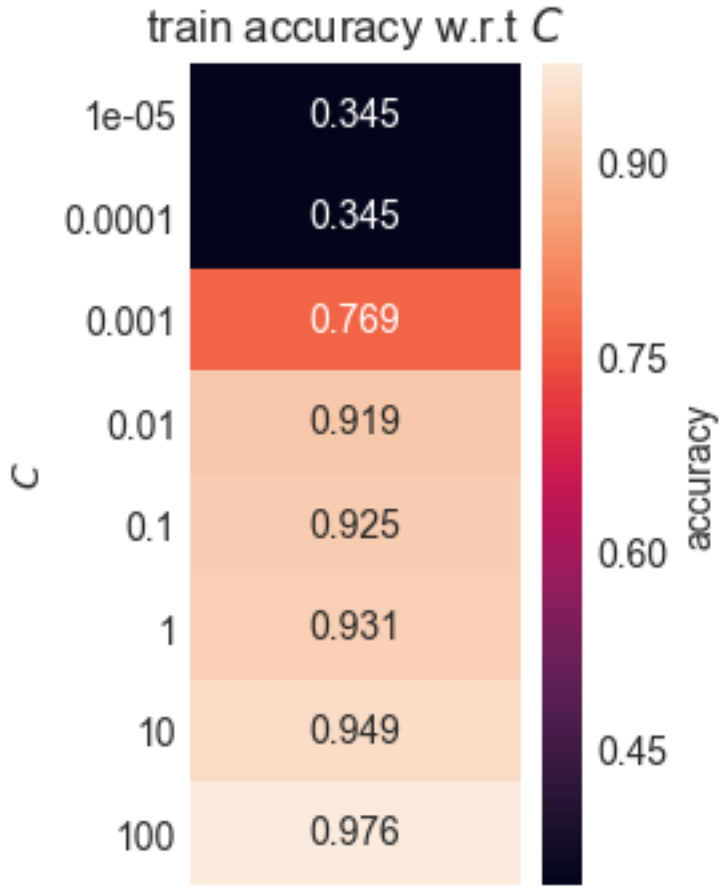
3rd Run)

Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

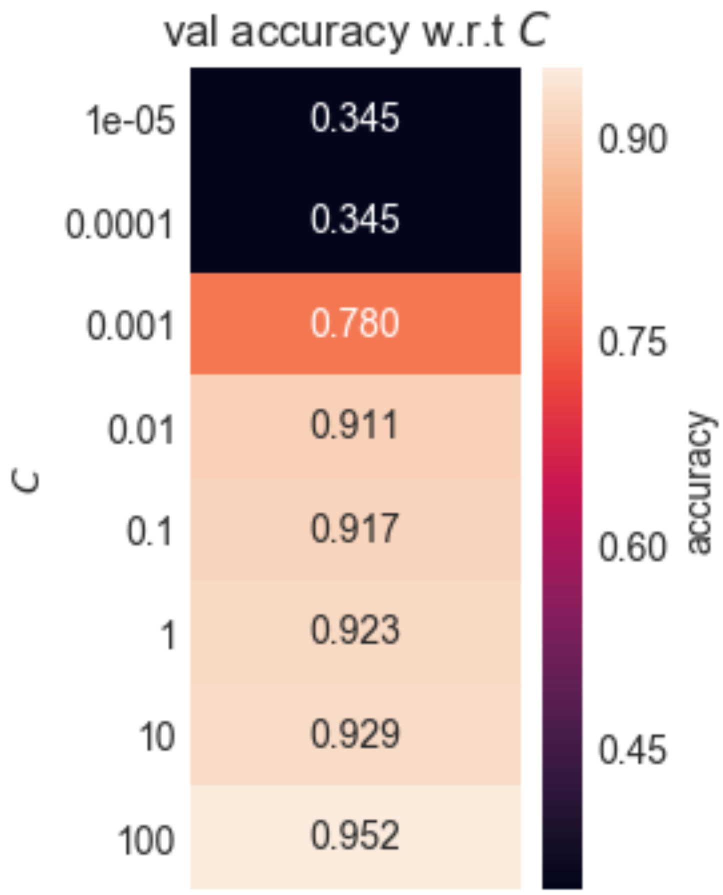
In [26]:

```
SVM_clfGridSearch = svmTrainValidation(X3_train_val, Y3_train_val, C_list, CV)
accuracyTrain = SVM_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = SVM_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', C_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', C_list)
```

```
[ 0.34524294  0.34524294  0.76912248  0.91863916  0.92458664  0.9312
0509
  0.94907809  0.97552162]
[ 0.3452381   0.3452381   0.7797619   0.91071429  0.91666667  0.9226
1905
  0.92857143  0.95238095]
```



train accuracy w.r.t C
C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
0.345
0.345
0.769
0.919
0.925
0.931
0.949
0.976
accuracy
0.90
0.75
0.60
0.45



val accuracy w.r.t C
C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
0.345
0.345
0.780
0.911
0.917
0.923
0.929
0.952
accuracy
0.90
0.75
0.60
0.45

In [27]:

```
best_C = bestValue(accuracyValidation, C_list)
print('Best C: ' + str(best_C))

optimalClassifier = svm.SVC(kernel = 'linear', C = best_C).fit(X3_train_val, Y3_
train_val)
pred = optimalClassifier.predict(X3_test)

accuracyTest = accuracy_score(Y3_test, pred)
SVM_accuracyTestList_80_20.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.952380952381 from index 7.
Best C: 100
Test Accuracy Score: 1.0
```

Mean of SVM's Test Accuracies on (80% train, 20% test)

In [28]:

```
import statistics

print('SVM_accuracyTestList:' + str(SVM_accuracyTestList_80_20))
SVM_accuracyAverage_80_20 = statistics.mean(SVM_accuracyTestList_80_20)
print('SVM_accuracyTestList mean: ' + str(SVM_accuracyAverage_80_20))
```

```
SVM_accuracyTestList:[0.97674418604651159, 0.95348837209302328, 1.0]
SVM_accuracyTestList mean: 0.976744186047
```

SVM on (50% train, 50% test)

Splits the 3 shuffled datasets into 2 parts:

1. (50% of all the data points) ---> Training set + Validation Set.
2. (50% of all the data points) ---> Test set.

In [29]:

```
X1_train_val, Y1_train_val, X1_test, Y1_test = partitionData(X1, Y1, 0.5)
X2_train_val, Y2_train_val, X2_test, Y2_test = partitionData(X2, Y2, 0.5)
X3_train_val, Y3_train_val, X3_test, Y3_test = partitionData(X3, Y3, 0.5)
```

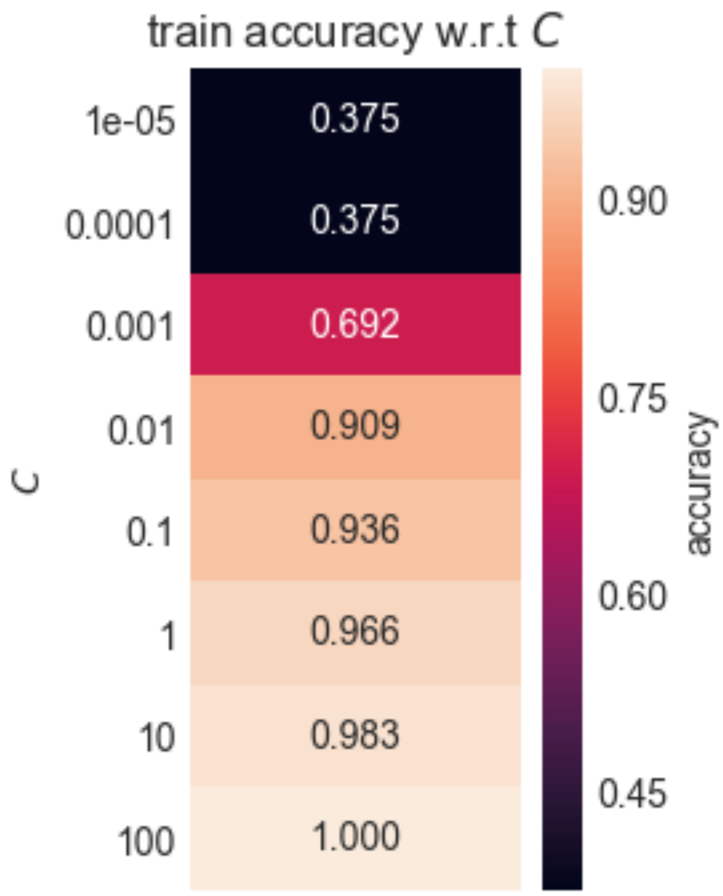
1st Run)

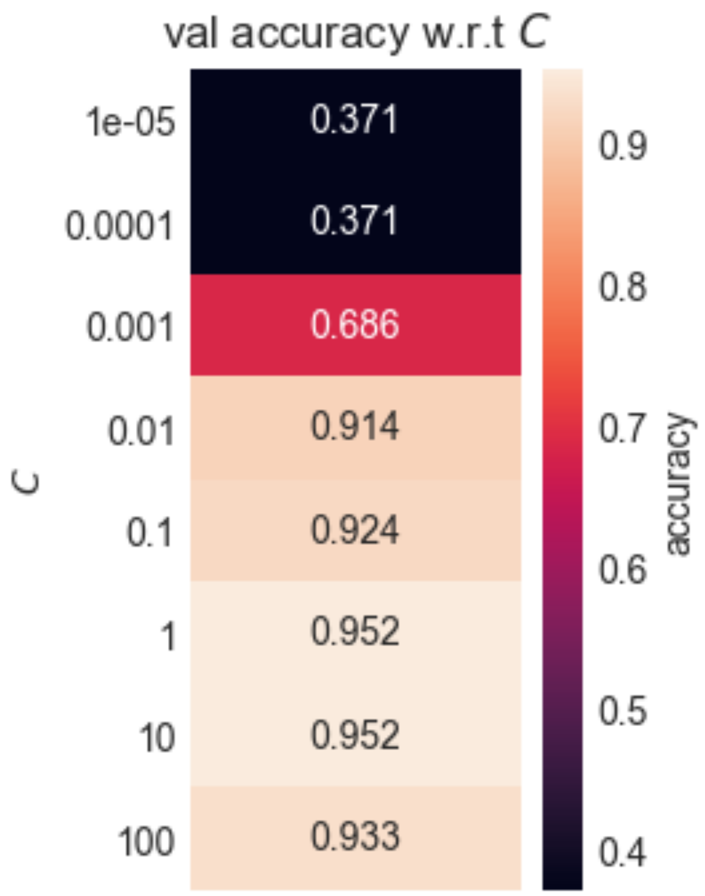
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [30]:

```
SVM_clfGridSearch = svmTrainValidation(X1_train_val, Y1_train_val, C_list, CV)
accuracyTrain = SVM_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = SVM_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', C_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', C_list)
```

```
[ 0.37549275  0.37549275  0.69215942  0.90903133  0.93649502  0.9661
0877
  0.98309907  1.        ]
[ 0.37142857  0.37142857  0.68571429  0.91428571  0.92380952  0.9523
8095
  0.95238095  0.93333333]
```





In [31]:

```
best_C = bestValue(accuracyValidation, C_list)
print('Best C: ' + str(best_C))

optimalClassifier = svm.SVC(kernel = 'linear', C = best_C).fit(X1_train_val, Y1_
train_val)
pred = optimalClassifier.predict(X1_test)

accuracyTest = accuracy_score(Y1_test, pred)
SVM_accuracyTestList_50_50.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.952380952381 from index 5.
Best C: 1
Test Accuracy Score: 0.915094339623
```

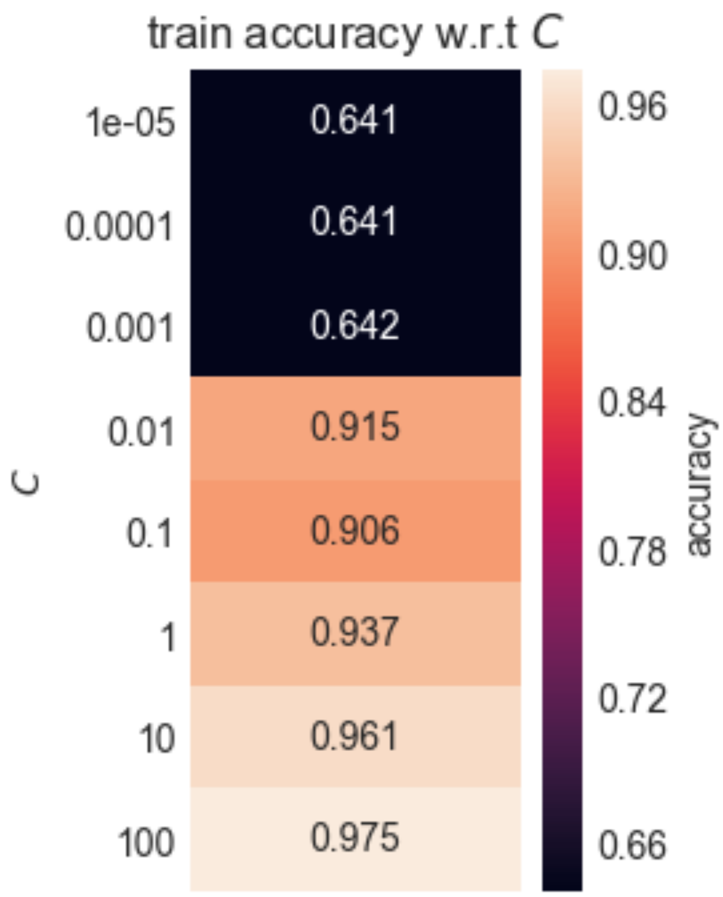
2nd Run)

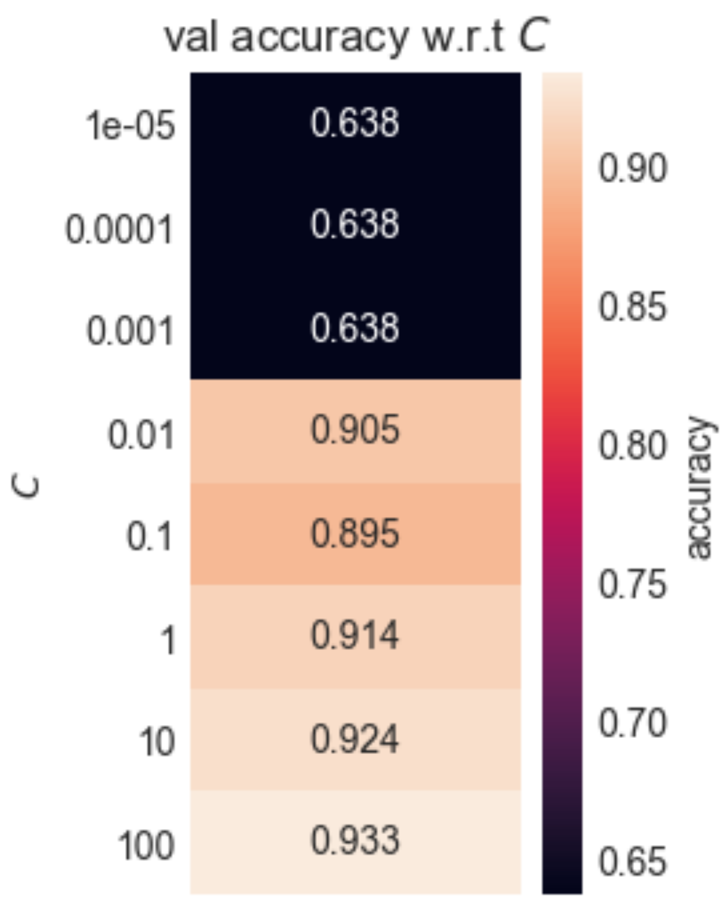
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [32]:

```
SVM_clfGridSearch = svmTrainValidation(X2_train_val, Y2_train_val, C_list, CV)
accuracyTrain = SVM_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = SVM_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', C_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', C_list)
```

```
[ 0.64119838  0.64119838  0.64224005  0.9153283   0.90583033  0.9365
0766
  0.9609014   0.97453171]
[ 0.63809524  0.63809524  0.63809524  0.9047619   0.8952381   0.9142
8571
  0.92380952  0.93333333]
```





In [33]:

```
best_C = bestValue(accuracyValidation, C_list)
print('Best C: ' + str(best_C))

optimalClassifier = svm.SVC(kernel = 'linear', C = best_C).fit(X2_train_val, Y2_
train_val)
pred = optimalClassifier.predict(X2_test)

accuracyTest = accuracy_score(Y2_test, pred)
SVM_accuracyTestList_50_50.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.933333333333 from index 7.
Best C: 100
Test Accuracy Score: 0.933962264151
```

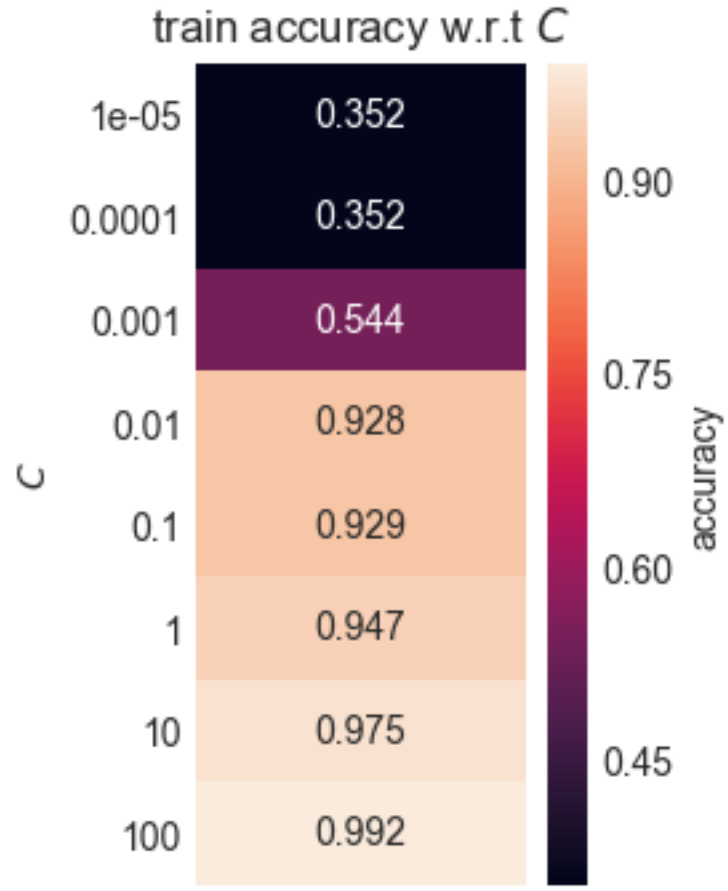
3rd Run)

Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

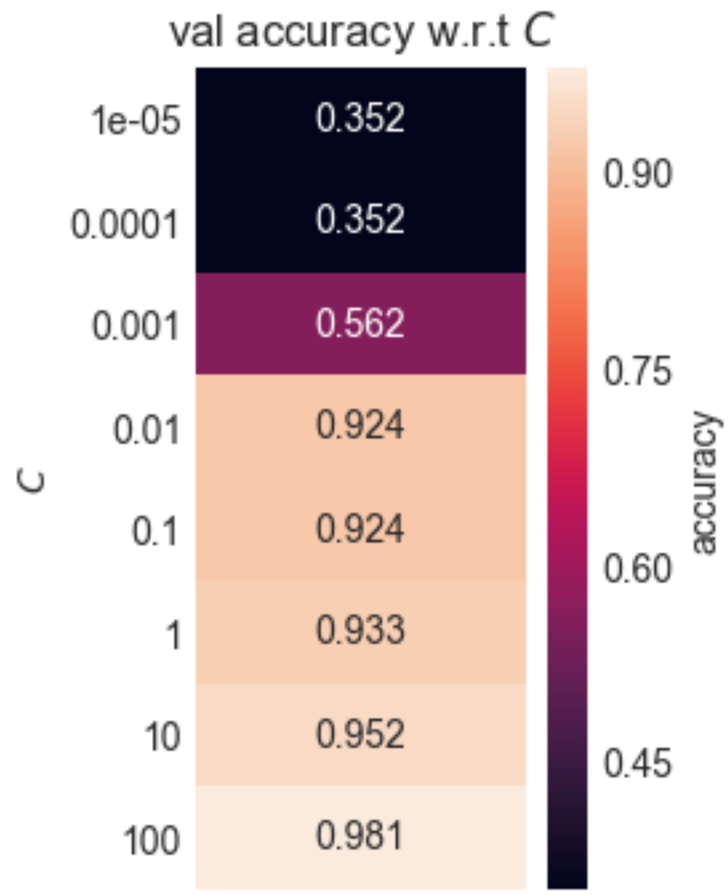
In [34]:

```
SVM_clfGridSearch = svmTrainValidation(X3_train_val, Y3_train_val, C_list, CV)
accuracyTrain = SVM_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = SVM_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', C_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', C_list)
```

```
[ 0.35238806  0.35238806  0.54381386  0.92799629  0.92903843  0.9470
5919
  0.97461058  0.99151057]
[ 0.35238095  0.35238095  0.56190476  0.92380952  0.92380952  0.9333
3333
  0.95238095  0.98095238]
```



train accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.352
0.352
0.544
0.928
0.929
0.947
0.975
0.992
0.90
0.75
0.60
0.45
accuracy



val accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.352
0.352
0.562
0.924
0.924
0.933
0.952
0.981
0.90
0.75
0.60
0.45
accuracy

In [35]:

```
best_C = bestValue(accuracyValidation, C_list)
print('Best C: ' + str(best_C))

optimalClassifier = svm.SVC(kernel = 'linear', C = best_C).fit(X3_train_val, Y3_
train_val)
pred = optimalClassifier.predict(X3_test)

accuracyTest = accuracy_score(Y3_test, pred)
SVM_accuracyTestList_50_50.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.980952380952 from index 7.
Best C: 100
Test Accuracy Score: 0.933962264151
```

Mean of SVM's Test Accuracies on (50% train, 50% test)

In [36]:

```
print('SVM_accuracyTestList:' + str(SVM_accuracyTestList_50_50))
SVM_accuracyAverage_50_50 = statistics.mean(SVM_accuracyTestList_50_50)
print('SVM_accuracyTestList mean: ' + str(SVM_accuracyAverage_50_50))
```

```
SVM_accuracyTestList:[0.9150943396226415 3, 0.9339622641509434 1, 0.93
396226415094341]
SVM_accuracyTestList mean: 0.927672955975
```

SVM on (20% train, 80% test)

Splits the 3 shuffled datasets into 2 parts:

1. (20% of all the data points) ---> Training set + Validation Set.
2. (80% of all the data points) ---> Test set.

In [37]:

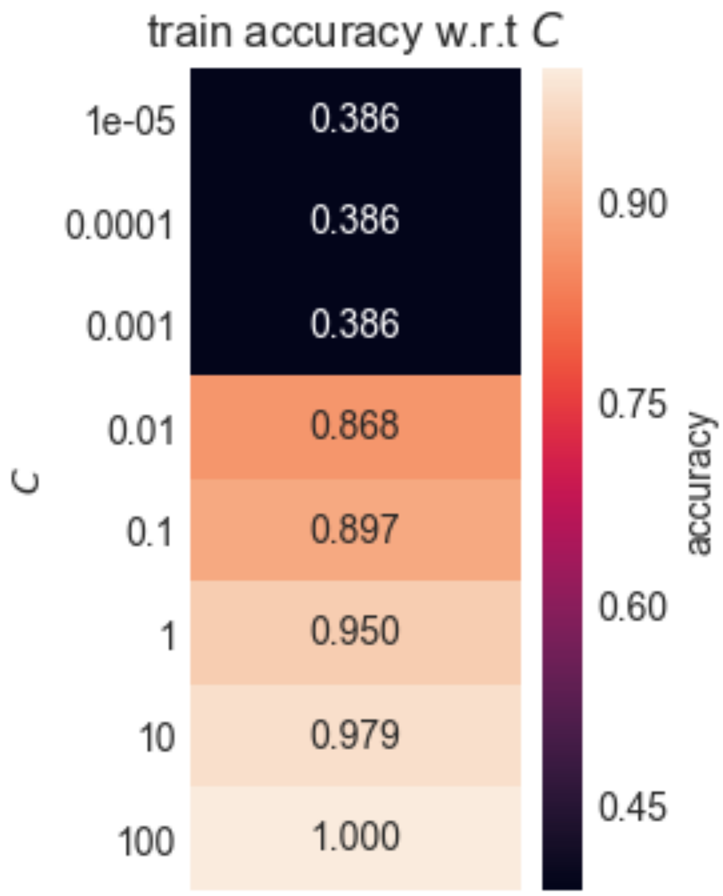
```
X1_train_val, Y1_train_val, X1_test, Y1_test = partitionData(X1, Y1, 0.2)
X2_train_val, Y2_train_val, X2_test, Y2_test = partitionData(X2, Y2, 0.2)
X3_train_val, Y3_train_val, X3_test, Y3_test = partitionData(X3, Y3, 0.5)
```

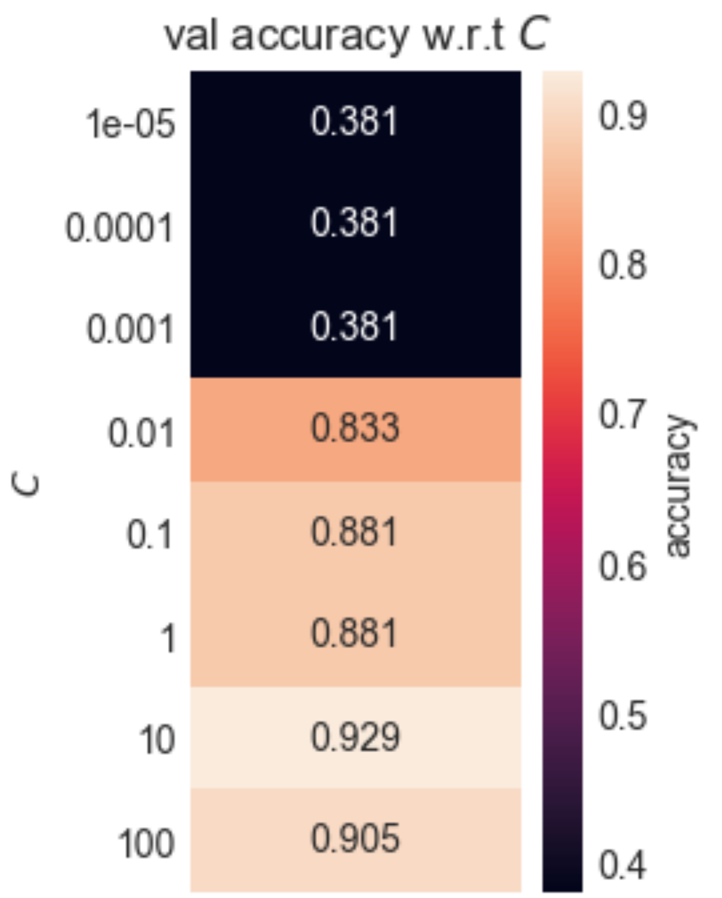
1st Run)

First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [38]:

```
SVM_clfGridSearch = svmTrainValidation(X1_train_val, Y1_train_val, C_list, CV)
accuracyTrain = SVM_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = SVM_clfGridSearch.cv_results_['mean_test_score']

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', C_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', C_list)
```





In [39]:

```
best_C = bestValue(accuracyValidation, C_list)
print('Best C: ' + str(best_C))

optimalClassifier = svm.SVC(kernel = 'linear', C = best_C).fit(X1_train_val, Y1_
train_val)
pred = optimalClassifier.predict(X1_test)

accuracyTest = accuracy_score(Y1_test, pred)
SVM_accuracyTestList_20_80.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

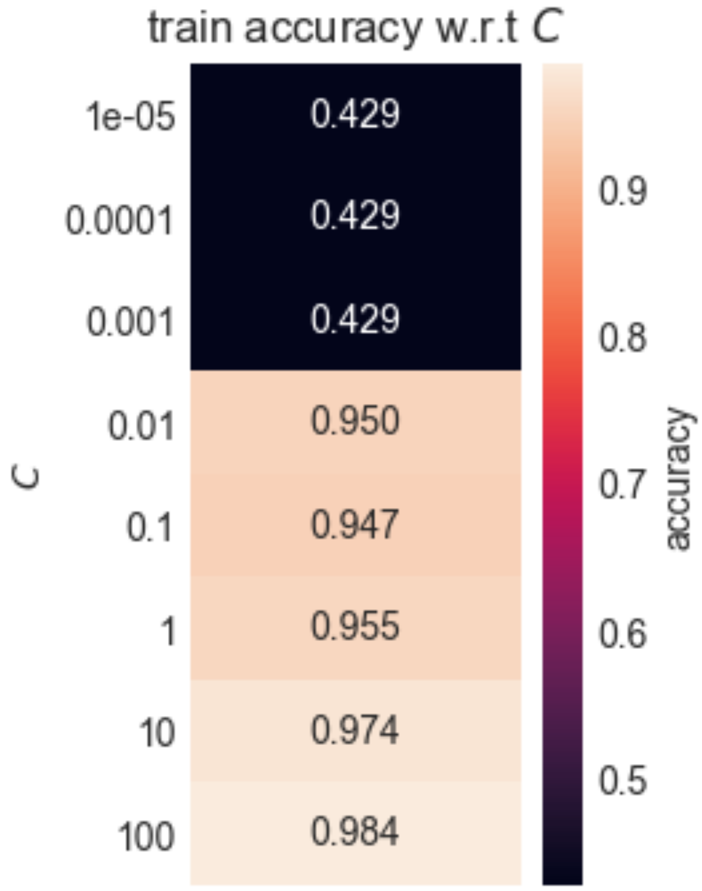
```
Largest value in accuracyValidation is 0.928571428571 from index 6.
Best C: 10
Test Accuracy Score: 0.94674556213
```

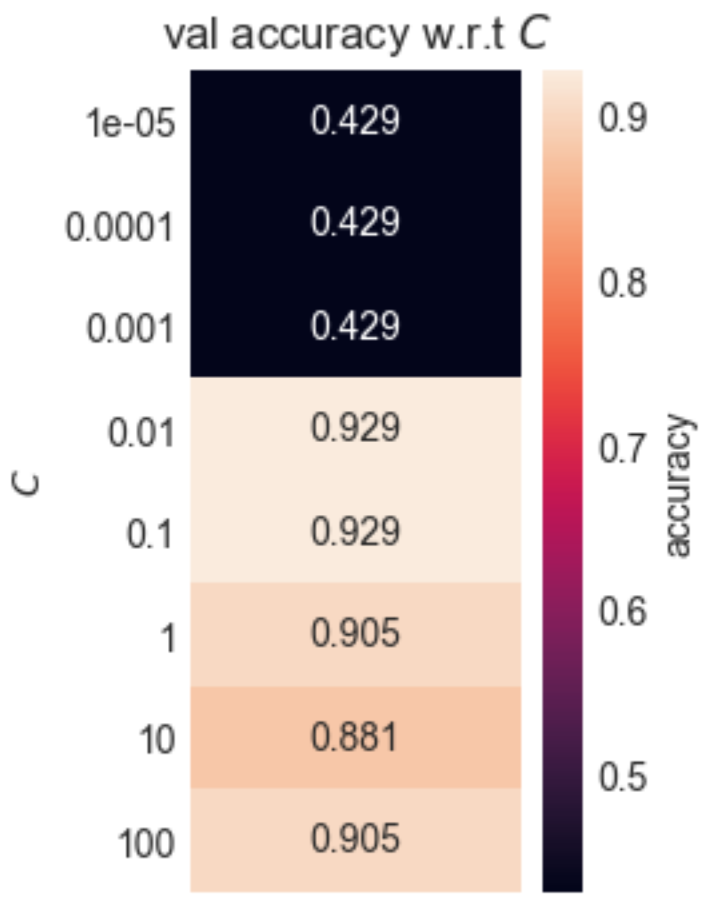
2nd Run)

Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [40]:

```
SVM_clfGridSearch = svmTrainValidation(X2_train_val, Y2_train_val, C_list, CV)
accuracyTrain = SVM_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = SVM_clfGridSearch.cv_results_['mean_test_score']

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', C_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', C_list)
```


train accuracy w.r.t C
C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
0.429
0.429
0.429
0.950
0.947
0.955
0.974
0.984
0.9
0.8
0.7
0.6
0.5
accuracy


val accuracy w.r.t C
C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
0.429
0.429
0.429
0.929
0.929
0.905
0.881
0.905
0.9
0.8
0.7
0.6
0.5
accuracy

In [41]:

```
best_C = bestValue(accuracyValidation, C_list)
print('Best C: ' + str(best_C))

optimalClassifier = svm.SVC(kernel = 'linear', C = best_C).fit(X2_train_val, Y2_
train_val)
pred = optimalClassifier.predict(X2_test)

accuracyTest = accuracy_score(Y2_test, pred)
SVM_accuracyTestList_20_80.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.928571428571 from index 3.
Best C: 0.01
Test Accuracy Score: 0.869822485207
```

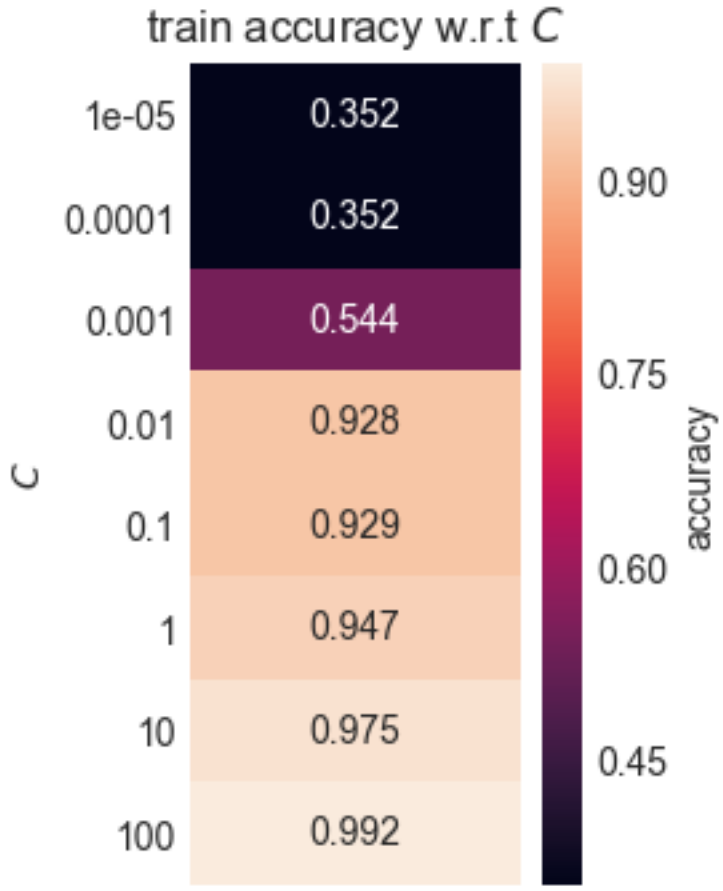
3rd Run)

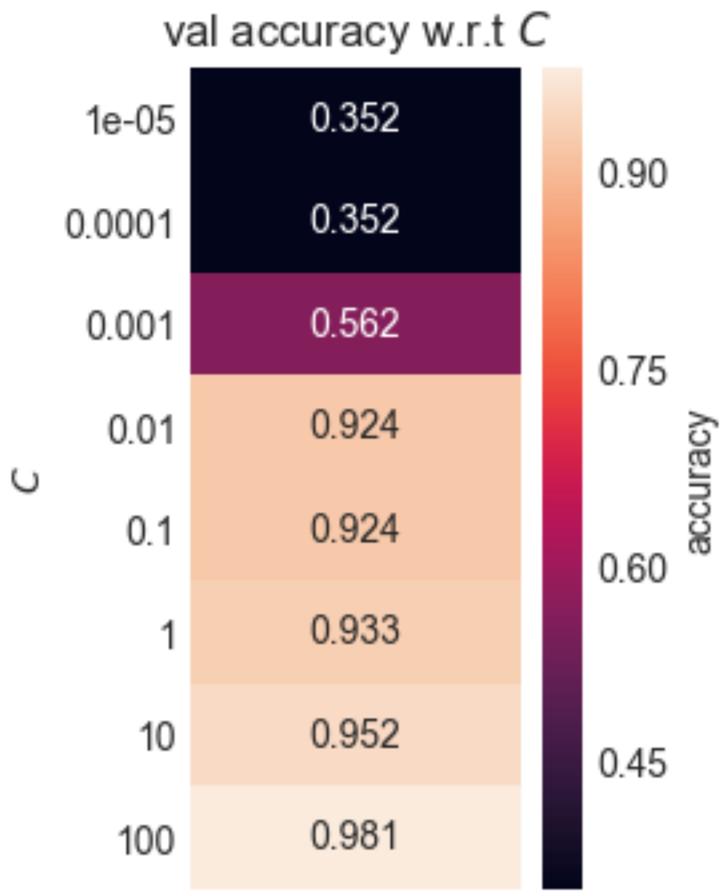
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [42]:

```
SVM_clfGridSearch = svmTrainValidation(X3_train_val, Y3_train_val, C_list, CV)
accuracyTrain = SVM_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = SVM_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', C_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', C_list)
```

```
[ 0.35238806  0.35238806  0.54381386  0.92799629  0.92903843  0.9470
5919
  0.97461058  0.99151057]
[ 0.35238095  0.35238095  0.56190476  0.92380952  0.92380952  0.9333
3333
  0.95238095  0.98095238]
```


train accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.352
0.352
0.544
0.928
0.929
0.947
0.975
0.992
0.90
0.75
0.60
0.45
accuracy


val accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.352
0.352
0.562
0.924
0.924
0.933
0.952
0.981
0.90
0.75
0.60
0.45
accuracy

In [43]:

```
best_C = bestValue(accuracyValidation, C_list)
print('Best C: ' + str(best_C))

optimalClassifier = svm.SVC(kernel = 'linear', C = best_C).fit(X3_train_val, Y3_
train_val)
pred = optimalClassifier.predict(X3_test)

accuracyTest = accuracy_score(Y3_test, pred)
SVM_accuracyTestList_20_80.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.980952380952 from index 7.
Best C: 100
Test Accuracy Score: 0.933962264151
```

Mean of SVM's Test Accuracies on (20% train, 80% test)

In [44]:

```
print('SVM_accuracyTestList:' + str(SVM_accuracyTestList_20_80))
SVM_accuracyAverage_20_80 = statistics.mean(SVM_accuracyTestList_20_80)
print('SVM_accuracyTestList mean: ' + str(SVM_accuracyAverage_20_80))
```

```
SVM_accuracyTestList:[0.9467455621301746, 0.8698224852071059, 0.93
39622641509434l]
SVM_accuracyTestList mean: 0.916843437163
```

Results of SVM

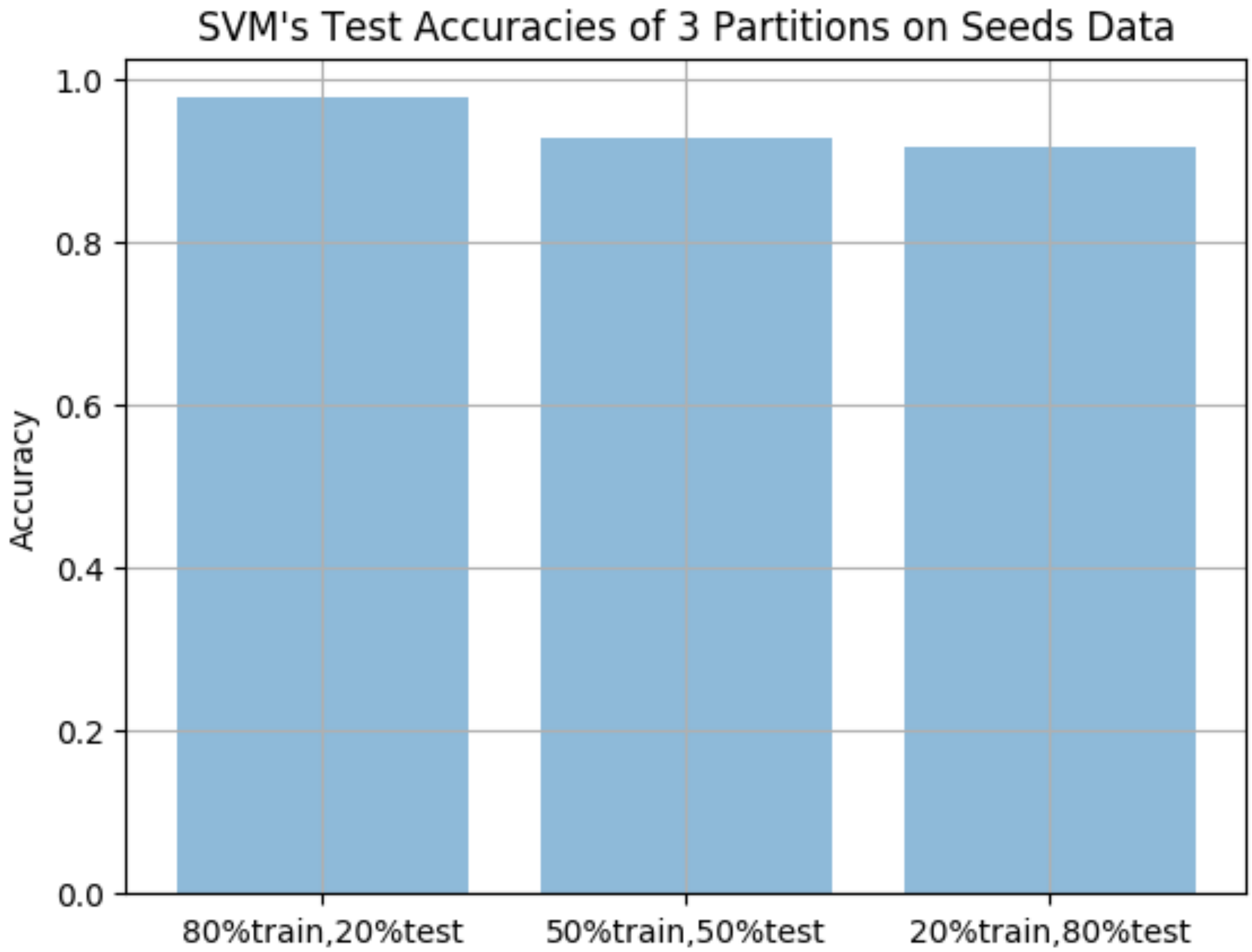
In [45]:

```
print('SVM_accuracyTestList (80% train, 20% test) partition mean: ' + str(SVM_ac
curacyAverage_80_20))
print('SVM_accuracyTestList (50% train, 50% test) partition mean: ' + str(SVM_ac
curacyAverage_50_50))
print('SVM_accuracyTestList (20% train, 80% test) partition mean: ' + str(SVM_ac
curacyAverage_20_80))
```

```
SVM_accuracyTestList (80% train, 20% test) partition mean: 0.9767441
86047
SVM_accuracyTestList (50% train, 50% test) partition mean: 0.9276729
55975
SVM_accuracyTestList (20% train, 80% test) partition mean: 0.9168434
37163
```

In [108]:

```
displayAccuracies('SVM', 'Seeds Data',  SVM_accuracyAverage_80_20, SVM_accuracyA
verage_50_50, SVM_accuracyAverage_20_80)
printAccuracies('SVM', SVM_accuracyTestList_80_20, SVM_accuracyTestList_50_50, S
VM_accuracyTestList_20_80)
```




```
Accuracy of SVM's 3 trials on (80% train, 20% test) partition :[0.97
674418604651159, 0.95348837209302328, 1.0]
Mean Accuracy of SVM on (80% train, 20% test) partition: 0.976744186
047

Accuracy of SVM's 3 trials on (50% train, 50% test) partition :[0.91
509433962264153, 0.93396226415094341, 0.93396226415094341]
Mean Accuracy of SVM on (50% train, 50% test) partition: 0.927672955
975

Accuracy of SVM's 3 trials on (20% train, 80% test) partition:[0.946
74556213017746, 0.86982248520710059, 0.93396226415094341]
Mean Accuracy of SVM on (20% train, 80% test) partition: 0.916843437
163
```

Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

In [47]:

```
#GLOBAL VARIABLES FOR Decision Tree
#D_list = np.asarray([1,2,3,4,5])
D_list = [1,2,3,4,5]
DT_accuracyTestList_80_20 = []
DT_accuracyTestList_50_50 = []
DT_accuracyTestList_20_80 = []
```

In [48]:

```
from sklearn import tree

def decisionTreeTrainValidation(X_train_val, Y_train_val, D_list, CV):

    DT_classifier = tree.DecisionTreeClassifier(criterion='entropy')

    parameters = {'max_depth': D_list}

    DT_clfGridSearch = GridSearchCV(DT_classifier, param_grid=parameters, cv=CV,
return_train_score=True)
    DT_clfGridSearch.fit(X_train_val, Y_train_val)

    return DT_clfGridSearch
```

Decision Tree on (80% train, 20% test)

Splits the 3 shuffled datasets into 2 parts:

1. (80% of all the data points) ---> Training set + Validation Set.
2. (20% of all the data points) ---> Test set.

In [49]:

```
X1_train_val, Y1_train_val, X1_test, Y1_test = partitionData(X1, Y1, 0.8)
X2_train_val, Y2_train_val, X2_test, Y2_test = partitionData(X2, Y2, 0.8)
X3_train_val, Y3_train_val, X3_test, Y3_test = partitionData(X3, Y3, 0.8)
```

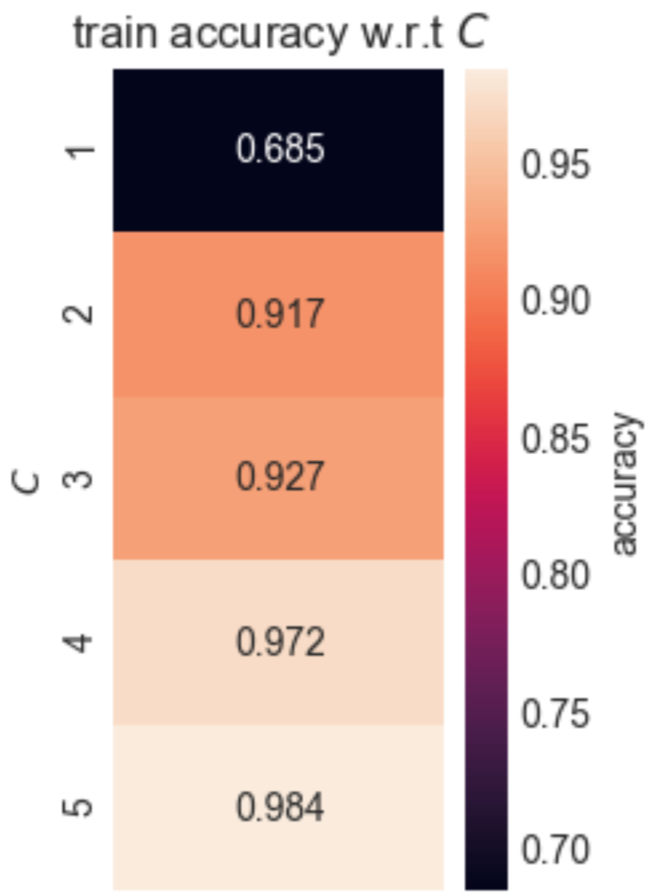
1st Run)

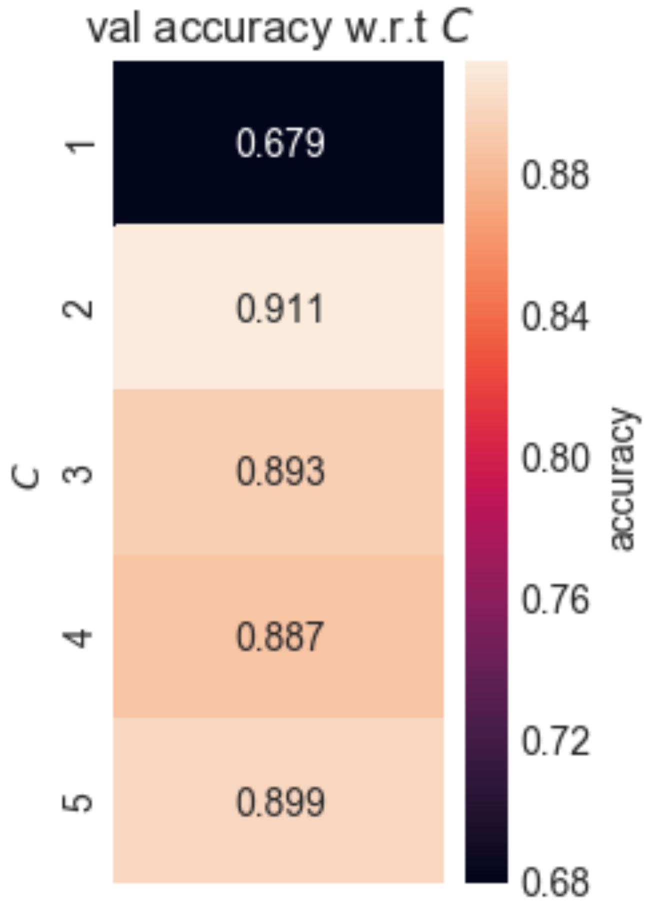
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [50]:

```
DT_clfGridSearch = decisionTreeTrainValidation(X1_train_val, Y1_train_val, D_lis
t, CV)
accuracyTrain = DT_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = DT_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)         #This is what shows up in the heat maps.
print(accuracyValidation)    #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', D_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', D_list)
```

```
[ 0.68452312  0.91666808  0.92657147  0.97222848  0.98412746]
[ 0.67857143  0.91071429  0.89285714  0.88690476  0.89880952]
```





In [51]:

```
best_D = bestValue(accuracyValidation, D_list)
print('Best D: ' + str(best_D))

optimalClassifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=b
est_D).fit(X1_train_val, Y1_train_val)
pred = optimalClassifier.predict(X1_test)

accuracyTest = accuracy_score(Y1_test, pred)
DT_accuracyTestList_80_20.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.910714285714 from index 1.
Best D: 2
Test Accuracy Score: 0.93023255814
```

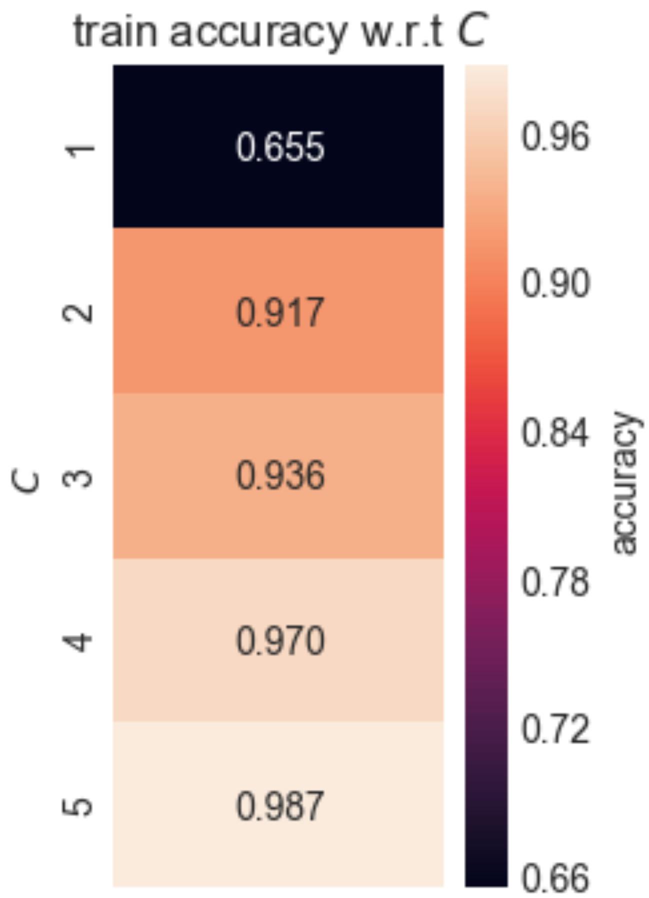
2nd Run)

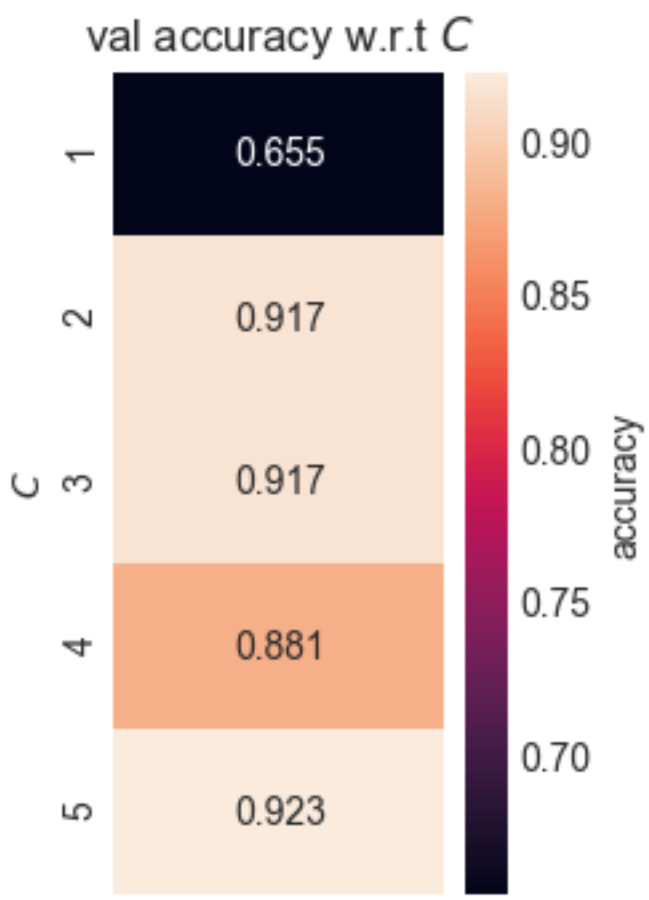
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [52]:

```
DT_clfGridSearch = decisionTreeTrainValidation(X2_train_val, Y2_train_val, D_lis
t, CV)
accuracyTrain = DT_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = DT_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', D_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', D_list)
```

```
[ 0.65475421  0.9166689   0.93583626  0.9702623   0.9868065 ]
[ 0.6547619   0.91666667  0.91666667  0.88095238  0.92261905]
```





In [53]:

```
#Use the best C to calculate the test accuracy.
best_D = bestValue(accuracyValidation, D_list)
print('Best D: ' + str(best_D))

optimalClassifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=b
est_D).fit(X2_train_val, Y2_train_val)
pred = optimalClassifier.predict(X2_test)

accuracyTest = accuracy_score(Y2_test, pred)
DT_accuracyTestList_80_20.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.922619047619 from index 4.
Best D: 5
Test Accuracy Score: 0.93023255814
```

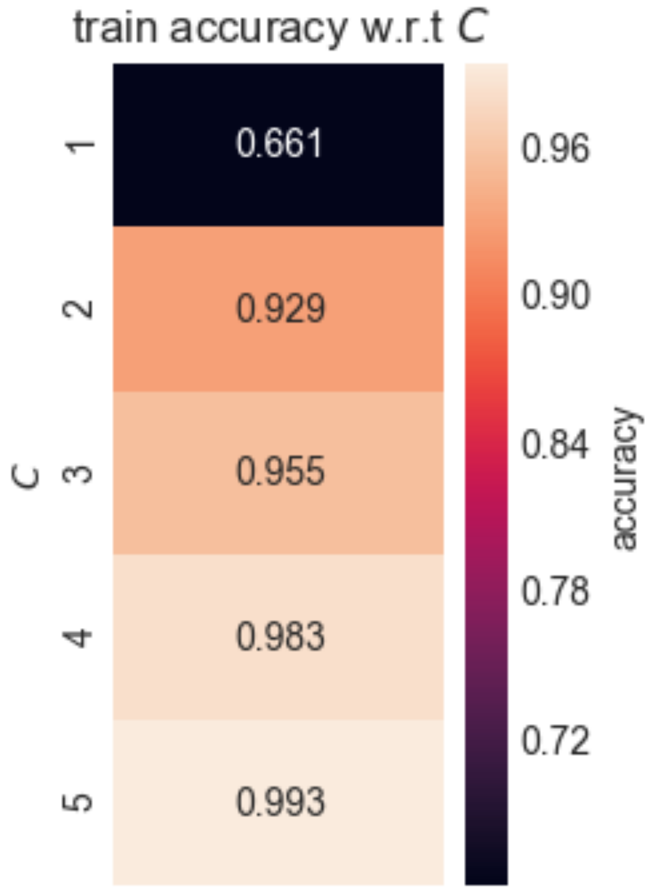
3rd Run)

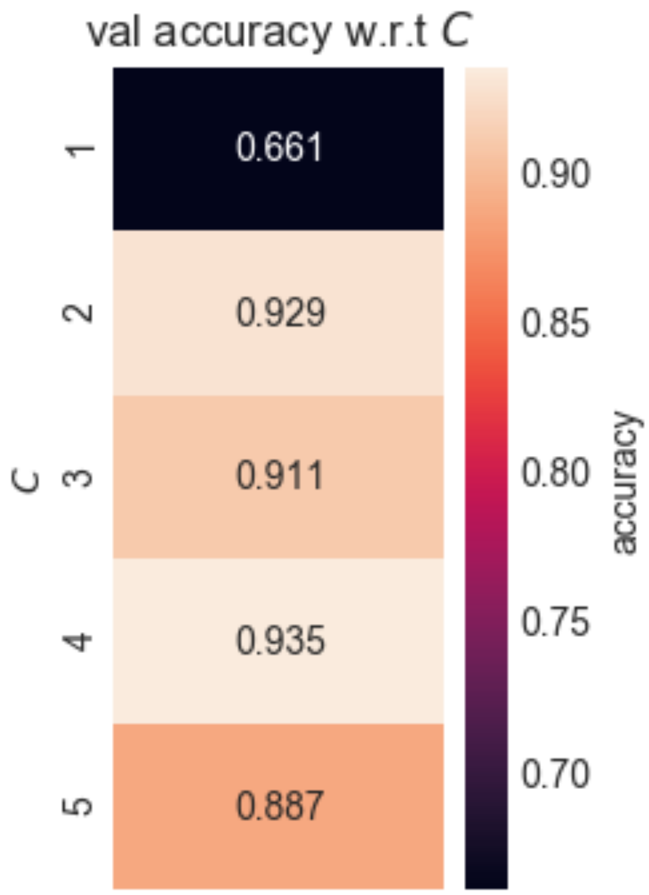
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [54]:

```
DT_clfGridSearch = decisionTreeTrainValidation(X3_train_val, Y3_train_val, D_lis
t, CV)
accuracyTrain = DT_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = DT_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', D_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', D_list)
```

```
[ 0.66072215  0.92857328  0.95499513  0.98279366  0.99337278]
[ 0.66071429  0.92857143  0.91071429  0.93452381  0.88690476]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.661
0.929
0.955
0.983
0.993
0.96
0.90
0.84
0.78
0.72
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.661
0.929
0.911
0.935
0.887
0.90
0.85
0.80
0.75
0.70
accuracy

In [55]:

```
#Use the best C to calculate the test accuracy.
best_D = bestValue(accuracyValidation, D_list)
print('Best D: ' + str(best_D))

optimalClassifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=b
est_D).fit(X3_train_val, Y3_train_val)
pred = optimalClassifier.predict(X3_test)

accuracyTest = accuracy_score(Y3_test, pred)
DT_accuracyTestList_80_20.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.934523809524 from index 3.
Best D: 4
Test Accuracy Score: 0.906976744186
```

Mean of DT's Test Accuracies on (80% train, 20% test)

In [56]:

```
print('DT_accuracyTestList:' + str(DT_accuracyTestList_80_20))
DT_accuracyAverage_80_20 = statistics.mean(DT_accuracyTestList_80_20)
print('DT_accuracyTestList mean: ' + str(DT_accuracyAverage_80_20))
```

```
DT_accuracyTestList:[0.93023255813953487, 0.93023255813953487, 0.906
97674418604646]
DT_accuracyTestList mean: 0.922480620155
```

Decision Tree on (50% train, 50% test)

Splits the 3 shuffled datasets into 2 parts:

1. (50% of all the data points) ---> Training set + Validation Set.
2. (50% of all the data points) ---> Test set.

In [57]:

```
X1_train_val, Y1_train_val, X1_test, Y1_test = partitionData(X1, Y1, 0.5)
X2_train_val, Y2_train_val, X2_test, Y2_test = partitionData(X2, Y2, 0.5)
X3_train_val, Y3_train_val, X3_test, Y3_test = partitionData(X3, Y3, 0.5)
```

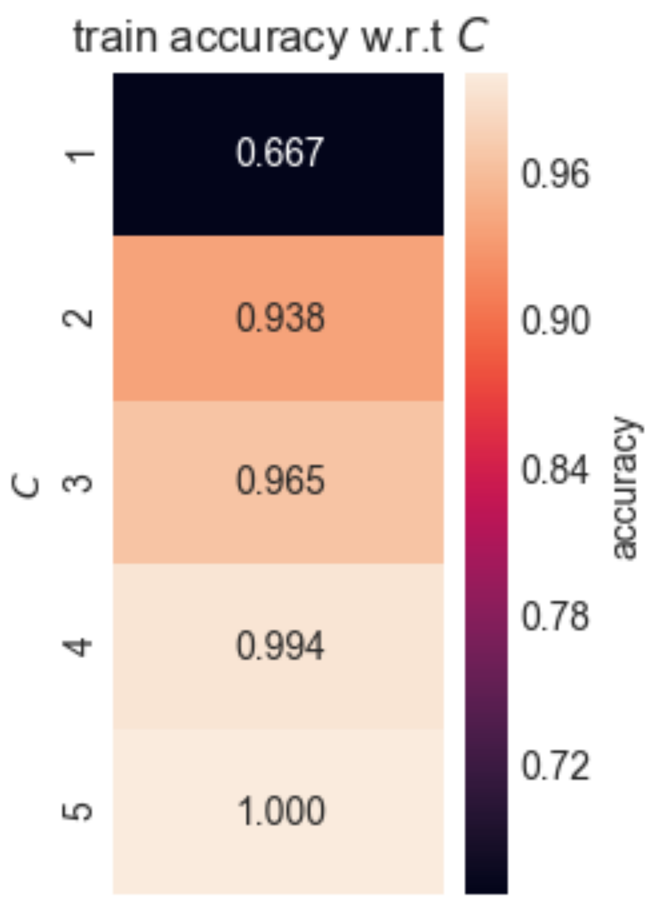
1st Run)

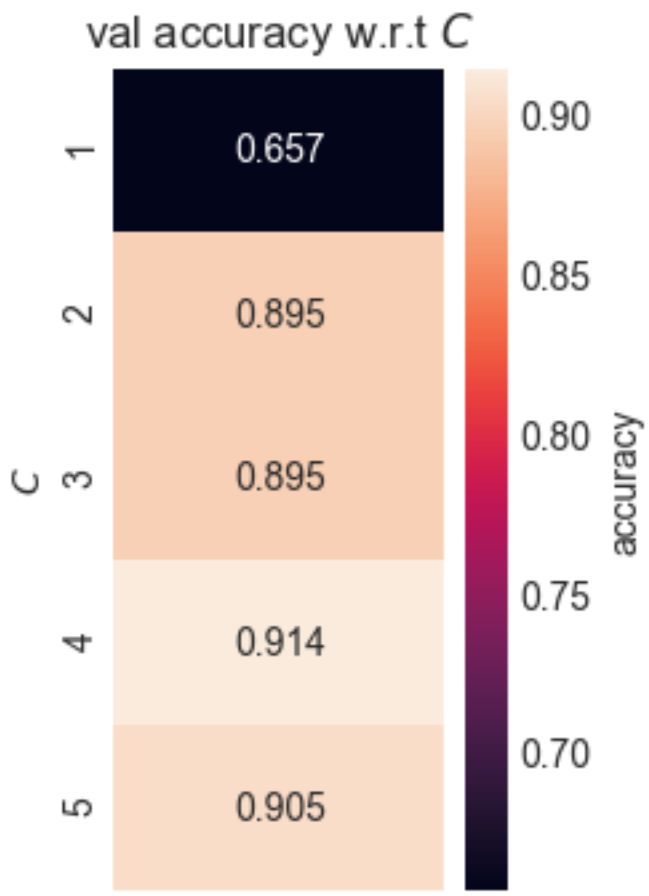
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [58]:

```
DT_clfGridSearch = decisionTreeTrainValidation(X1_train_val, Y1_train_val, D_lis
t, CV)
accuracyTrain = DT_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = DT_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', D_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', D_list)
```

```
[ 0.66668184  0.93760342  0.96520056  0.9936275   1.        ]
[ 0.65714286  0.8952381   0.8952381   0.91428571  0.9047619 ]
```





In [59]:

```
best_D = bestValue(accuracyValidation, D_list)
print('Best D: ' + str(best_D))

optimalClassifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=b
est_D).fit(X1_train_val, Y1_train_val)
pred = optimalClassifier.predict(X1_test)

accuracyTest = accuracy_score(Y1_test, pred)
DT_accuracyTestList_50_50.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.914285714286 from index 3.
Best D: 4
Test Accuracy Score: 0.905660377358
```

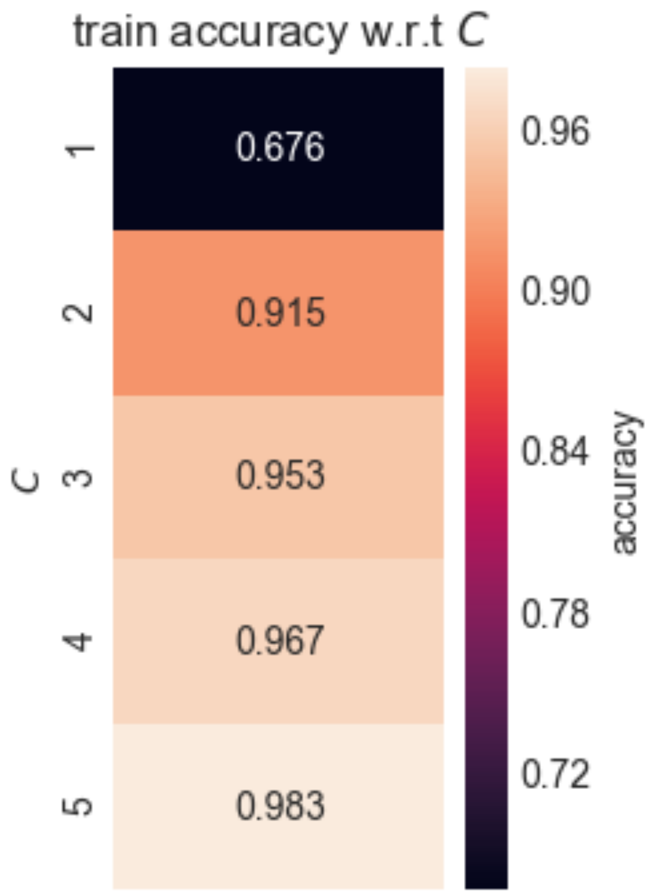
2nd Run)

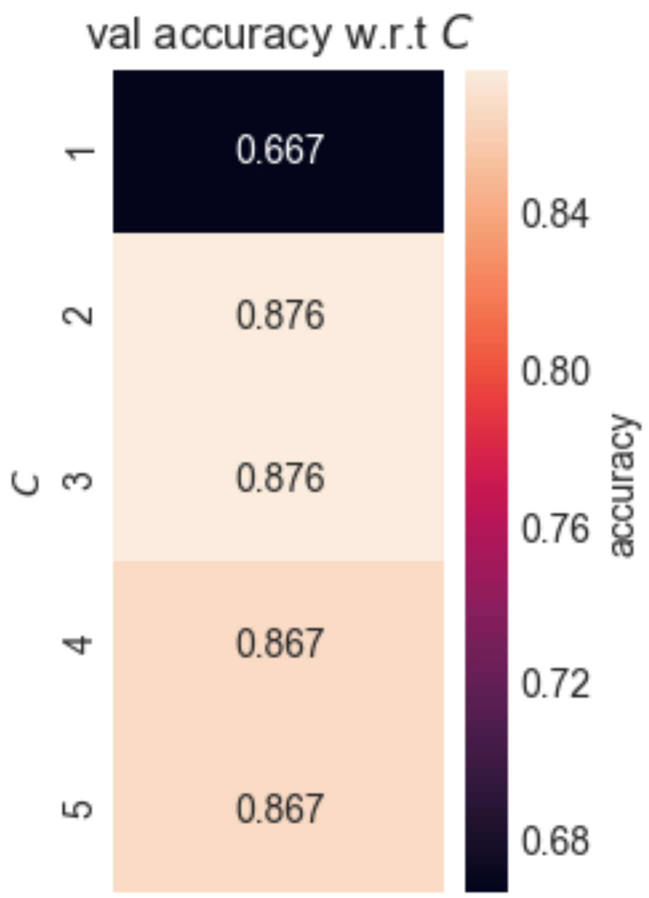
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [60]:

```
DT_clfGridSearch = decisionTreeTrainValidation(X2_train_val, Y2_train_val, D_lis
t, CV)
accuracyTrain = DT_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = DT_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', D_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', D_list)
```

```
[ 0.67616464  0.91530614  0.95333019  0.96722932  0.9830209 ]
[ 0.66666667  0.87619048  0.87619048  0.86666667  0.86666667]
```





In [61]:

```
best_D = bestValue(accuracyValidation, D_list)
print('Best D: ' + str(best_D))

optimalClassifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=b
est_D).fit(X2_train_val, Y2_train_val)
pred = optimalClassifier.predict(X2_test)

accuracyTest = accuracy_score(Y2_test, pred)
DT_accuracyTestList_50_50.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.87619047619 from index 1.
Best D: 2
Test Accuracy Score: 0.924528301887
```

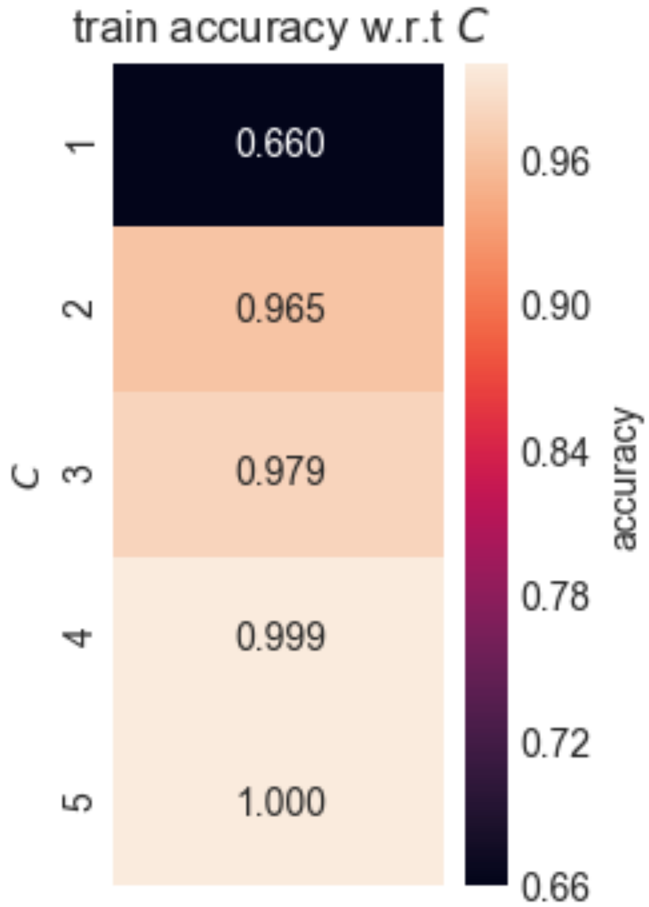
3rd Run)

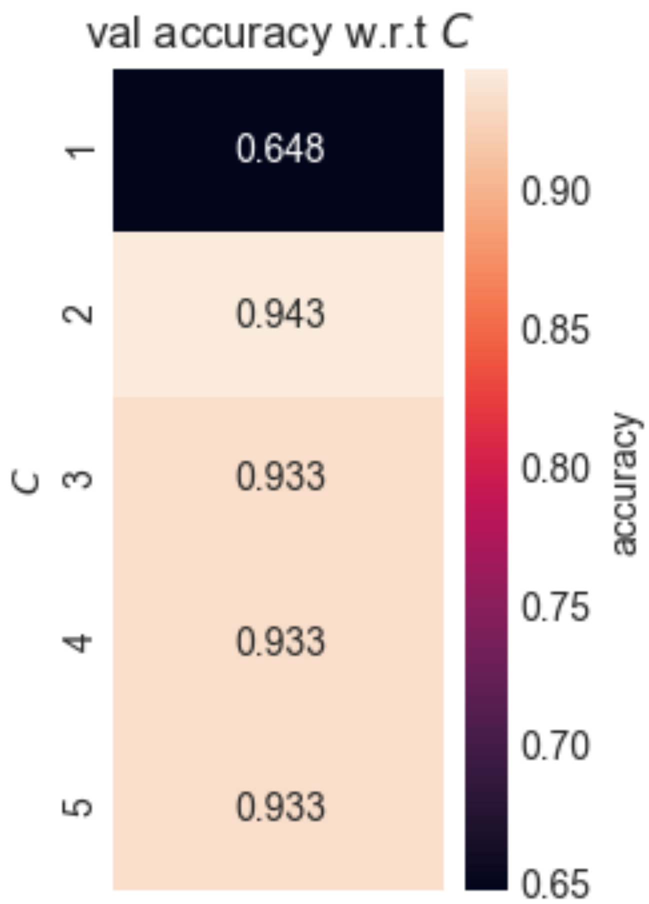
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [62]:

```
DT_clfGridSearch = decisionTreeTrainValidation(X3_train_val, Y3_train_val, D_lis
t, CV)
accuracyTrain = DT_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = DT_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', D_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', D_list)
```

```
[ 0.66029091  0.96499107  0.9788211   0.99894737  1.        ]
[ 0.64761905  0.94285714  0.93333333  0.93333333  0.93333333]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.660
0.965
0.979
0.999
1.000
0.96
0.90
0.84
0.78
0.72
0.66
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.648
0.943
0.933
0.933
0.933
0.90
0.85
0.80
0.75
0.70
0.65
accuracy

In [63]:

```
best_D = bestValue(accuracyValidation, D_list)
print('Best D: ' + str(best_D))

optimalClassifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=b
est_D).fit(X3_train_val, Y3_train_val)
pred = optimalClassifier.predict(X3_test)

accuracyTest = accuracy_score(Y3_test, pred)
DT_accuracyTestList_50_50.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.942857142857 from index 1.
Best D: 2
Test Accuracy Score: 0.905660377358
```

Mean of DT's Test Accuracies on (50% train, 50% test)

In [64]:

```
print('DT_accuracyTestList:' + str(DT_accuracyTestList_50_50))
DT_accuracyAverage_50_50 = statistics.mean(DT_accuracyTestList_50_50)
print('DT_accuracyTestList mean: ' + str(DT_accuracyAverage_50_50))
```

```
DT_accuracyTestList:[0.90566037735849059, 0.92452830188679247, 0.905
66037735849059]
DT_accuracyTestList mean: 0.911949685535
```

Decision Tree on (20% train, 80% test)

Splits the 3 shuffled datasets into 2 parts:

1. (20% of all the data points) ---> Training set + Validation Set.
2. (80% of all the data points) ---> Test set.

In [65]:

```
X1_train_val, Y1_train_val, X1_test, Y1_test = partitionData(X1, Y1, 0.2)
X2_train_val, Y2_train_val, X2_test, Y2_test = partitionData(X2, Y2, 0.2)
X3_train_val, Y3_train_val, X3_test, Y3_test = partitionData(X3, Y3, 0.2)
```

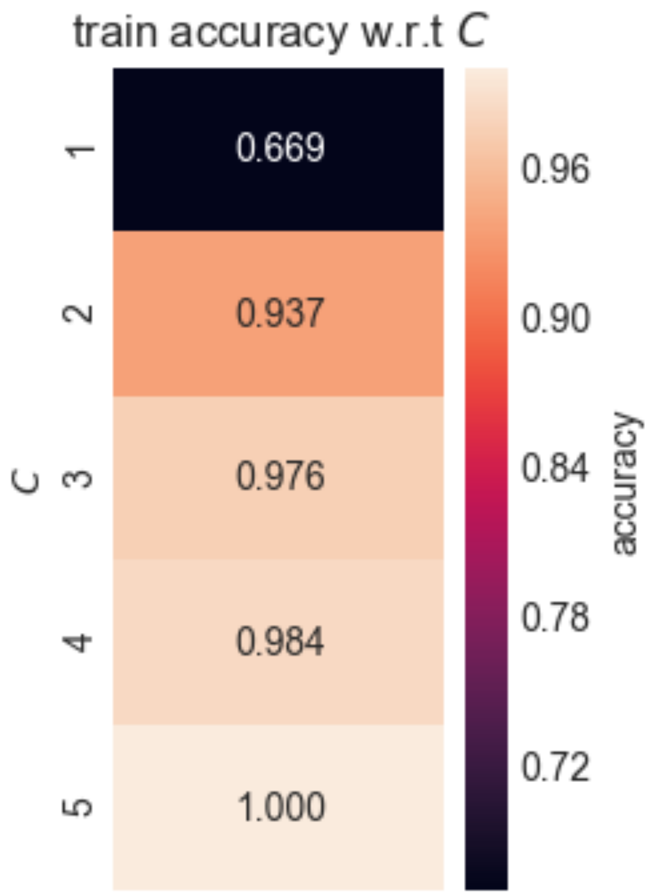
1st Run)

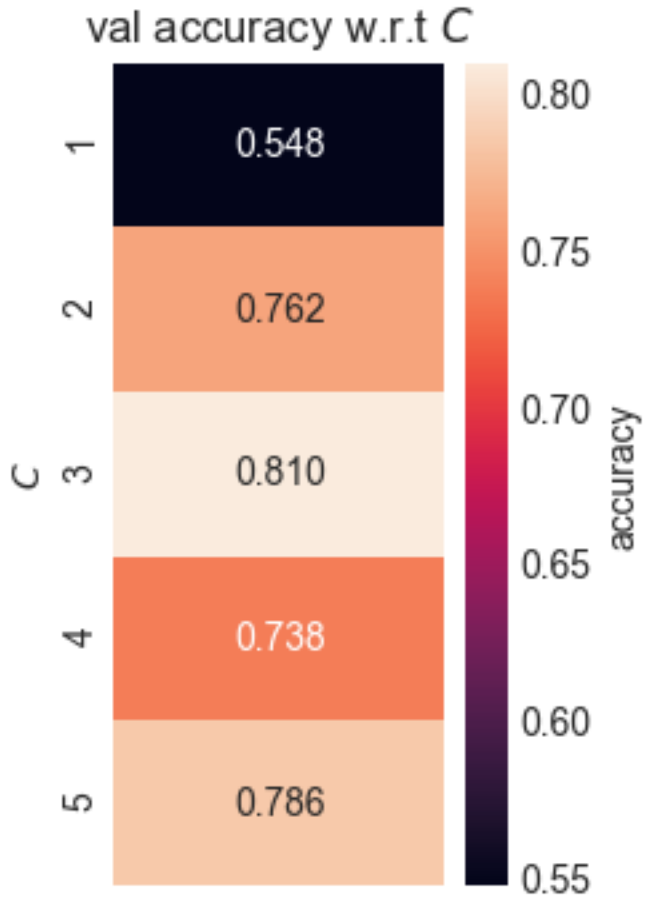
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [66]:

```
DT_clfGridSearch = decisionTreeTrainValidation(X1_train_val, Y1_train_val, D_lis
t, CV)
accuracyTrain = DT_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = DT_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', D_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', D_list)
```

```
[ 0.66946815  0.93691353  0.97636077  0.98440171  1.        ]
[ 0.54761905  0.76190476  0.80952381  0.73809524  0.78571429]
```





In [67]:

```
best_D = bestValue(accuracyValidation, D_list)
print('Best D: ' + str(best_D))

optimalClassifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=b
est_D).fit(X1_train_val, Y1_train_val)
pred = optimalClassifier.predict(X1_test)

accuracyTest = accuracy_score(Y1_test, pred)
DT_accuracyTestList_20_80.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.809523809524 from index 2.
Best D: 3
Test Accuracy Score: 0.905325443787
```

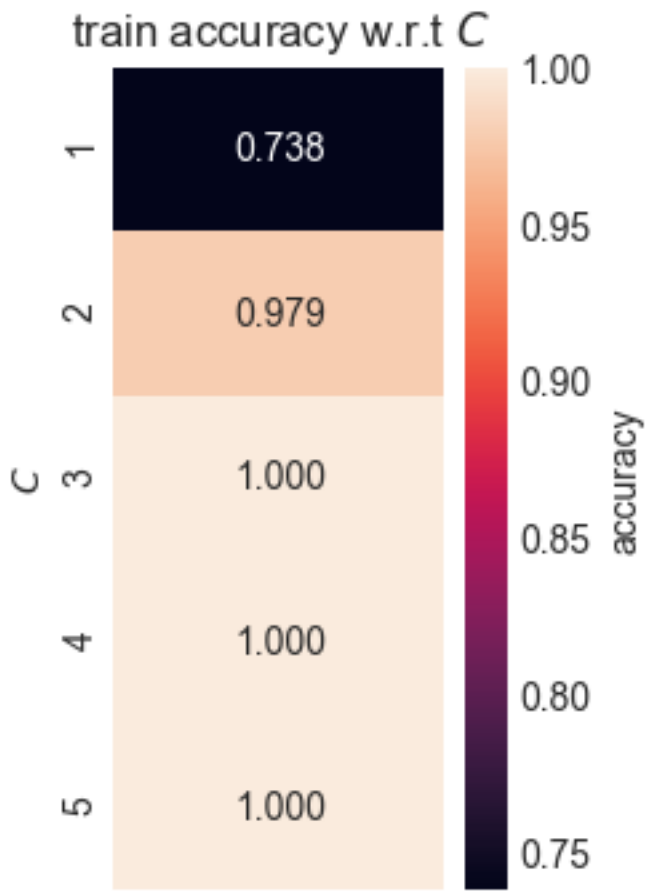
2nd Run)

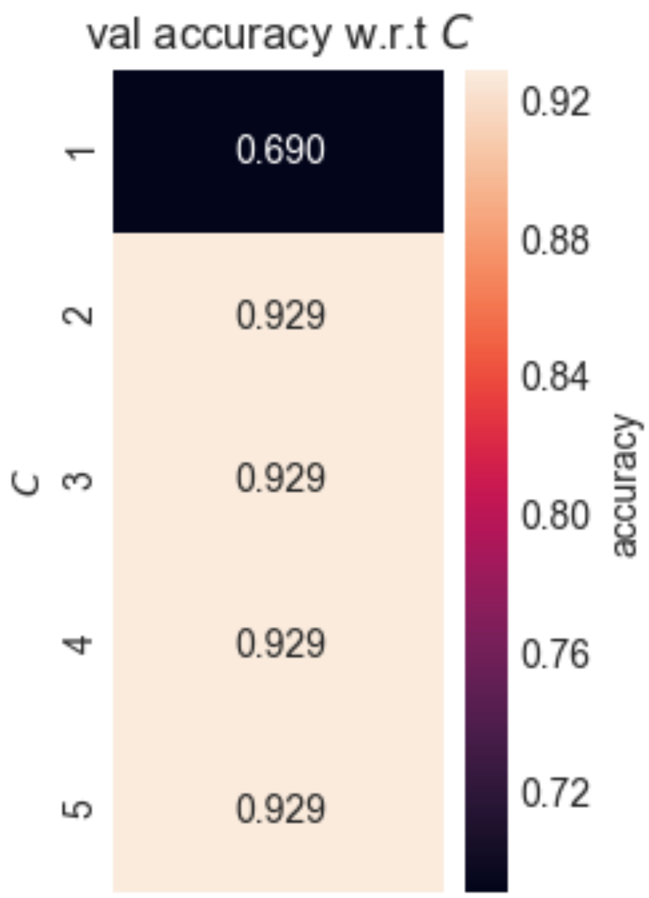
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [68]:

```
DT_clfGridSearch = decisionTreeTrainValidation(X2_train_val, Y2_train_val, D_lis
t, CV)
accuracyTrain = DT_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = DT_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', D_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', D_list)
```

```
[ 0.73804939  0.97886895  1.          1.          1.        ]
[ 0.69047619  0.92857143  0.92857143  0.92857143  0.92857143]
```





In [69]:

```
best_D = bestValue(accuracyValidation, D_list)
print('Best D: ' + str(best_D))

optimalClassifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=b
est_D).fit(X2_train_val, Y2_train_val)
pred = optimalClassifier.predict(X2_test)

accuracyTest = accuracy_score(Y2_test, pred)
DT_accuracyTestList_20_80.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.928571428571 from index 1.
Best D: 2
Test Accuracy Score: 0.822485207101
```

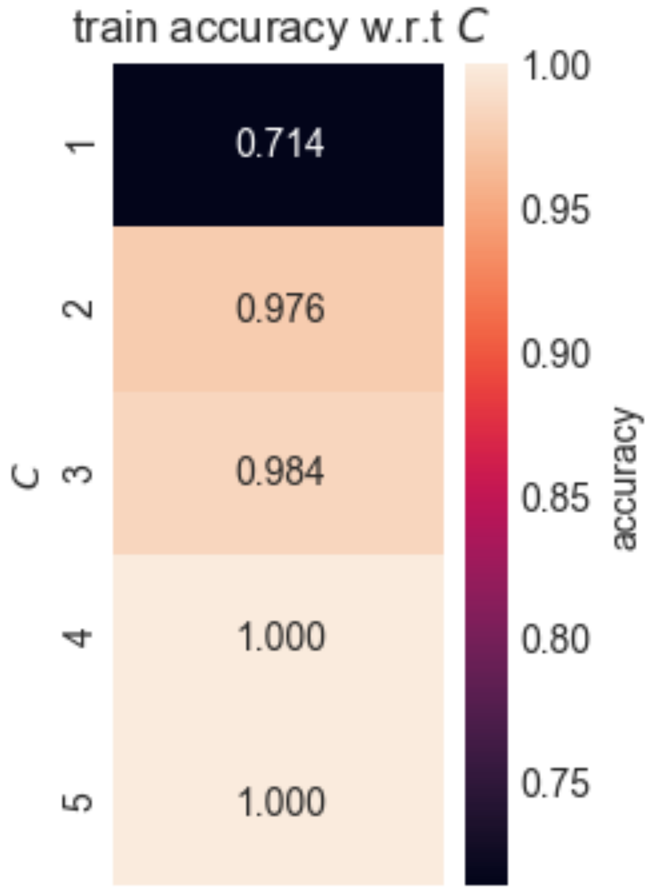
3rd Run)

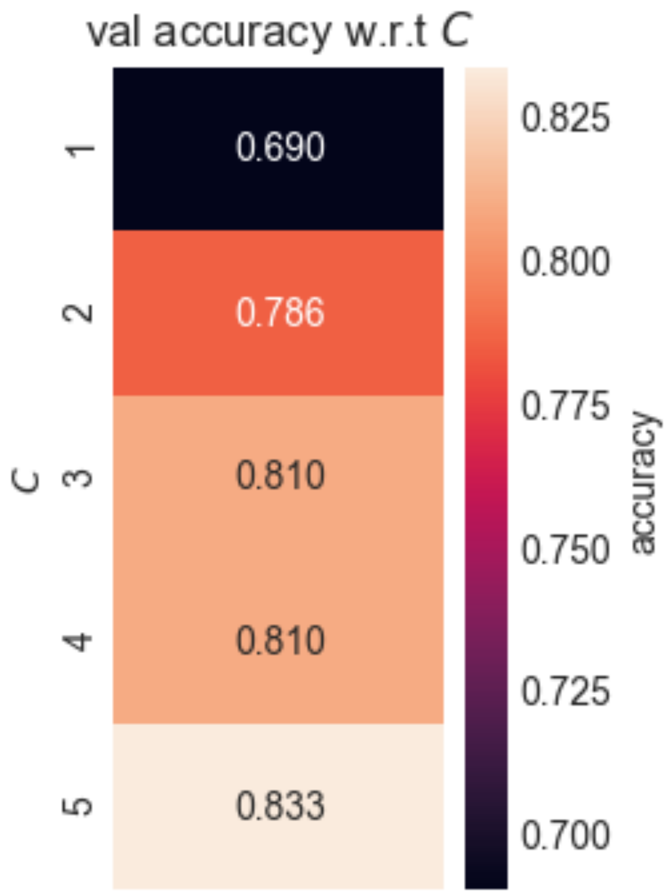
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [70]:

```
DT_clfGridSearch = decisionTreeTrainValidation(X3_train_val, Y3_train_val, D_lis
t, CV)
accuracyTrain = DT_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = DT_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', D_list)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', D_list)
```

```
[ 0.71423812  0.9763012   0.98433818  1.          1.        ]
[ 0.69047619  0.78571429  0.80952381  0.80952381  0.83333333]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.714
0.976
0.984
1.000
1.000
accuracy
1.00
0.95
0.90
0.85
0.80
0.75


val accuracy w.r.t C
C
1
2
3
4
5
0.690
0.786
0.810
0.810
0.833
accuracy
0.825
0.800
0.775
0.750
0.725
0.700

In [71]:

```
best_D = bestValue(accuracyValidation, D_list)
print('Best D: ' + str(best_D))

optimalClassifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=b
est_D).fit(X3_train_val, Y3_train_val)
pred = optimalClassifier.predict(X3_test)

accuracyTest = accuracy_score(Y3_test, pred)
DT_accuracyTestList_20_80.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.833333333333 from index 4.
Best D: 5
Test Accuracy Score: 0.923076923077
```

Mean of DT's Test Accuracies on (20% train, 80% test)

In [72]:

```
print('DT_accuracyTestList:' + str(DT_accuracyTestList_20_80))
DT_accuracyAverage_20_80 = statistics.mean(DT_accuracyTestList_20_80)
print('DT_accuracyTestList mean: ' + str(DT_accuracyAverage_20_80))
```

```
DT_accuracyTestList:[0.90532544378698221, 0.8224852071005917, 0.9230
7692307692313]
DT_accuracyTestList mean: 0.883629191321
```

Results of Decision Tree

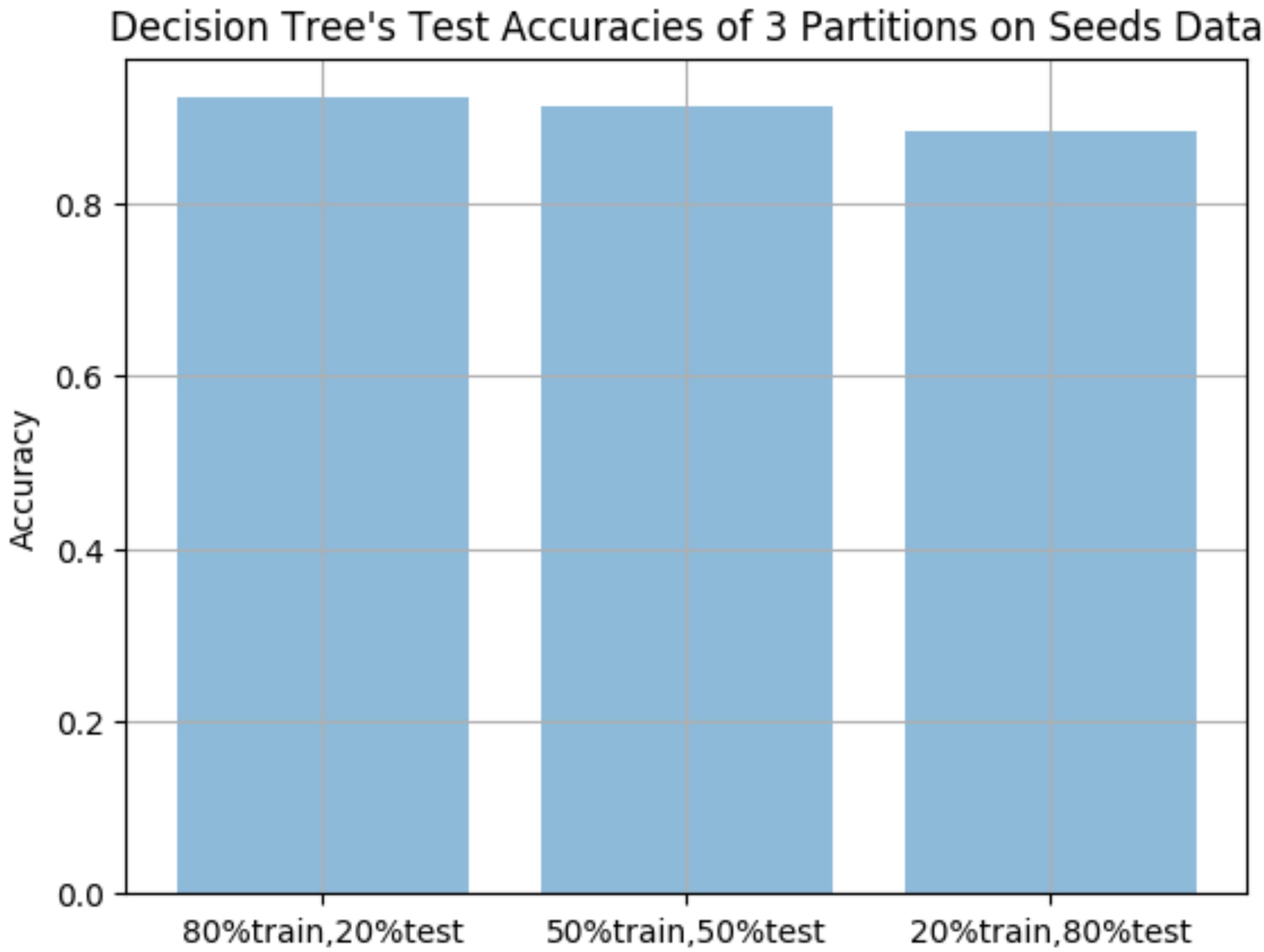
In [73]:

```
print('DT_accuracyTestList (80% train, 20% test) partition mean: ' + str(DT_accu
racyAverage_80_20))
print('DT_accuracyTestList (50% train, 50% test) partition mean: ' + str(DT_accu
racyAverage_50_50))
print('DT_accuracyTestList (20% train, 80% test) partition mean: ' + str(DT_accu
racyAverage_20_80))
```

```
DT_accuracyTestList (80% train, 20% test) partition mean: 0.92248062
0155
DT_accuracyTestList (50% train, 50% test) partition mean: 0.91194968
5535
DT_accuracyTestList (20% train, 80% test) partition mean: 0.88362919
1321
```

In [111]:

```
displayAccuracies('Decision Tree', 'Seeds Data', DT_accuracyAverage_80_20, DT_accuracyAverage_50_50, DT_accuracyAverage_20_80)
printAccuracies('DT', DT_accuracyTestList_80_20, DT_accuracyTestList_50_50, DT_accuracyTestList_20_80)
```




```
Accuracy of DT's 3 trials on (80% train, 20% test) partition :[0.93023255813953487, 0.93023255813953487, 0.90697674418604646]
Mean Accuracy of DT on (80% train, 20% test) partition: 0.922480620155

Accuracy of DT's 3 trials on (50% train, 50% test) partition :[0.90566037735849059, 0.92452830188679247, 0.90566037735849059]
Mean Accuracy of DT on (50% train, 50% test) partition: 0.911949685535

Accuracy of DT's 3 trials on (20% train, 80% test) partition:[0.90532544378698221, 0.8224852071005917, 0.92307692307692313]
Mean Accuracy of DT on (20% train, 80% test) partition: 0.883629191321
```

Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

In [75]:

```
#Global Variables For Random Forest
max_depth_List = [1,2,3,4,5]
RF_accuracyTestList_80_20 = []
RF_accuracyTestList_50_50 = []
RF_accuracyTestList_20_80 = []
```

In [76]:

```
from sklearn.ensemble import RandomForestClassifier

#max_depth_List: The chosen hyperparameter.
#cv: Number of folds when doing cross validation.
def randomForestTrainValidation(X_train_val, Y_train_val, max_depth_List, CV):

    #svm_classifier = svm.SVC(kernel = 'linear')
    RF_classifier = RandomForestClassifier()

    parameters = {'max_depth':max_depth_List}

# param_grid = {
#     'bootstrap': [True],
#     'max_depth': [80, 90, 100, 110],
#     'max_features': [2, 3],
#     'min_samples_leaf': [3, 4, 5],
#     'min_samples_split': [8, 10, 12],
#     'n_estimators': [100, 200, 300, 1000]
# }

    RF_clfGridSearch = GridSearchCV(RF_classifier, param_grid=parameters, cv=CV,
return_train_score=True)
    RF_clfGridSearch.fit(X_train_val, Y_train_val)

    return RF_clfGridSearch
```

Random Forest on (80% train, 20% test)

Splits the 3 shuffled datasets into 2 parts:

1. (80% of all the data points) ---> Training set + Validation Set.
2. (20% of all the data points) ---> Test set.

In [77]:

```
X1_train_val, Y1_train_val, X1_test, Y1_test = partitionData(X1, Y1, 0.8)
X2_train_val, Y2_train_val, X2_test, Y2_test = partitionData(X2, Y2, 0.8)
X3_train_val, Y3_train_val, X3_test, Y3_test = partitionData(X3, Y3, 0.8)
```

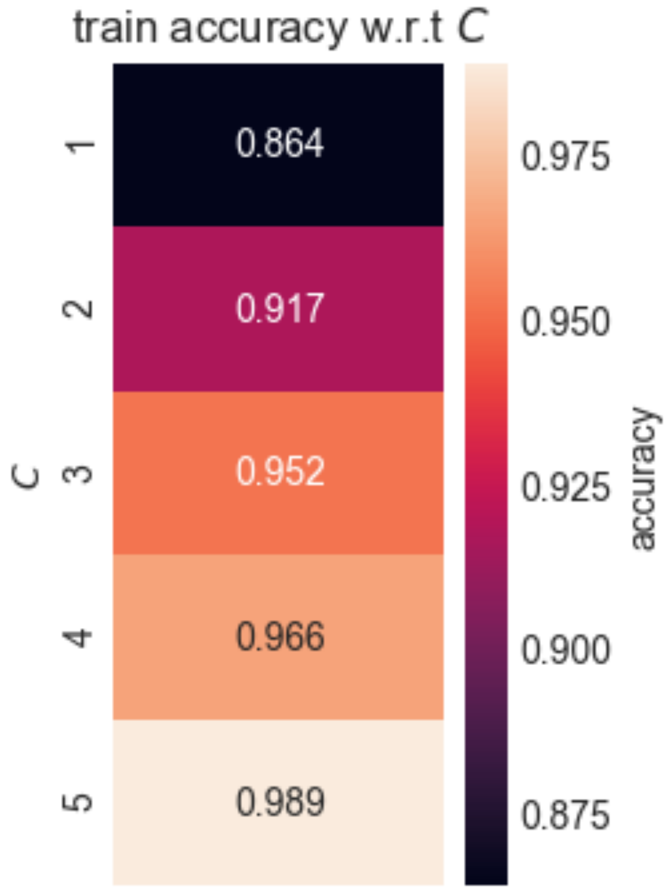
1st Run)

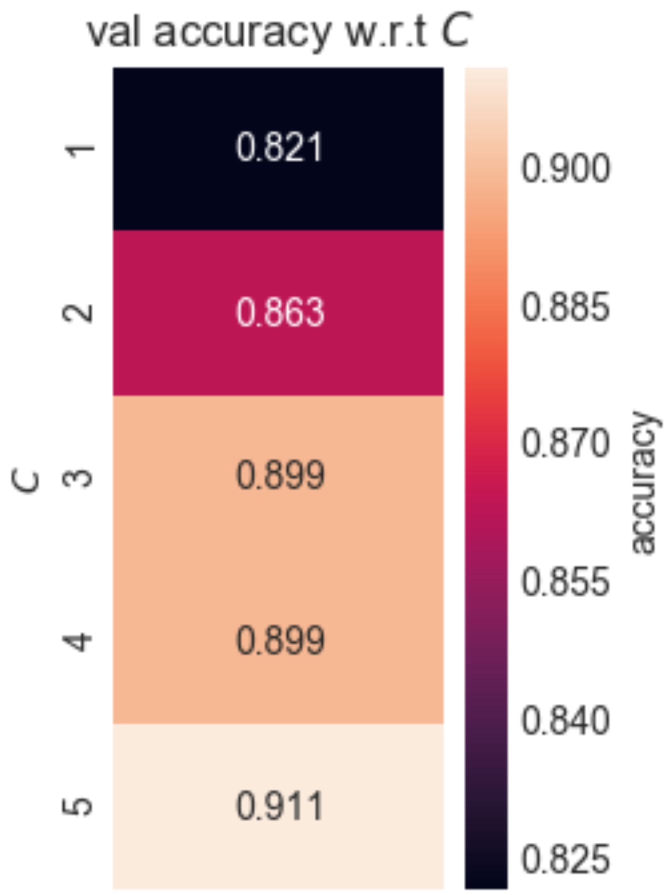
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [78]:

```
RF_clfGridSearch = randomForestTrainValidation(X1_train_val, Y1_train_val, max_d
epth_List, CV)
accuracyTrain = RF_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = RF_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', max_depth_List)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', max_depth_List)
```

```
[ 0.86426068  0.91732609  0.95236477  0.96558395  0.98876775]
[ 0.82142857  0.86309524  0.89880952  0.89880952  0.91071429]
```


train accuracy w.r.t C
0.864
0.917
0.952
0.966
0.989
1
2
3
4
5
C
0.975
0.950
0.925
0.900
0.875
accuracy


val accuracy w.r.t C
0.821
0.863
0.899
0.899
0.911
1
2
3
4
5
C
0.900
0.885
0.870
0.855
0.840
0.825
accuracy

In [79]:

```
best_max_depth = bestValue(accuracyValidation, max_depth_List)
print('Best max_depth: ' + str(best_max_depth))

optimalClassifier = RandomForestClassifier(max_depth=best_max_depth).fit(X1_trai
n_val, Y1_train_val)
pred = optimalClassifier.predict(X1_test)

accuracyTest = accuracy_score(Y1_test, pred)
RF_accuracyTestList_80_20.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.910714285714 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.906976744186
```

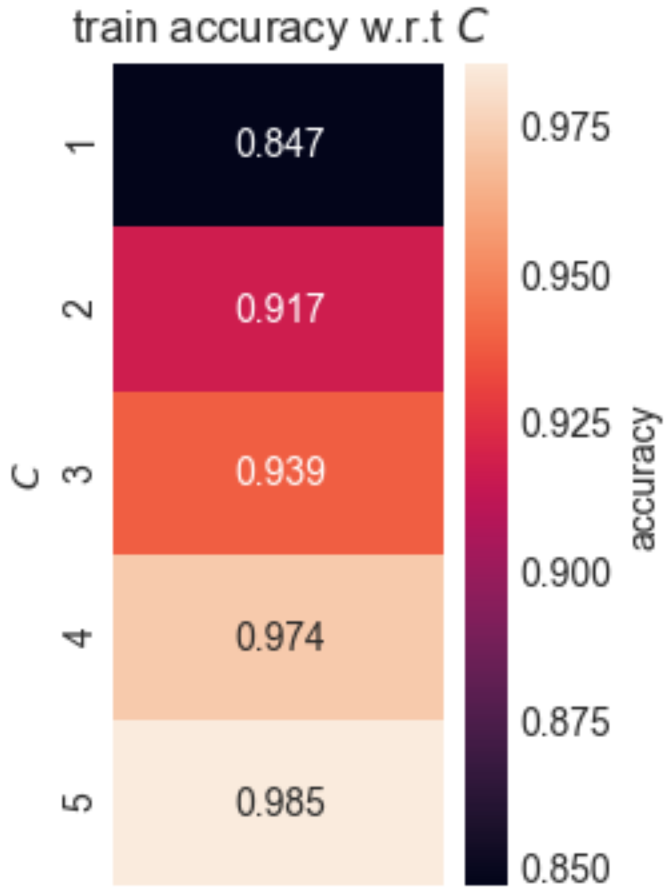
2nd Run)

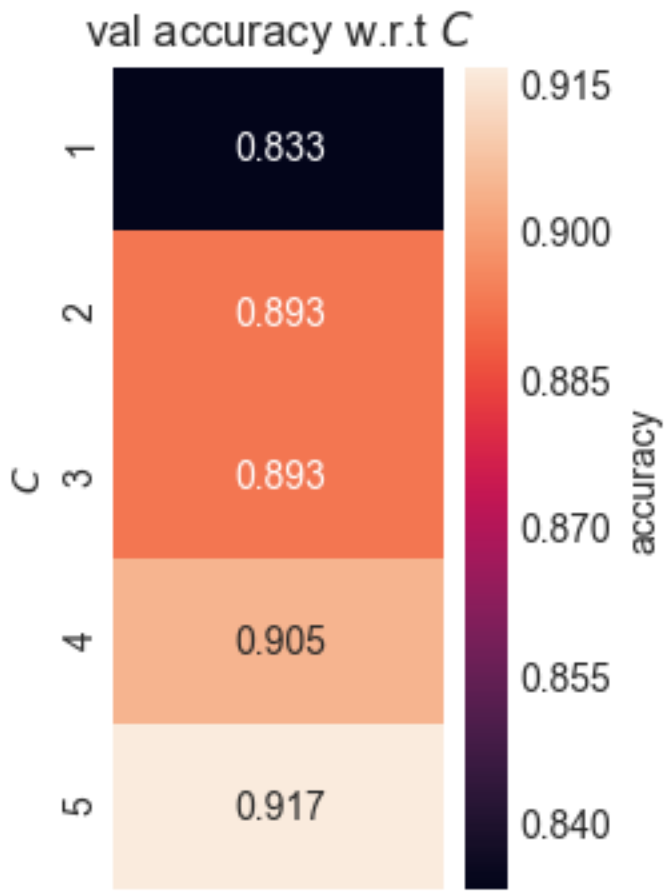
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [80]:

```
RF_clfGridSearch = randomForestTrainValidation(X2_train_val, Y2_train_val, max_d
epth_List, CV)
accuracyTrain = RF_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = RF_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', max_depth_List)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', max_depth_List)
```

```
[ 0.8470461   0.91733574  0.93852425  0.9735215   0.98544255]
[ 0.83333333  0.89285714  0.89285714  0.9047619   0.91666667]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.847
0.917
0.939
0.974
0.985
0.975
0.950
0.925
0.900
0.875
0.850
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.833
0.893
0.893
0.905
0.917
0.915
0.900
0.885
0.870
0.855
0.840
accuracy

In [81]:

```
best_max_depth = bestValue(accuracyValidation, max_depth_List)
print('Best max_depth: ' + str(best_max_depth))

optimalClassifier = RandomForestClassifier(max_depth=best_max_depth).fit(X2_trai
n_val, Y2_train_val)
pred = optimalClassifier.predict(X2_test)

accuracyTest = accuracy_score(Y2_test, pred)
RF_accuracyTestList_80_20.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.916666666667 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.883720930233
```

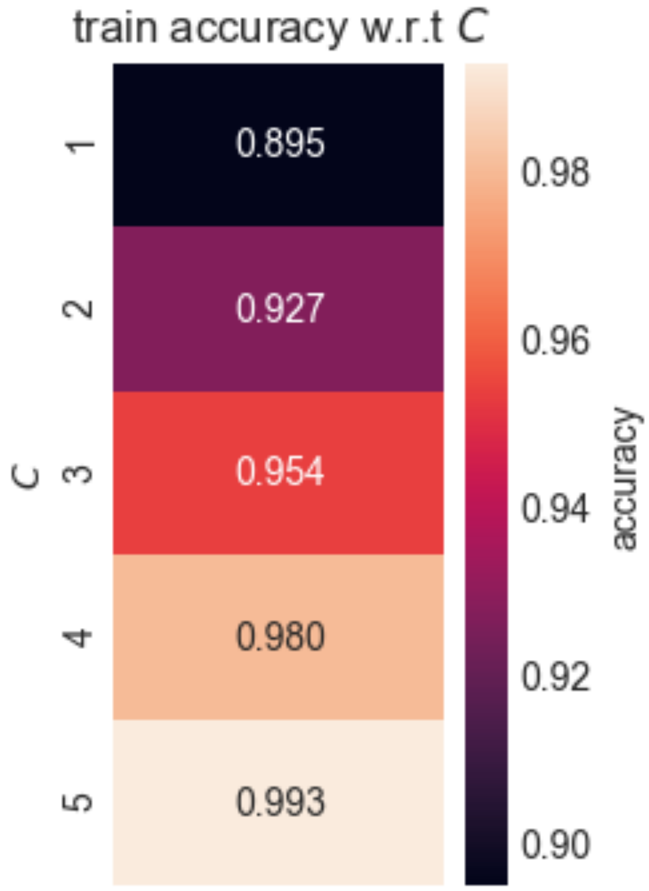
3rd Run)

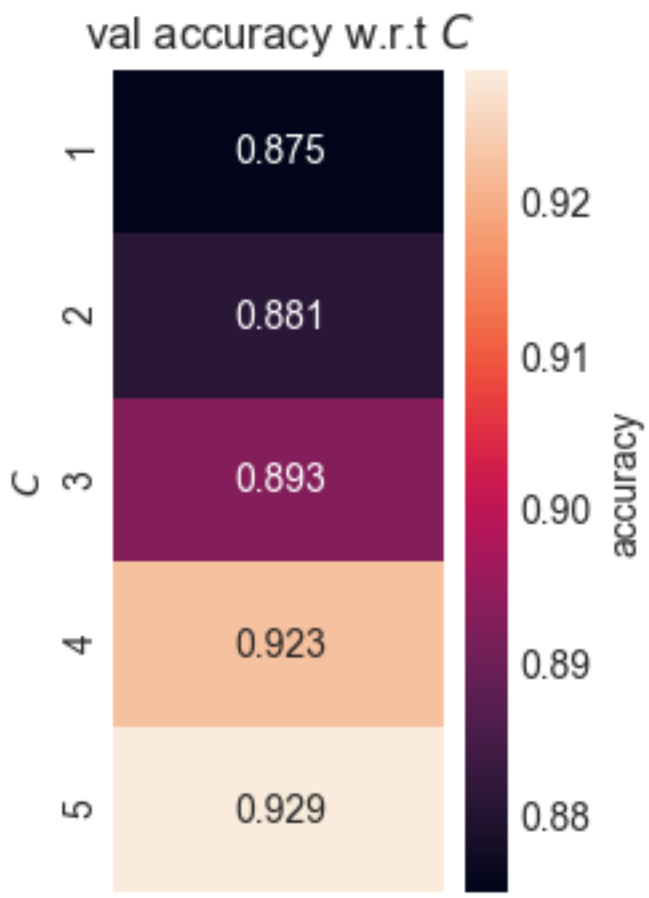
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [82]:

```
RF_clfGridSearch = randomForestTrainValidation(X3_train_val, Y3_train_val, max_d
epth_List, CV)
accuracyTrain = RF_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = RF_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', max_depth_List)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', max_depth_List)
```

```
[ 0.89485748  0.92730995  0.95367499  0.98014872  0.9927412 ]
[ 0.875       0.88095238  0.89285714  0.92261905  0.92857143]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.895
0.927
0.954
0.980
0.993
0.98
0.96
0.94
0.92
0.90
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.875
0.881
0.893
0.923
0.929
0.92
0.91
0.90
0.89
0.88
accuracy

In [83]:

```
best_max_depth = bestValue(accuracyValidation, max_depth_List)
print('Best max_depth: ' + str(best_max_depth))

optimalClassifier = RandomForestClassifier(max_depth=best_max_depth).fit(X3_trai
n_val, Y3_train_val)
pred = optimalClassifier.predict(X3_test)

accuracyTest = accuracy_score(Y3_test, pred)
RF_accuracyTestList_80_20.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.928571428571 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.93023255814
```

Mean of RF's Test Accuracies on (80% train, 20% test)

In [84]:

```
print('RF_accuracyTestList:' + str(RF_accuracyTestList_80_20))
RF_accuracyAverage_80_20 = statistics.mean(RF_accuracyTestList_80_20)
print('RF_accuracyTestList mean: ' + str(RF_accuracyAverage_80_20))
```

```
RF_accuracyTestList:[0.90697674418604646, 0.88372093023255816, 0.930
23255813953487]
RF_accuracyTestList mean: 0.906976744186
```

Random Forest on (50% train, 50% test)

Splits the 3 shuffled datasets into 2 parts:

1. (50% of all the data points) ---> Training set + Validation Set.
2. (50% of all the data points) ---> Test set.

In [85]:

```
X1_train_val, Y1_train_val, X1_test, Y1_test = partitionData(X1, Y1, 0.5)
X2_train_val, Y2_train_val, X2_test, Y2_test = partitionData(X2, Y2, 0.5)
X3_train_val, Y3_train_val, X3_test, Y3_test = partitionData(X3, Y3, 0.5)
```

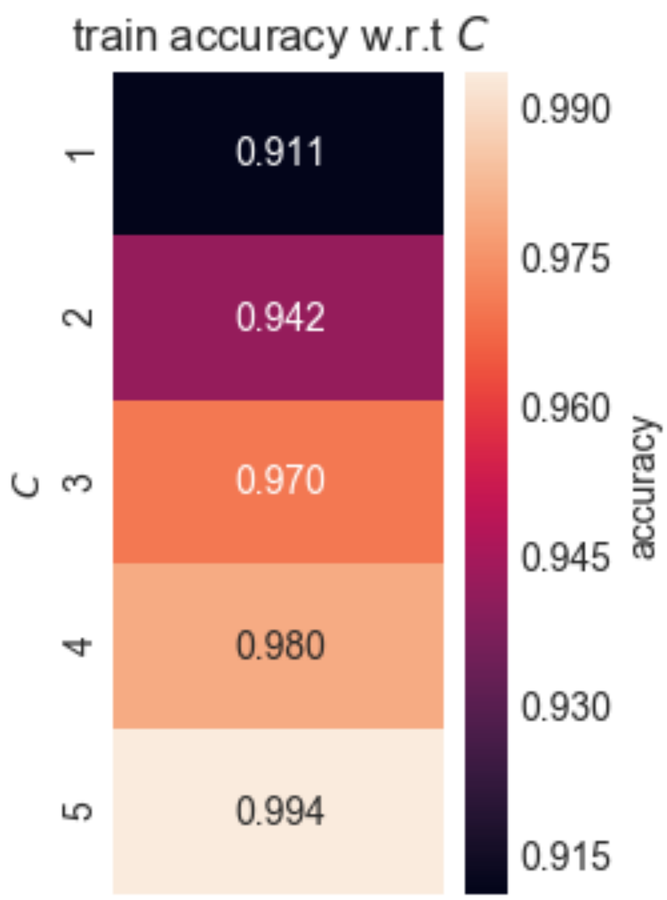
1st Run)

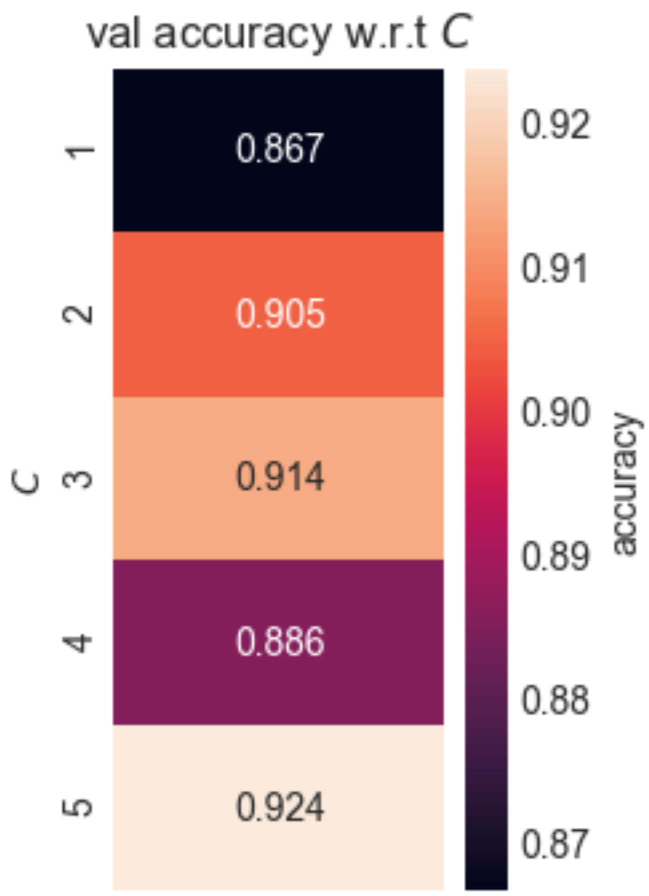
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [86]:

```
RF_clfGridSearch = randomForestTrainValidation(X1_train_val, Y1_train_val, max_d
epth_List, CV)
accuracyTrain = RF_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = RF_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', max_depth_List)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', max_depth_List)
```

```
[ 0.91112587  0.94181465  0.97040913  0.97989614  0.99363847]
[ 0.86666667  0.9047619   0.91428571  0.88571429  0.92380952]
```





In [87]:

```
best_max_depth = bestValue(accuracyValidation, max_depth_List)
print('Best max_depth: ' + str(best_max_depth))

optimalClassifier = RandomForestClassifier(max_depth=best_max_depth).fit(X1_trai
n_val, Y1_train_val)
pred = optimalClassifier.predict(X1_test)

accuracyTest = accuracy_score(Y1_test, pred)
RF_accuracyTestList_50_50.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.92380952381 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.915094339623
```

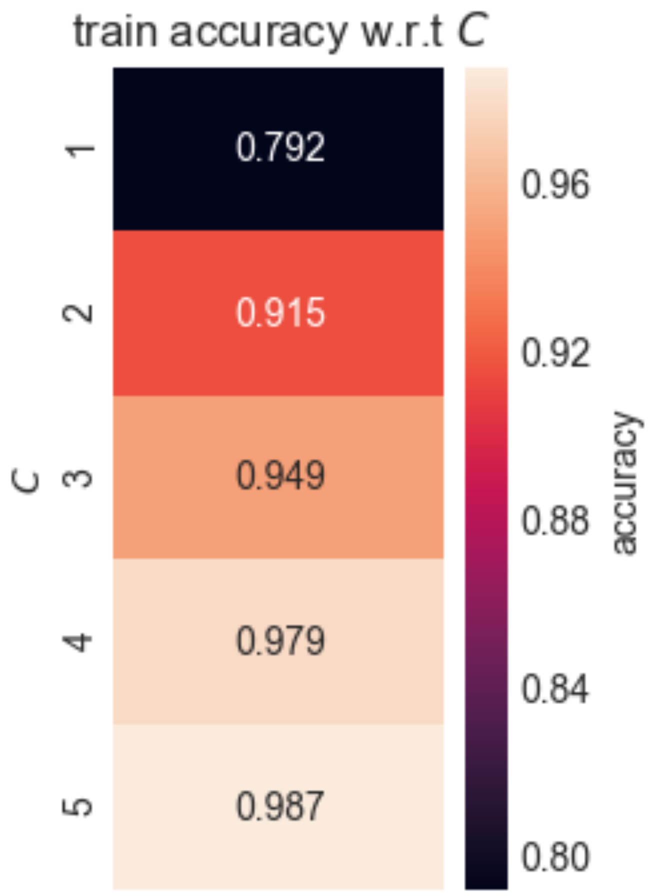
2nd Run)

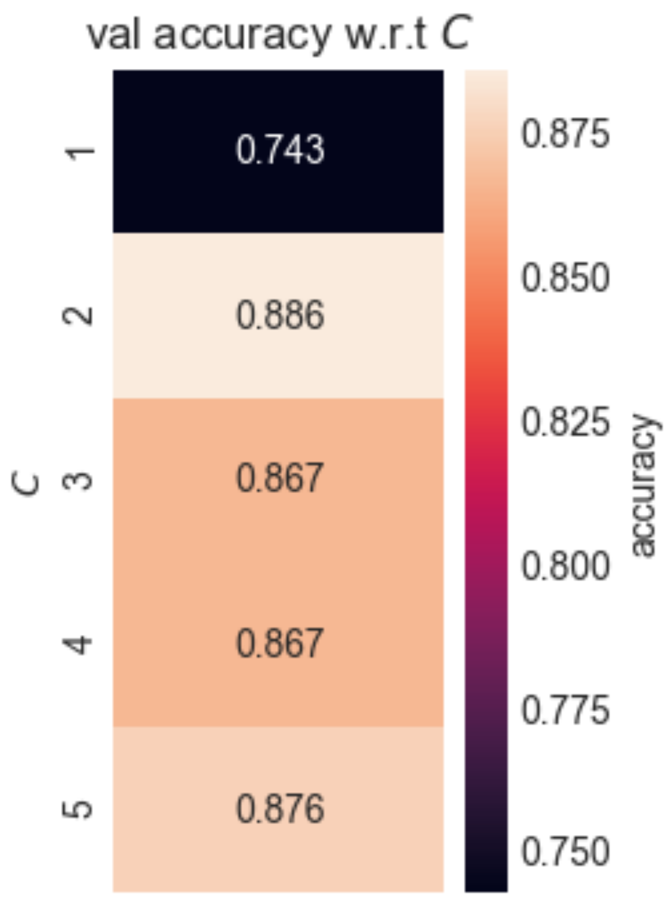
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [88]:

```
RF_clfGridSearch = randomForestTrainValidation(X2_train_val, Y2_train_val, max_d
epth_List, CV)
accuracyTrain = RF_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = RF_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', max_depth_List)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', max_depth_List)
```

```
[ 0.79166524  0.91521677  0.94919641  0.97874271  0.98731054]
[ 0.74285714  0.88571429  0.86666667  0.86666667  0.87619048]
```





In [89]:

```
best_max_depth = bestValue(accuracyValidation, max_depth_List)
print('Best max_depth: ' + str(best_max_depth))

optimalClassifier = RandomForestClassifier(max_depth=best_max_depth).fit(X2_trai
n_val, Y2_train_val)
pred = optimalClassifier.predict(X2_test)

accuracyTest = accuracy_score(Y2_test, pred)
RF_accuracyTestList_50_50.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.885714285714 from index 1.
Best max_depth: 2
Test Accuracy Score: 0.933962264151
```

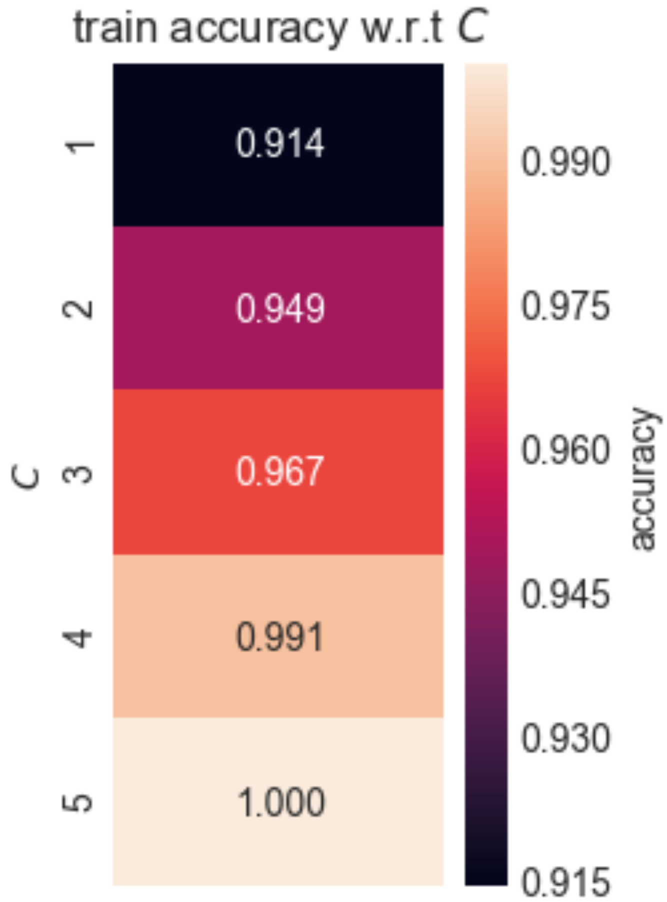
3rd Run)

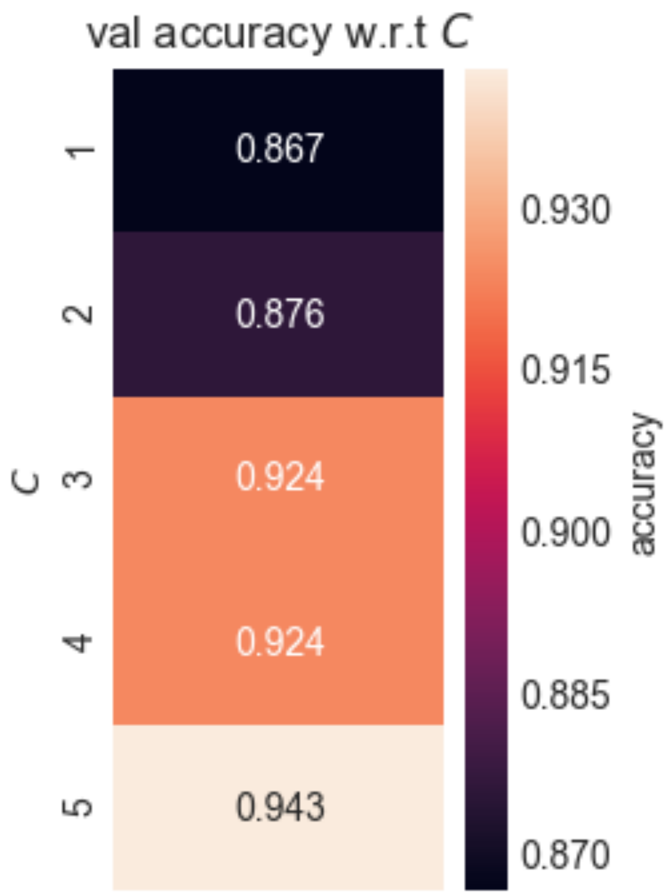
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [90]:

```
RF_clfGridSearch = randomForestTrainValidation(X3_train_val, Y3_train_val, max_d
epth_List, CV)
accuracyTrain = RF_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = RF_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', max_depth_List)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', max_depth_List)
```

```
[ 0.91442339  0.9492991   0.96717402  0.99053611  1.        ]
[ 0.86666667  0.87619048  0.92380952  0.92380952  0.94285714]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.914
0.949
0.967
0.991
1.000
0.990
0.975
0.960
0.945
0.930
0.915
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.867
0.876
0.924
0.924
0.943
0.930
0.915
0.900
0.885
0.870
accuracy

In [91]:

```
best_max_depth = bestValue(accuracyValidation, max_depth_List)
print('Best max_depth: ' + str(best_max_depth))

optimalClassifier = RandomForestClassifier(max_depth=best_max_depth).fit(X3_trai
n_val, Y3_train_val)
pred = optimalClassifier.predict(X3_test)

accuracyTest = accuracy_score(Y3_test, pred)
RF_accuracyTestList_50_50.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.942857142857 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.905660377358
```

Mean of RF's Test Accuracies on (50% train, 50% test)

In [92]:

```
print('RF_accuracyTestList:' + str(RF_accuracyTestList_50_50))
RF_accuracyAverage_50_50 = statistics.mean(RF_accuracyTestList_50_50)
print('RF_accuracyTestList mean: ' + str(RF_accuracyAverage_50_50))
```

```
RF_accuracyTestList:[0.91509433962264153, 0.93396226415094341, 0.905
66037735849059]
RF_accuracyTestList mean: 0.918238993711
```

Random Forest on (20% train, 80% test)

Splits the 3 shuffled datasets into 2 parts:

1. (20% of all the data points) ---> Training set + Validation Set.
2. (80% of all the data points) ---> Test set.

In [93]:

```
X1_train_val, Y1_train_val, X1_test, Y1_test = partitionData(X1, Y1, 0.2)
X2_train_val, Y2_train_val, X2_test, Y2_test = partitionData(X2, Y2, 0.2)
X3_train_val, Y3_train_val, X3_test, Y3_test = partitionData(X3, Y3, 0.2)
```

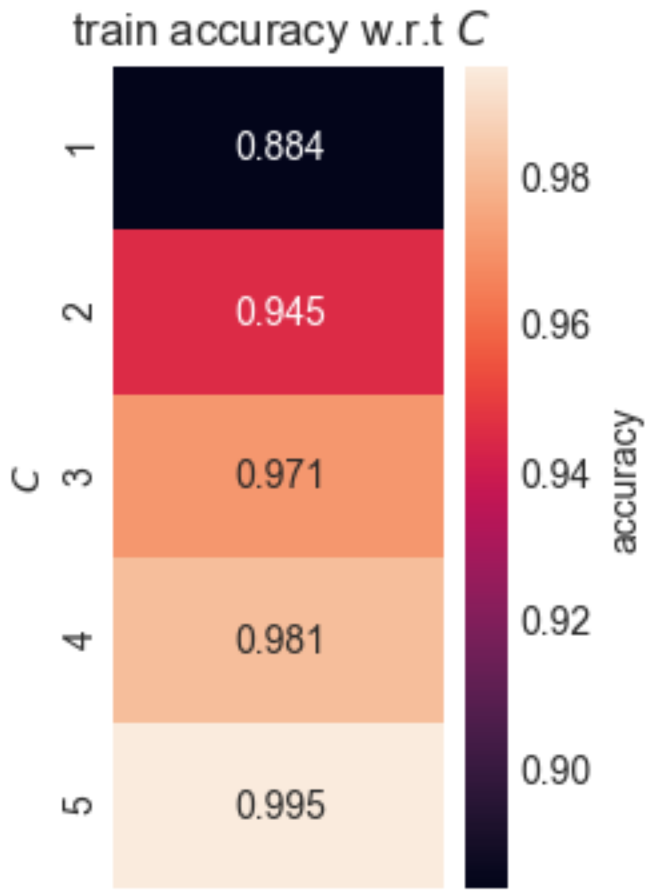
1st Run)

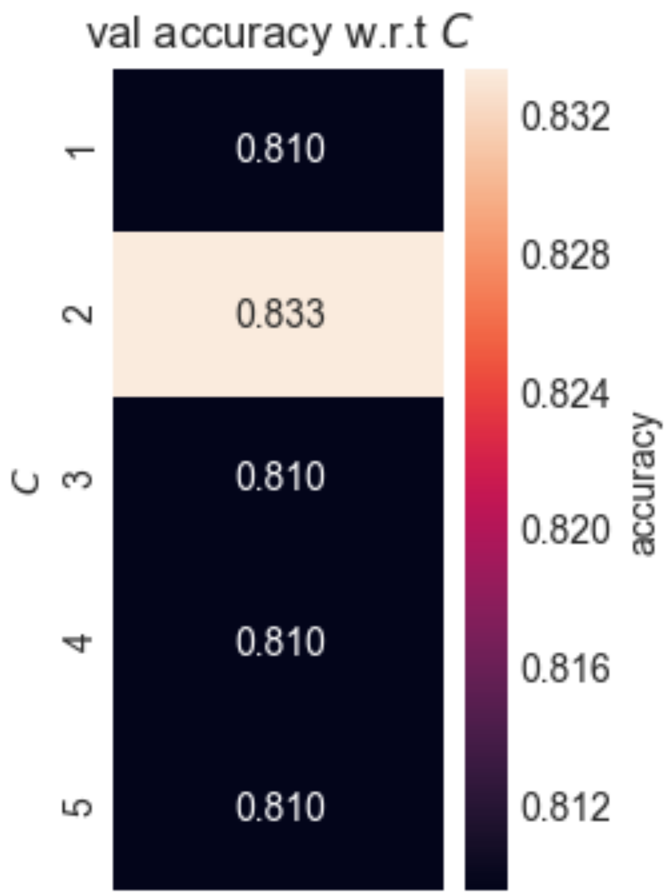
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [94]:

```
RF_clfGridSearch = randomForestTrainValidation(X1_train_val, Y1_train_val, max_d
epth_List, CV)
accuracyTrain = RF_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = RF_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', max_depth_List)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', max_depth_List)
```

```
[ 0.8837236   0.94487939  0.97116509  0.98134278  0.99465812]
[ 0.80952381  0.83333333  0.80952381  0.80952381  0.80952381]
```





In [95]:

```
best_max_depth = bestValue(accuracyValidation, max_depth_List)
print('Best max_depth: ' + str(best_max_depth))

optimalClassifier = RandomForestClassifier(max_depth=best_max_depth).fit(X1_trai
n_val, Y1_train_val)
pred = optimalClassifier.predict(X1_test)

accuracyTest = accuracy_score(Y1_test, pred)
RF_accuracyTestList_20_80.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.833333333333 from index 1.
Best max_depth: 2
Test Accuracy Score: 0.852071005917
```

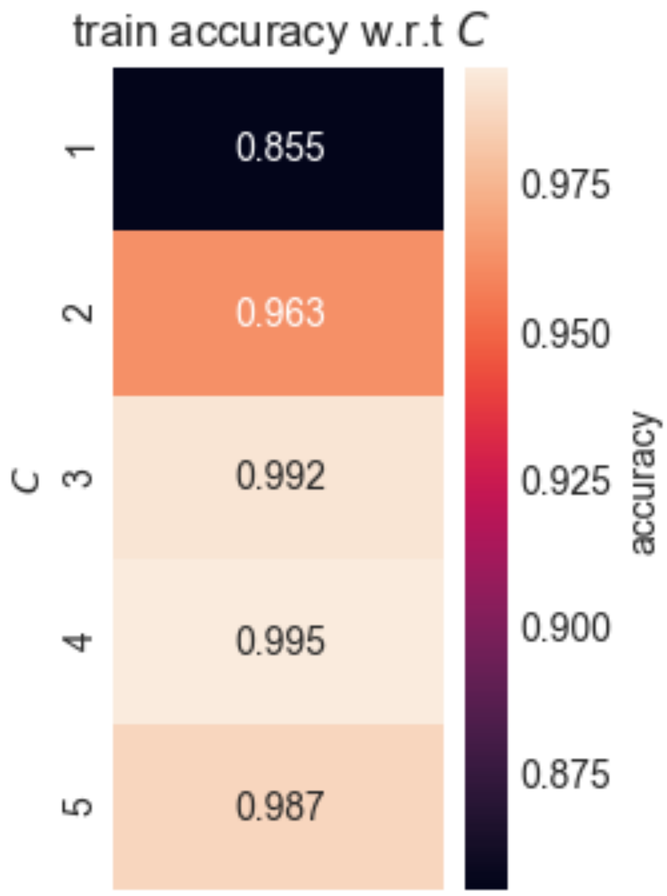
2nd Run)

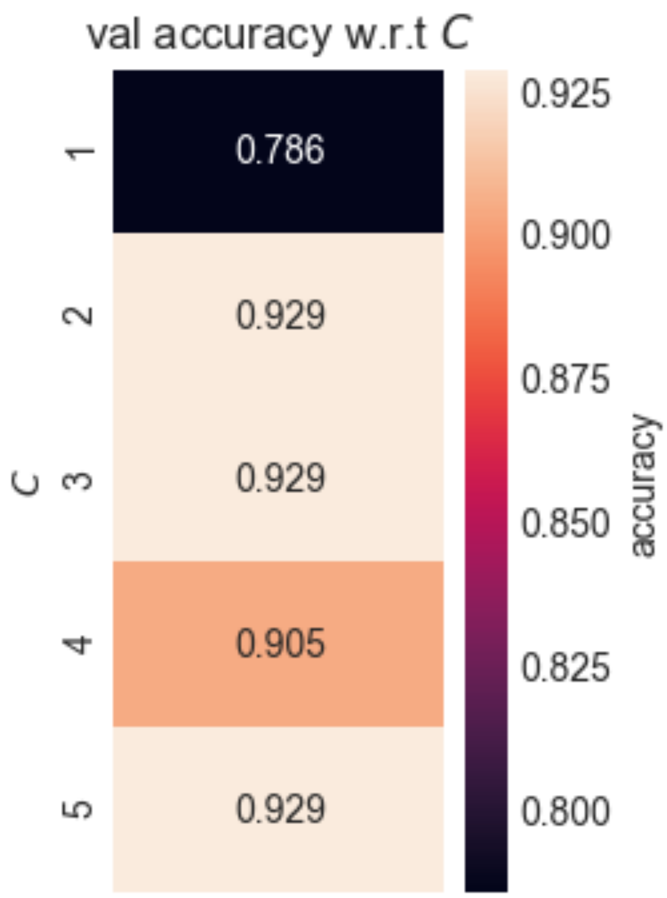
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [96]:

```
RF_clfGridSearch = randomForestTrainValidation(X2_train_val, Y2_train_val, max_d
epth_List, CV)
accuracyTrain = RF_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = RF_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', max_depth_List)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', max_depth_List)
```

```
[ 0.85510267  0.96307583  0.99210162  0.99480432  0.98690593]
[ 0.78571429  0.92857143  0.92857143  0.9047619   0.92857143]
```





In [97]:

```
best_max_depth = bestValue(accuracyValidation, max_depth_List)
print('Best max_depth: ' + str(best_max_depth))

optimalClassifier = RandomForestClassifier(max_depth=best_max_depth).fit(X2_trai
n_val, Y2_train_val)
pred = optimalClassifier.predict(X2_test)

accuracyTest = accuracy_score(Y2_test, pred)
RF_accuracyTestList_20_80.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.928571428571 from index 1.
Best max_depth: 2
Test Accuracy Score: 0.828402366864
```

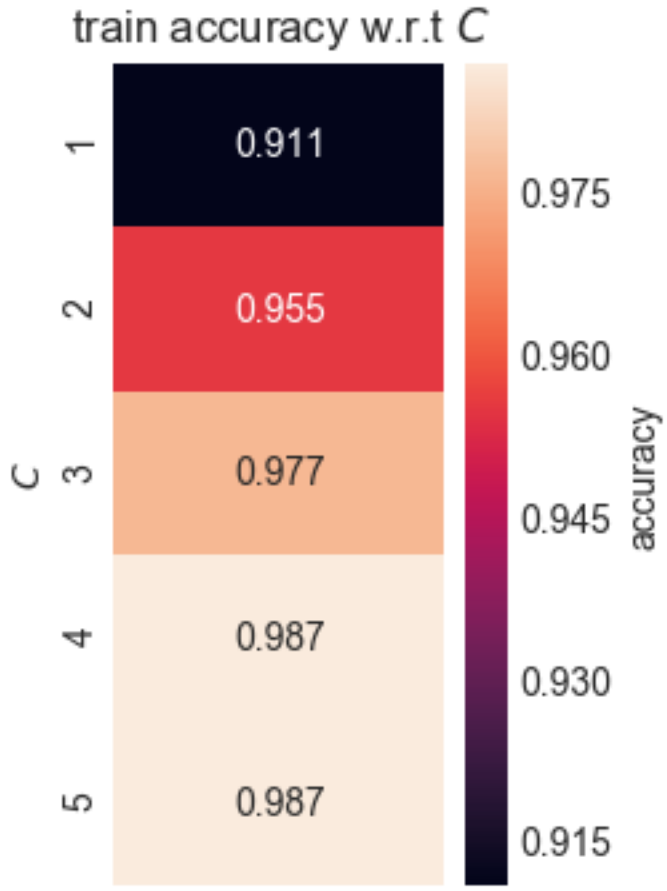
3rd Run)

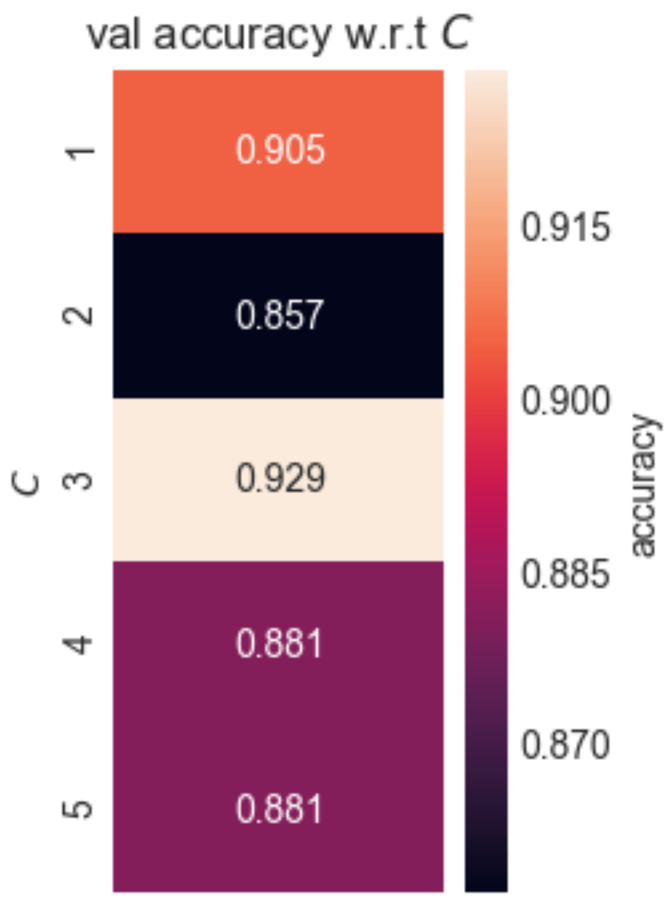
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [98]:

```
RF_clfGridSearch = randomForestTrainValidation(X3_train_val, Y3_train_val, max_d
epth_List, CV)
accuracyTrain = RF_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = RF_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)   #This is what shows up in the heat maps.

train_acc = (accuracyTrain).reshape(-1,1)
draw_heatmap_linear(train_acc, 'train accuracy', max_depth_List)
val_acc = (accuracyValidation).reshape(-1,1)
draw_heatmap_linear(val_acc, 'val accuracy', max_depth_List)
```

```
[ 0.91096218  0.95530115  0.9765784   0.98683481  0.98676369]
[ 0.9047619   0.85714286  0.92857143  0.88095238  0.88095238]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.911
0.955
0.977
0.987
0.987
0.975
0.960
0.945
0.930
0.915
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.905
0.857
0.929
0.881
0.881
0.915
0.900
0.885
0.870
accuracy

In [99]:

```
best_max_depth = bestValue(accuracyValidation, max_depth_List)
print('Best max_depth: ' + str(best_max_depth))

optimalClassifier = RandomForestClassifier(max_depth=best_max_depth).fit(X3_trai
n_val, Y3_train_val)
pred = optimalClassifier.predict(X3_test)

accuracyTest = accuracy_score(Y3_test, pred)
RF_accuracyTestList_20_80.append(accuracyTest)
print('Test Accuracy Score: ' + str(accuracyTest))
```

```
Largest value in accuracyValidation is 0.928571428571 from index 2.
Best max_depth: 3
Test Accuracy Score: 0.899408284024
```

Mean of RF's Test Accuracies on (20% train, 80% test)

In [100]:

```
print('RT_accuracyTestList:' + str(RF_accuracyTestList_20_80))
RF_accuracyAverage_20_80 = statistics.mean(RF_accuracyTestList_20_80)
print('RT_accuracyTestList mean: ' + str(RF_accuracyAverage_20_80))
```

```
RT_accuracyTestList:[0.85207100591715978, 0.82840236686390534, 0.899
40828402366868]
RT_accuracyTestList mean: 0.859960552268
```

Results of Random Forest

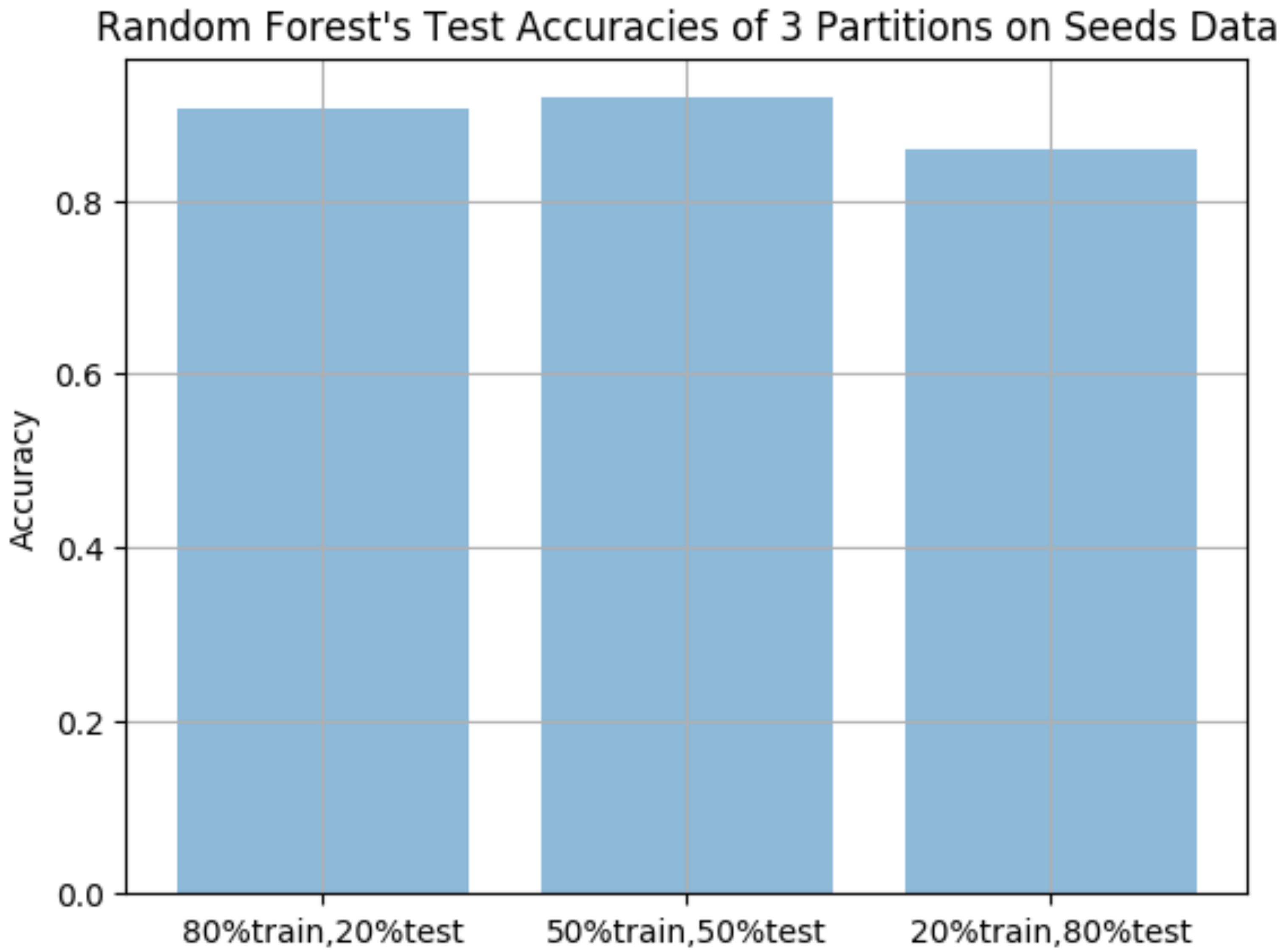
In [101]:

```
print('RT_accuracyTestList (80% train, 20% test) partition mean: ' + str(RF_accu
racyAverage_80_20))
print('RT_accuracyTestList (50% train, 50% test) partition mean: ' + str(RF_accu
racyAverage_50_50))
print('RT_accuracyTestList (20% train, 80% test) partition mean: ' + str(RF_accu
racyAverage_20_80))
```

```
RT_accuracyTestList (80% train, 20% test) partition mean: 0.90697674
4186
RT_accuracyTestList (50% train, 50% test) partition mean: 0.91823899
3711
RT_accuracyTestList (20% train, 80% test) partition mean: 0.85996055
2268
```

In [109]:

```
displayAccuracies('Random Forest', 'Seeds Data', RF_accuracyAverage_80_20, RF_ac
curacyAverage_50_50, RF_accuracyAverage_20_80)
printAccuracies('RT', RF_accuracyTestList_80_20, RF_accuracyTestList_50_50, RF_a
ccuracyTestList_20_80)
```




```
Accuracy of RT's 3 trials on (80% train, 20% test) partition :[0.906
97674418604646, 0.88372093023255816, 0.93023255813953487]
Mean Accuracy of RT on (80% train, 20% test) partition: 0.9069767441
86

Accuracy of RT's 3 trials on (50% train, 50% test) partition :[0.915
09433962264153, 0.93396226415094341, 0.90566037735849059]
Mean Accuracy of RT on (50% train, 50% test) partition: 0.9182389937
11

Accuracy of RT's 3 trials on (20% train, 80% test) partition:[0.8520
7100591715978, 0.82840236686390534, 0.89940828402366868]
Mean Accuracy of RT on (20% train, 80% test) partition: 0.8599605522
68
```

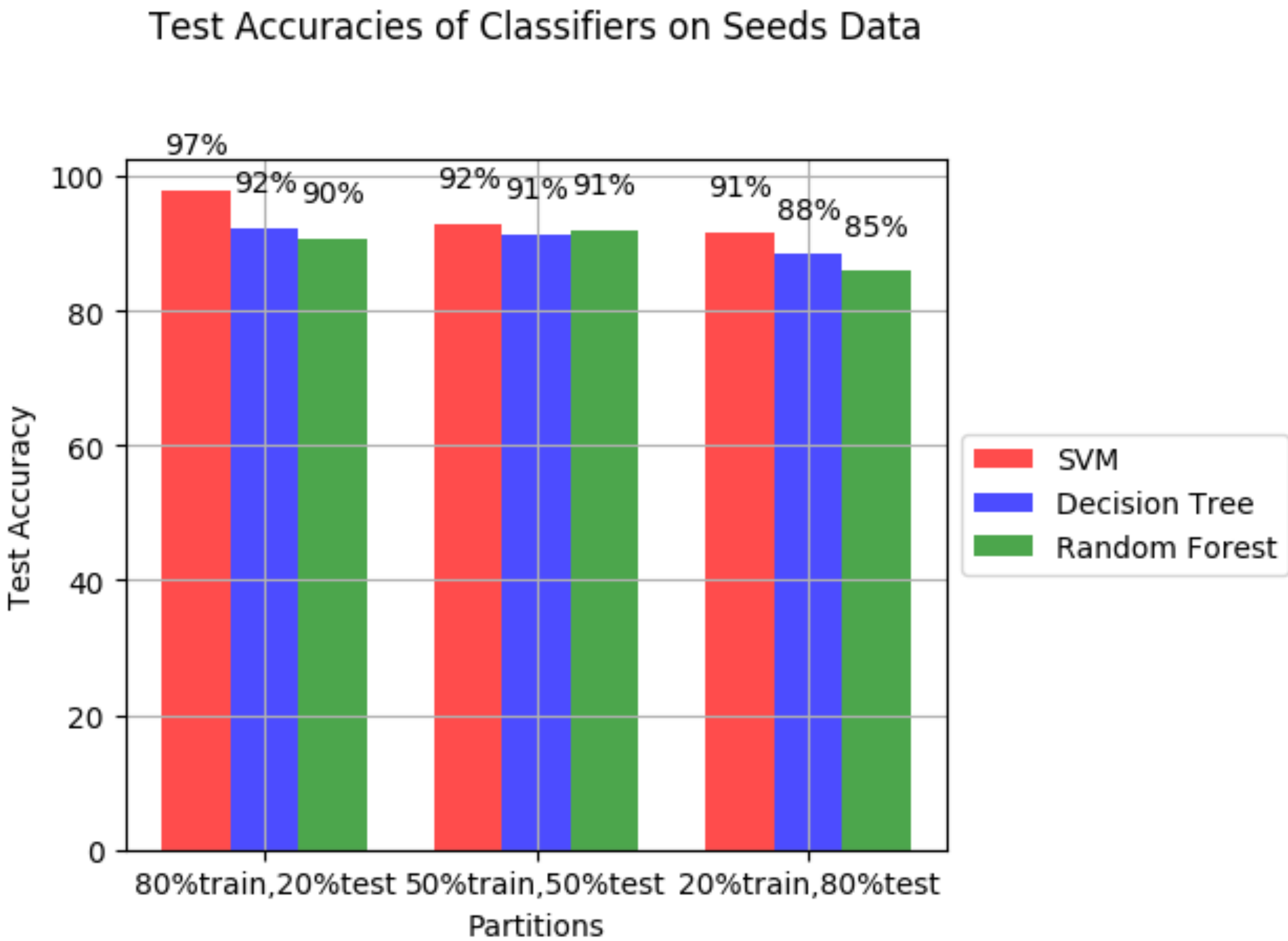
Results

In [103]:

```
def autolabel(rects):
    """
    Attach a text label above each bar displaying its height
    """
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
                #'%d' % int(height),
                '%d' % int(height) + '%',
                ha='center', va='bottom')
```

In [112]:

```
import numpy as np
import matplotlib.pyplot as plt

# data to plot
n_groups = 3
SVM_partitions = (SVM_accuracyAverage_80_20*100, SVM_accuracyAverage_50_50*100,
SVM_accuracyAverage_20_80*100)
DT_partitions = (DT_accuracyAverage_80_20*100, DT_accuracyAverage_50_50*100, DT_
accuracyAverage_20_80*100)
RT_partitions = (RF_accuracyAverage_80_20*100, RF_accuracyAverage_50_50*100, RF_
accuracyAverage_20_80*100)

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = .25
opacity = .7

SVM = plt.bar(index, SVM_partitions, bar_width,#align='center',
                 alpha=opacity,
                 color='r',
                 label='SVM')

DT = plt.bar(index + bar_width, DT_partitions, bar_width,#align='center',
                 alpha=opacity,
                 color='b',
                 label='Decision Tree')

RT = plt.bar(index + bar_width + bar_width, RT_partitions, bar_width,#align='cen
ter',
                 alpha=opacity,
                 color='g',
                 label='Random Forest')

plt.xlabel('Partitions')
plt.ylabel('Test Accuracy')
plt.title('Test Accuracies of Classifiers on Seeds Data', y=1.15)
plt.xticks(index + bar_width, ('80%train,20%test', '50%train,50%test', '20%train
,80%test'))
#plt.legend()
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

autolabel(SVM)
autolabel(DT)
autolabel(RT)

plt.tight_layout()
plt.grid()
plt.show()
```


Test Accuracies of Classifiers on Seeds Data
Test Accuracy
Partitions
80%train,20%test
50%train,50%test
20%train,80%test
97%
92%
90%
92%
91%
91%
91%
88%
85%
0
20
40
60
80
100
SVM
Decision Tree
Random Forest