EEG Eye State Data Set

The data set consists of 14 EEG values and a value indicating the eye state. Eye state equal to '1' indicates the eye is closed and '0' indicates the eye is open.

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/EEGEyeState/E
EGEyeState.csv')
```

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: (14980, 15)
```

Out[3]:

	@AF3	@ F2	@ F3	@FC5	@T7	@P7	@O1	@O2	@P8	@
0	4329.23	4009.23	4289.23	4148.21	4350.26	4586.15	4096.92	4641.03	4222.05	4238
1	4324.62	4004.62	4293.85	4148.72	4342.05	4586.67	4097.44	4638.97	4210.77	4226
2	4327.69	4006.67	4295.38	4156.41	4336.92	4583.59	4096.92	4630.26	4207.69	4222
3	4328.72	4011.79	4296.41	4155.90	4343.59	4582.56	4097.44	4630.77	4217.44	4235
4	4326.15	4011.79	4292.31	4151.28	4347.69	4586.67	4095.90	4627.69	4210.77	4244

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 EEGState Eye Data Set.
print('Number of NULL values in df: ' + str(df.isnull().sum().sum()))

uniqueClasses = df['eyeDetection'].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['eyeDetection']==i).sum()))
```

```
Number of NULL values in df: 0
Number of unique classes in df: (2,)
Class 0 count: 8257
Class 1 count: 6723
```

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]:

```
denominator = 4
```

In [9]:

```
df_array1 = np.array(df_shuffle1)              #Convert dataframe to array in or
der to slice into X and Y.
#Reduce the number of rows by a certain fraction. This is to reduce the run time
.
cutNumber = int(df_array1.shape[0]/denominator)
df_array1 = df_array1[0:cutNumber, :]
print(df_array1.shape)

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))
```

```
(3745, 15)
X1 shape: (3745, 14)
Y1 shape: (3745,)
```

In [10]:

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

#Reduce the number of rows by a certain fraction. This is to reduce the run time
.
cutNumber = int(df_array2.shape[0]/denominator)
df_array2 = df_array2[0:cutNumber, :]
print(df_array2.shape)

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))
```

```
(3745, 15)
X2 shape: (3745, 14)
Y2 shape: (3745,)
```

In [11]:

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

#Reduce the number of rows by a certain fraction. This is to reduce the run time
.
cutNumber = int(df_array3.shape[0]/denominator)
df_array3 = df_array3[0:cutNumber, :]
print(df_array3.shape)

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))

#print(X3[:, 0])
#print(Y3)
```

```
(3745, 15)
X3 shape: (3745, 14)
Y3 shape: (3745,)
```

Functions Used For All Classifiers

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

In [12]:

```
#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 [13]:

```
#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 [14]:

```
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 [15]:

```
#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 [16]:

```
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 [18]:

```
#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 [19]:

```
#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 [20]:

```
#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 [21]:

```
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 [22]:

```
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)
```

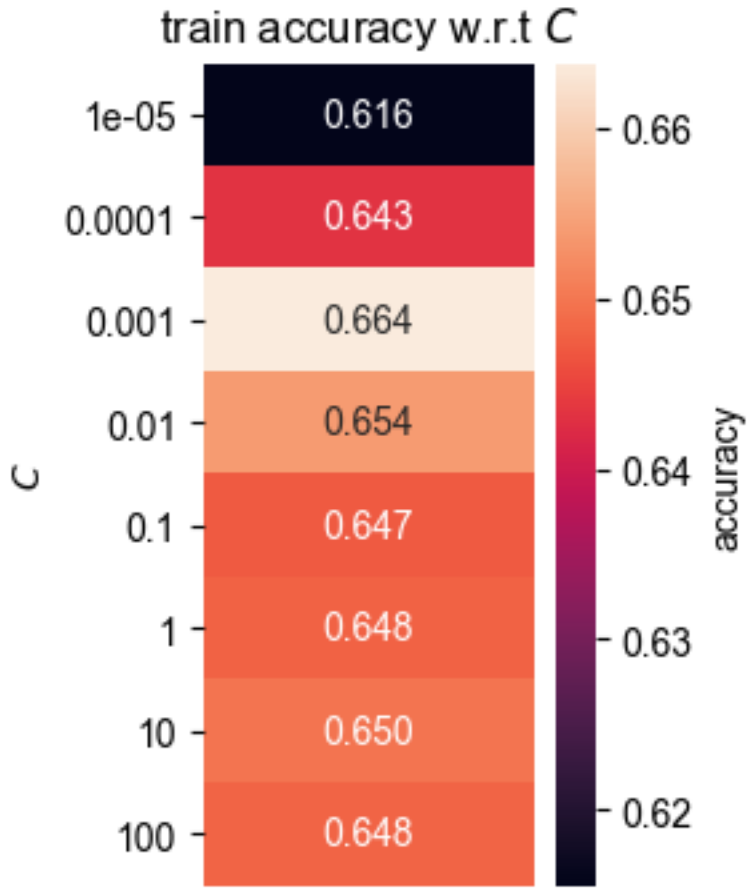
1st Run)

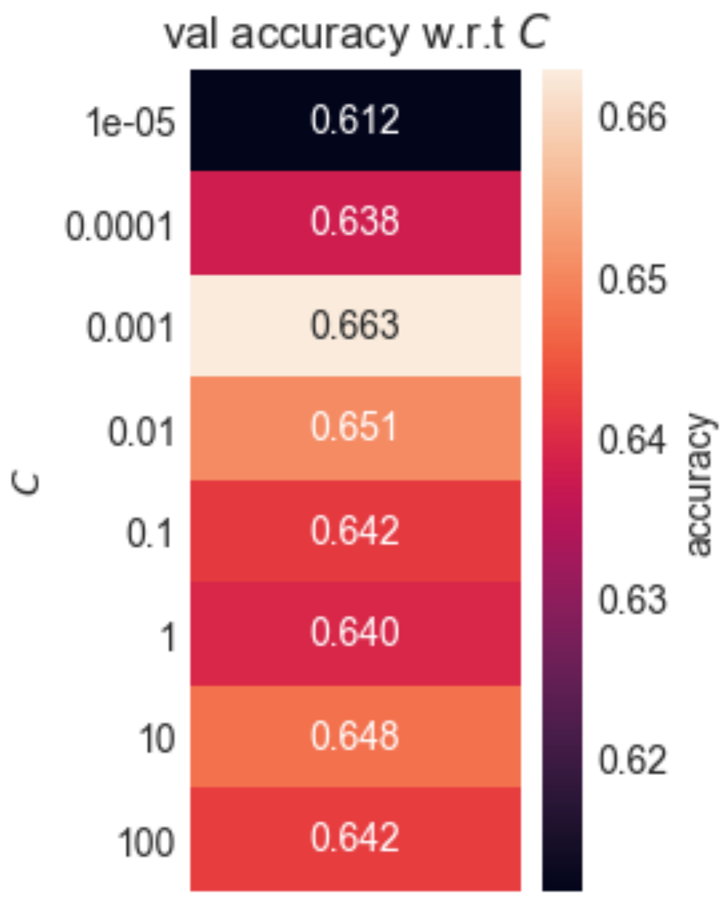
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [23]:

```
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)
```

```
[ 0.6155623   0.64311755  0.66381134  0.65394671  0.64708546  0.6479
3893
  0.64968124  0.64816102]
[ 0.61181575  0.63785047  0.66288385  0.65053405  0.64185581  0.6395
1936
  0.64753004  0.64218959]
```





In [24]:

```
#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 0.662883845127 from index 2.
Best C: 0.001
Test Accuracy Score: 0.624833110814
```

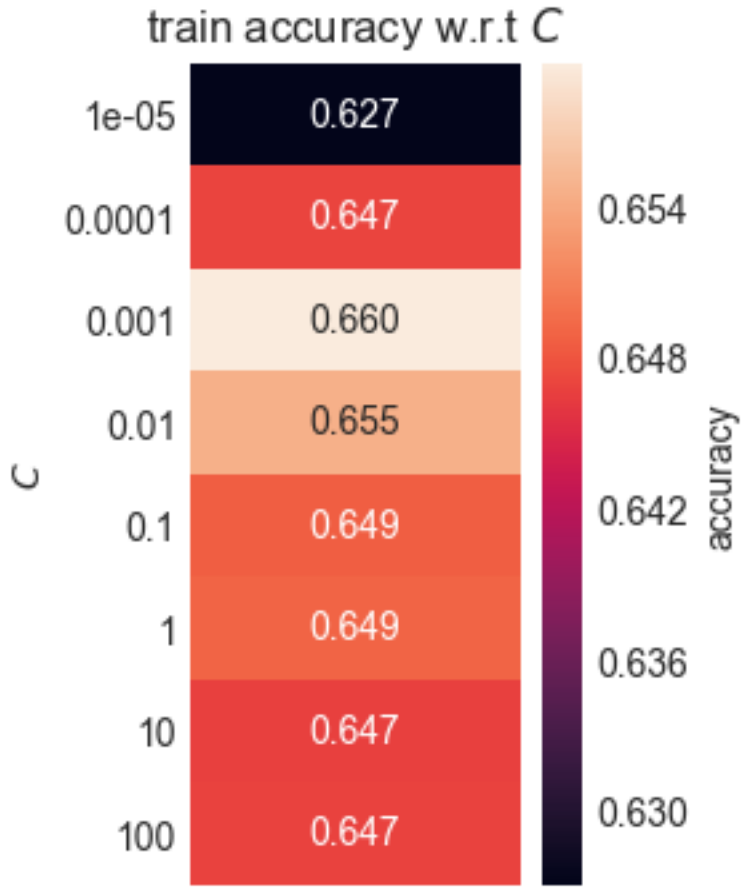
2nd Run)

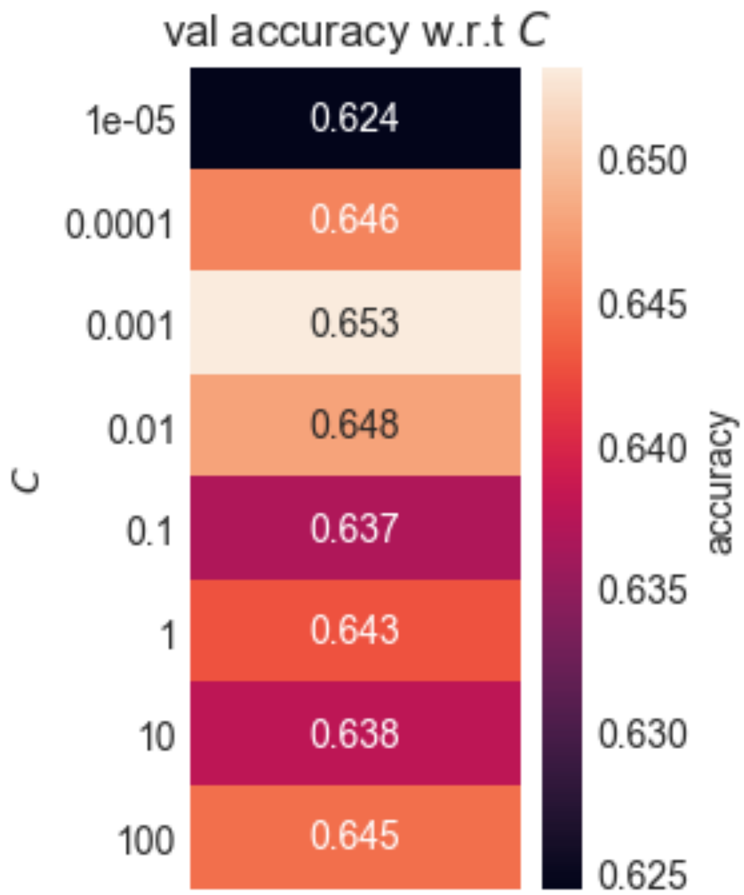
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [25]:

```
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)
```

```
[ 0.62705782  0.6469741   0.65973199  0.65468783  0.64871644  0.6490
5109
  0.64678795  0.64686354]
[ 0.62449933  0.64586115  0.65320427  0.64786382  0.63684913  0.6428
5714
  0.63785047  0.64452603]
```


train accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.627
0.647
0.660
0.655
0.649
0.649
0.647
0.647
0.654
0.648
0.642
0.636
0.630
accuracy


val accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.624
0.646
0.653
0.648
0.637
0.643
0.638
0.645
0.650
0.645
0.640
0.635
0.630
0.625
accuracy

In [26]:

```
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 0.653204272363 from index 2.
Best C: 0.001
Test Accuracy Score: 0.667556742323
```

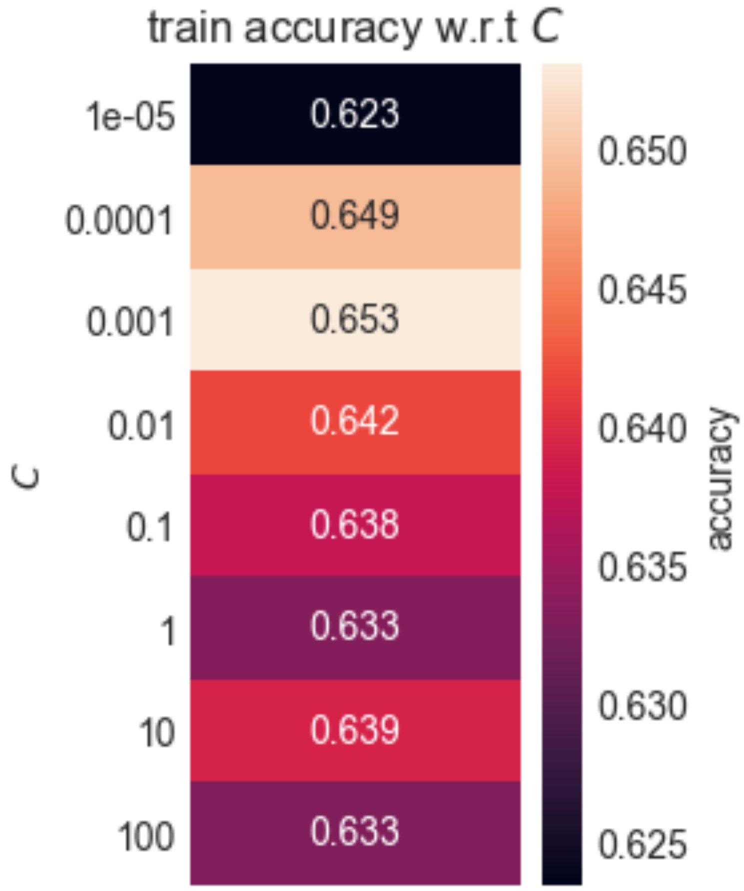
3rd Run)

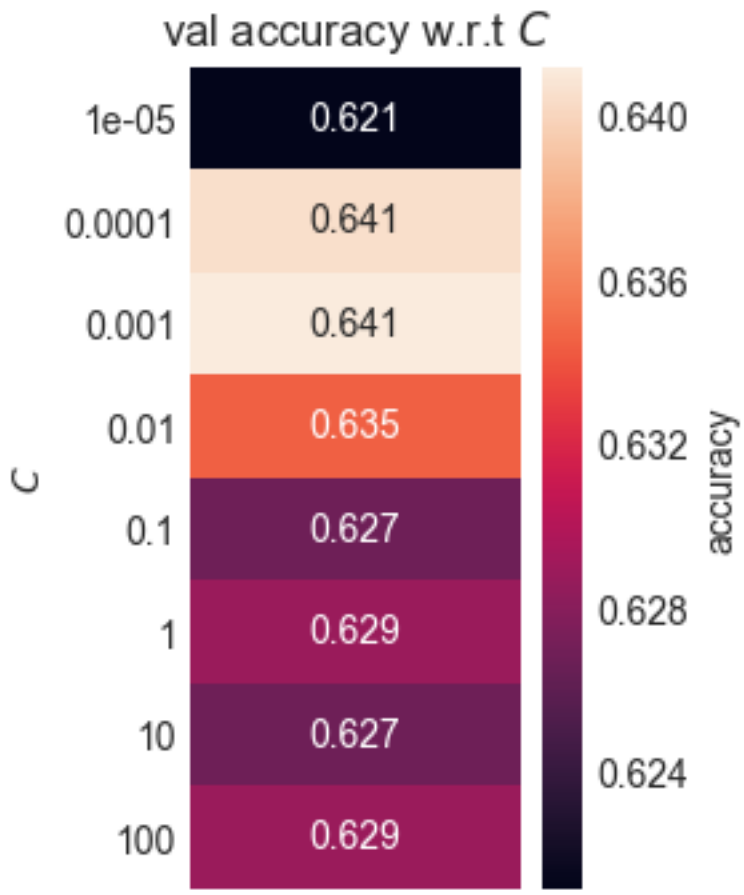
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [27]:

```
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)
```

```
[ 0.6234981   0.6493096   0.65305497  0.64163233  0.6377016   0.6333
6225
  0.63918507  0.6330279 ]
[ 0.62116155  0.64052069  0.64118825  0.63451268  0.62683578  0.6288
3845
  0.62683578  0.62883845]
```


train accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.623
0.649
0.653
0.642
0.638
0.633
0.639
0.633
0.650
0.645
0.640
0.635
0.630
0.625
accuracy


val accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.621
0.641
0.641
0.635
0.627
0.629
0.627
0.629
0.640
0.636
0.632
0.628
0.624
accuracy

In [28]:

```
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 0.641188251001 from index 2.
Best C: 0.001
Test Accuracy Score: 0.662216288385
```

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

In [29]:

```
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:[0.6248331108144192 8, 0.6675567423230974 7, 0.66
221628838451263]
SVM_accuracyTestList mean: 0.651535380507
```

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 [30]:

```
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)
```

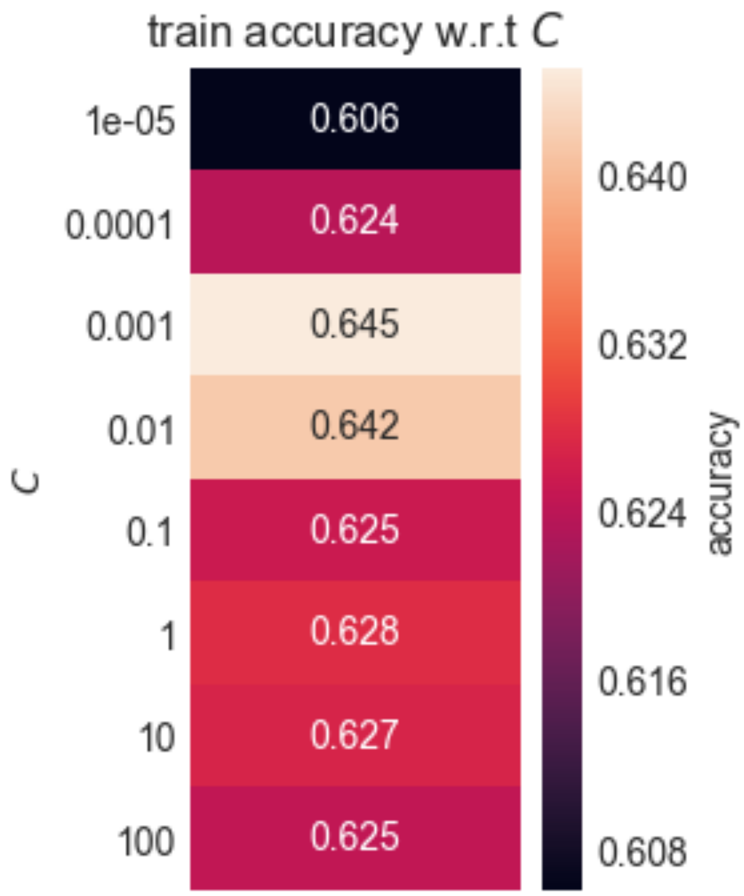
1st Run)

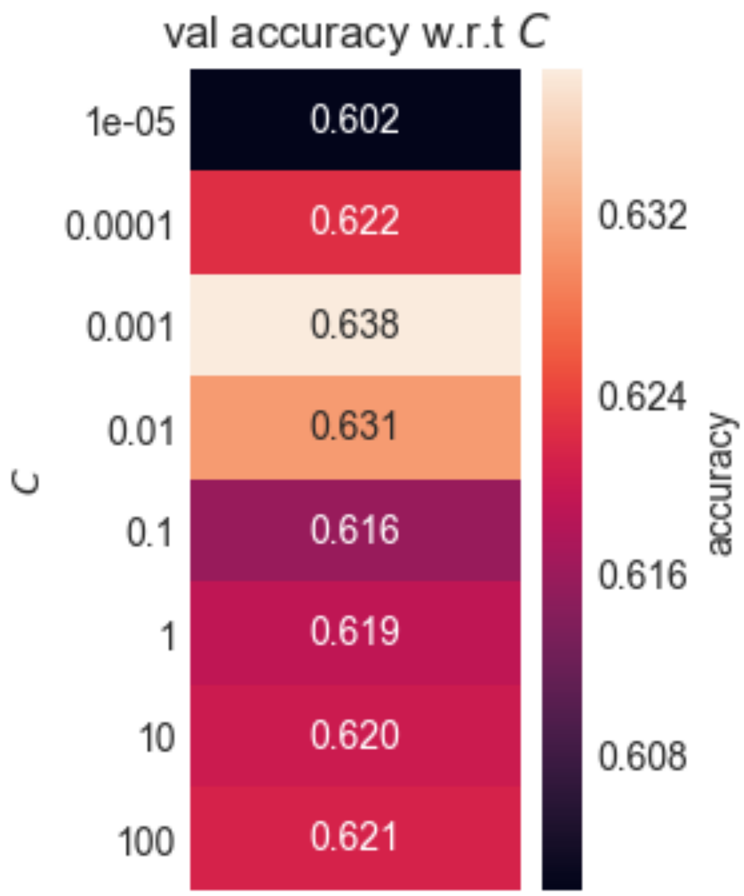
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [31]:

```
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)
```

```
[ 0.60606624  0.6238127   0.64506231  0.6416195   0.625474    0.6278
4856
  0.62689946  0.62464392]
[ 0.60202991  0.62232906  0.6383547   0.63087607  0.6159188   0.6191
2393
  0.62019231  0.62126068]
```





In [32]:

```
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 0.638354700855 from index 2.
Best C: 0.001
Test Accuracy Score: 0.647090229578
```

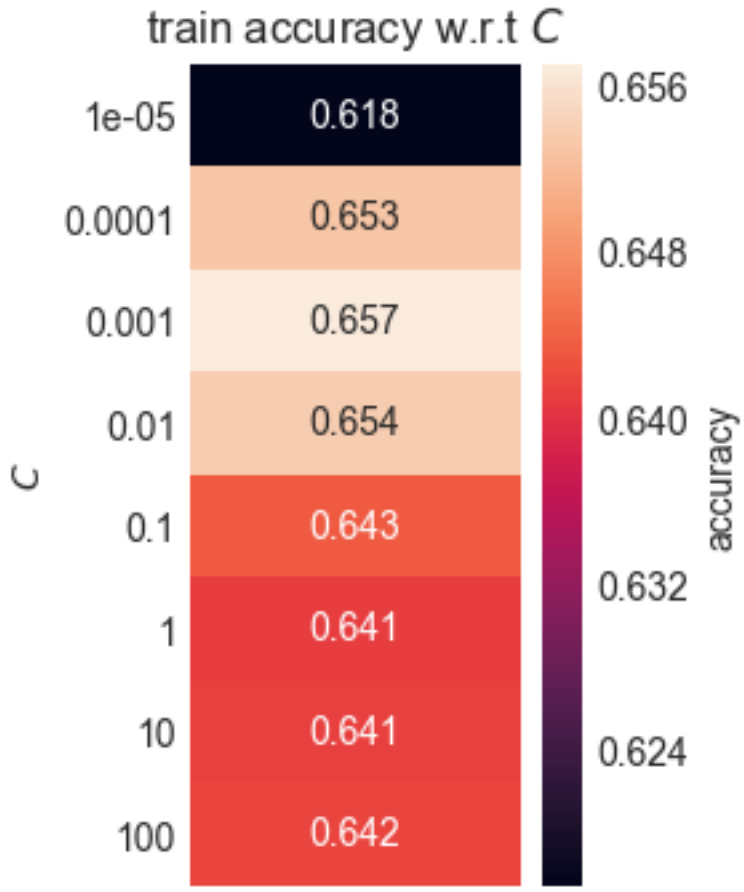
2nd Run)

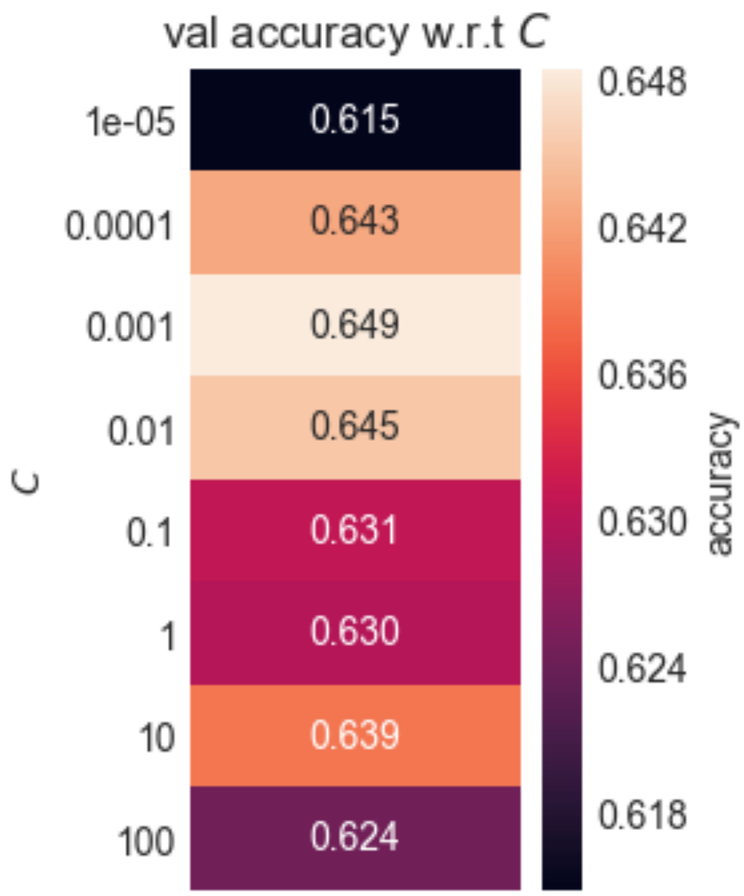
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [33]:

```
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)
```

```
[ 0.61752113  0.65307443  0.65705186  0.65384732  0.64346082  0.6410
248
  0.641383    0.64161637]
[ 0.61485043  0.64262821  0.64850427  0.64529915  0.63087607  0.6298
0769
  0.63888889  0.62446581]
```





In [34]:

```
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 0.648504273504 from index 2.
Best C: 0.001
Test Accuracy Score: 0.665242925788
```

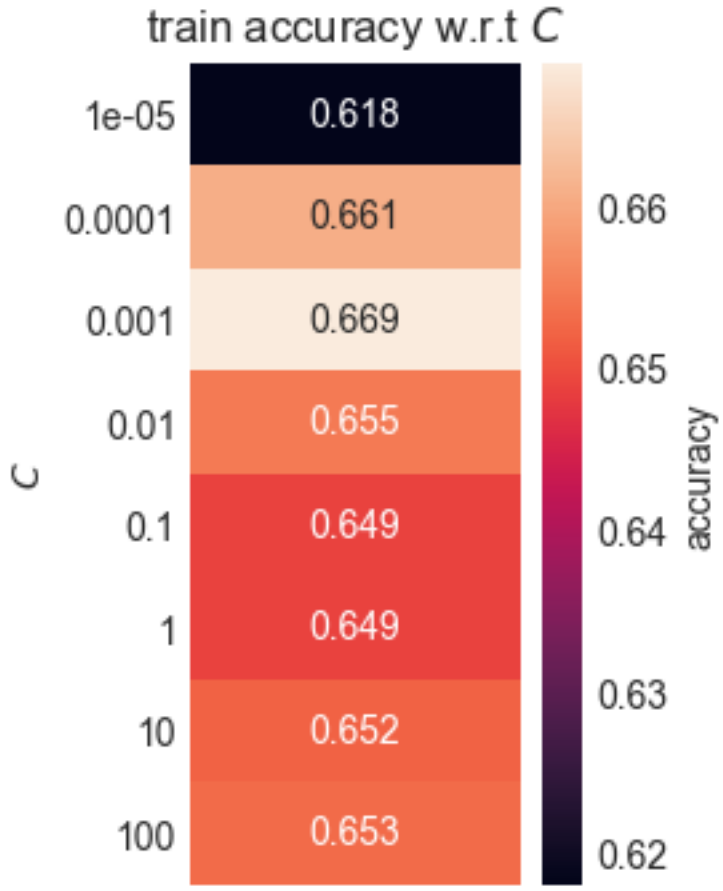
3rd Run)

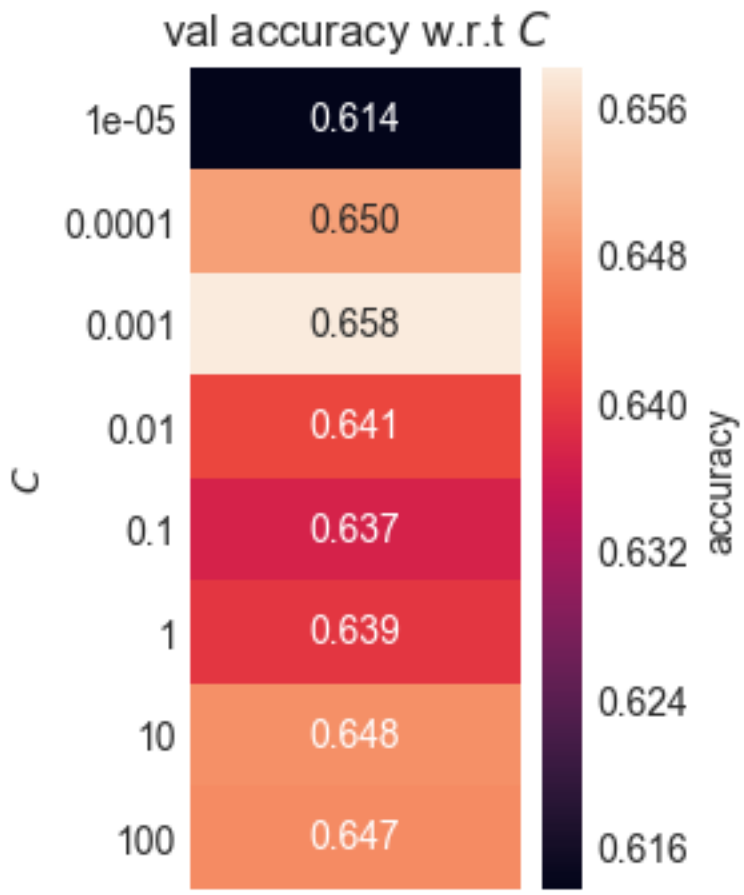
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [35]:

```
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)
```

```
[ 0.61811492  0.66084961  0.66886215  0.65473664  0.64897823  0.6489
1952
  0.65218432  0.65313381]
[ 0.61378205  0.64957265  0.65811966  0.64102564  0.63728632  0.6394
2308
  0.64797009  0.6474359 ]
```


train accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.618
0.661
0.669
0.655
0.649
0.649
0.652
0.653
0.66
0.65
0.64
0.63
0.62
accuracy


val accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.614
0.650
0.658
0.641
0.637
0.639
0.648
0.647
0.656
0.648
0.640
0.632
0.624
0.616
accuracy

In [36]:

```
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 0.65811965812 from index 2.
Best C: 0.001
Test Accuracy Score: 0.654030966364
```

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

In [37]:

```
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:[0.64709022957821671, 0.66524292578750666, 0.65
403096636412172]
SVM_accuracyTestList mean: 0.655454707243
```

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 [38]:

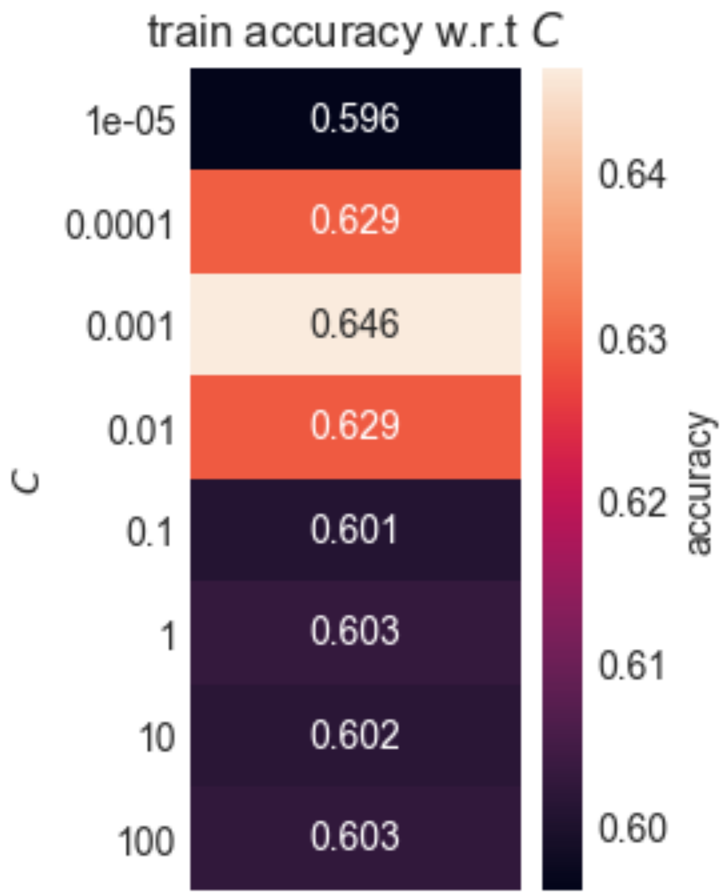
```
X1_train_val, Y1_train_val, X1_test, Y1_test = partitionData(X1, Y1, 0.2)
X2_train_val, Y2_train_val, X2_test, Y2_test = partitionData(X2, Y2, 0.2)
X3_train_val, Y3_train_val, X3_test, Y3_test = partitionData(X3, Y3, 0.5)
```

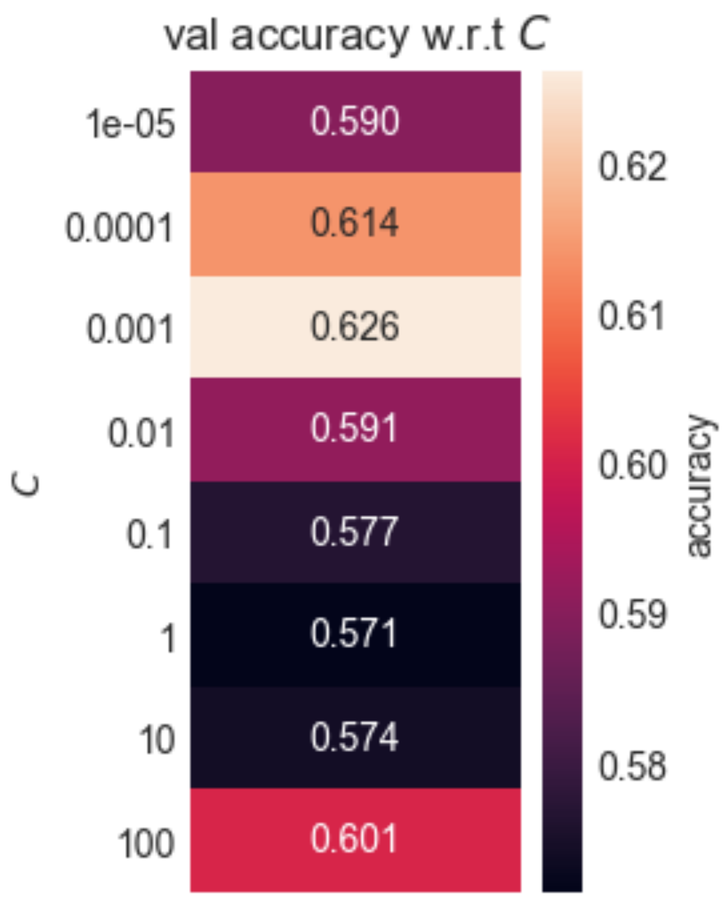
1st Run)

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

In [39]:

```
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 [40]:

```
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))
```

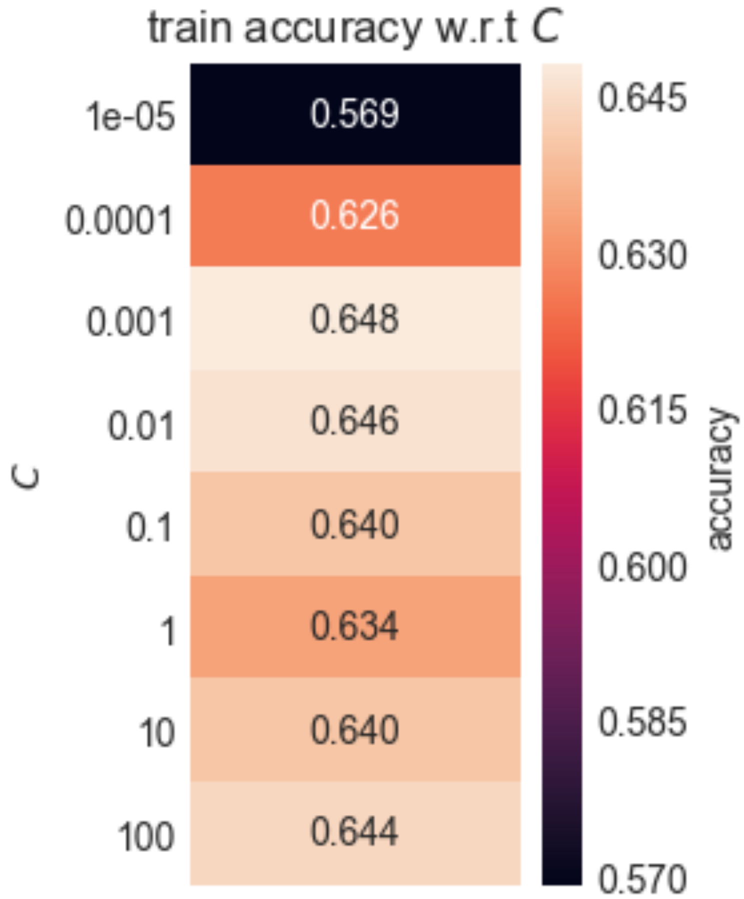
```
Largest value in accuracyValidation is 0.626168224299 from index 2.
Best C: 0.001
Test Accuracy Score: 0.644192256342
```

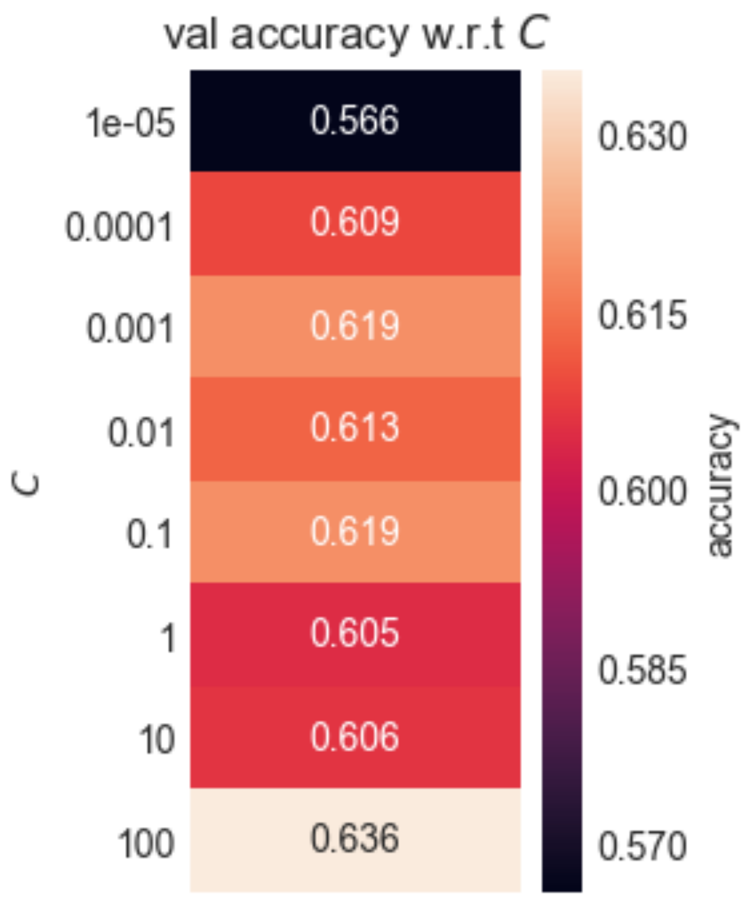
2nd Run)

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

In [41]:

```
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
0.569
0.626
0.648
0.646
0.640
0.634
0.640
0.644
0.645
0.630
0.615
0.600
0.585
0.570
accuracy


val accuracy w.r.t C
C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
0.566
0.609
0.619
0.613
0.619
0.605
0.606
0.636
0.630
0.615
0.600
0.585
0.570
accuracy

In [42]:

```
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 0.635514018692 from index 7.
Best C: 100
Test Accuracy Score: 0.629506008011
```

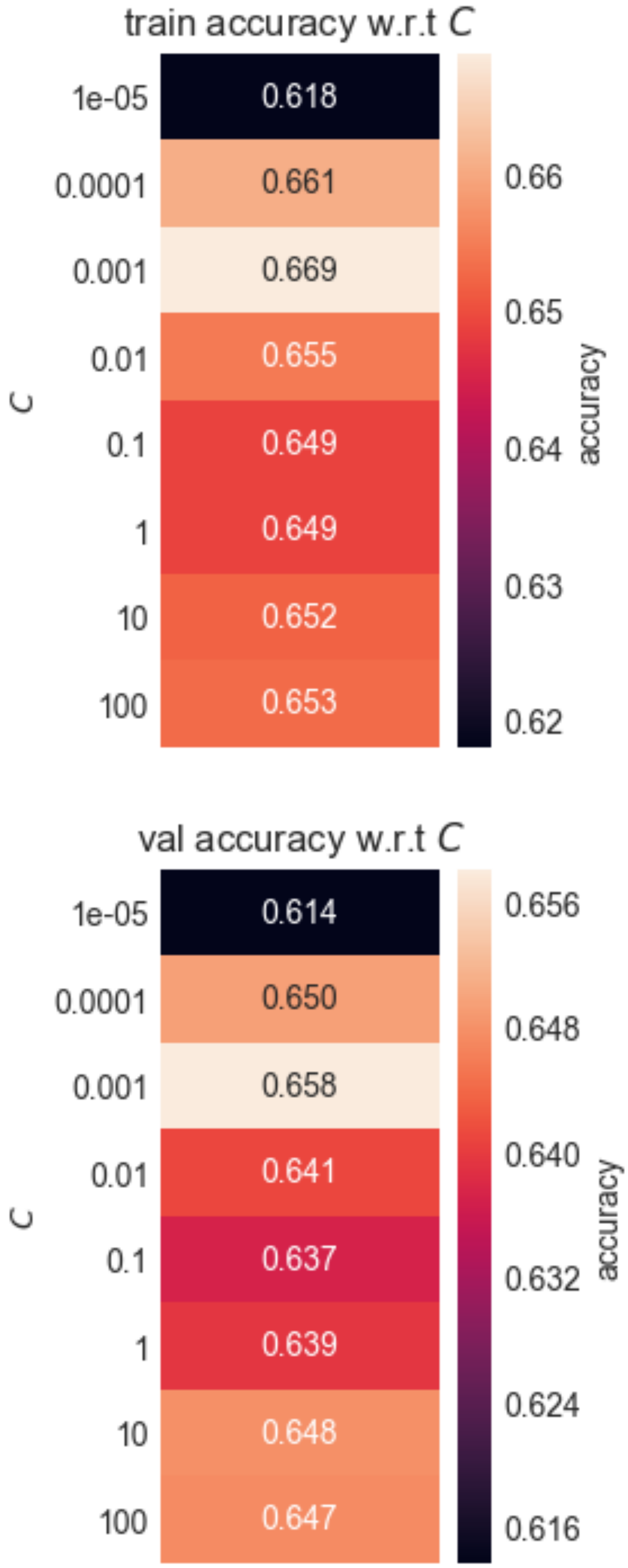
3rd Run)

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

In [43]:

```
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)
```

```
[ 0.61811492  0.66084961  0.66886215  0.65473664  0.64897823  0.6489
1952
  0.65218432  0.65313381]
[ 0.61378205  0.64957265  0.65811966  0.64102564  0.63728632  0.6394
2308
  0.64797009  0.6474359 ]
```


train accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.618
0.661
0.669
0.655
0.649
0.649
0.652
0.653
0.66
0.65
0.64
0.63
0.62
accuracy
val accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.614
0.650
0.658
0.641
0.637
0.639
0.648
0.647
0.656
0.648
0.640
0.632
0.624
0.616
accuracy

In [44]:

```
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 0.65811965812 from index 2.
Best C: 0.001
Test Accuracy Score: 0.654030966364
```

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

In [45]:

```
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:[0.644192256341789, 0.62950600801068091, 0.6540
3096636412172]
SVM_accuracyTestList mean: 0.642576410239
```

Results of SVM

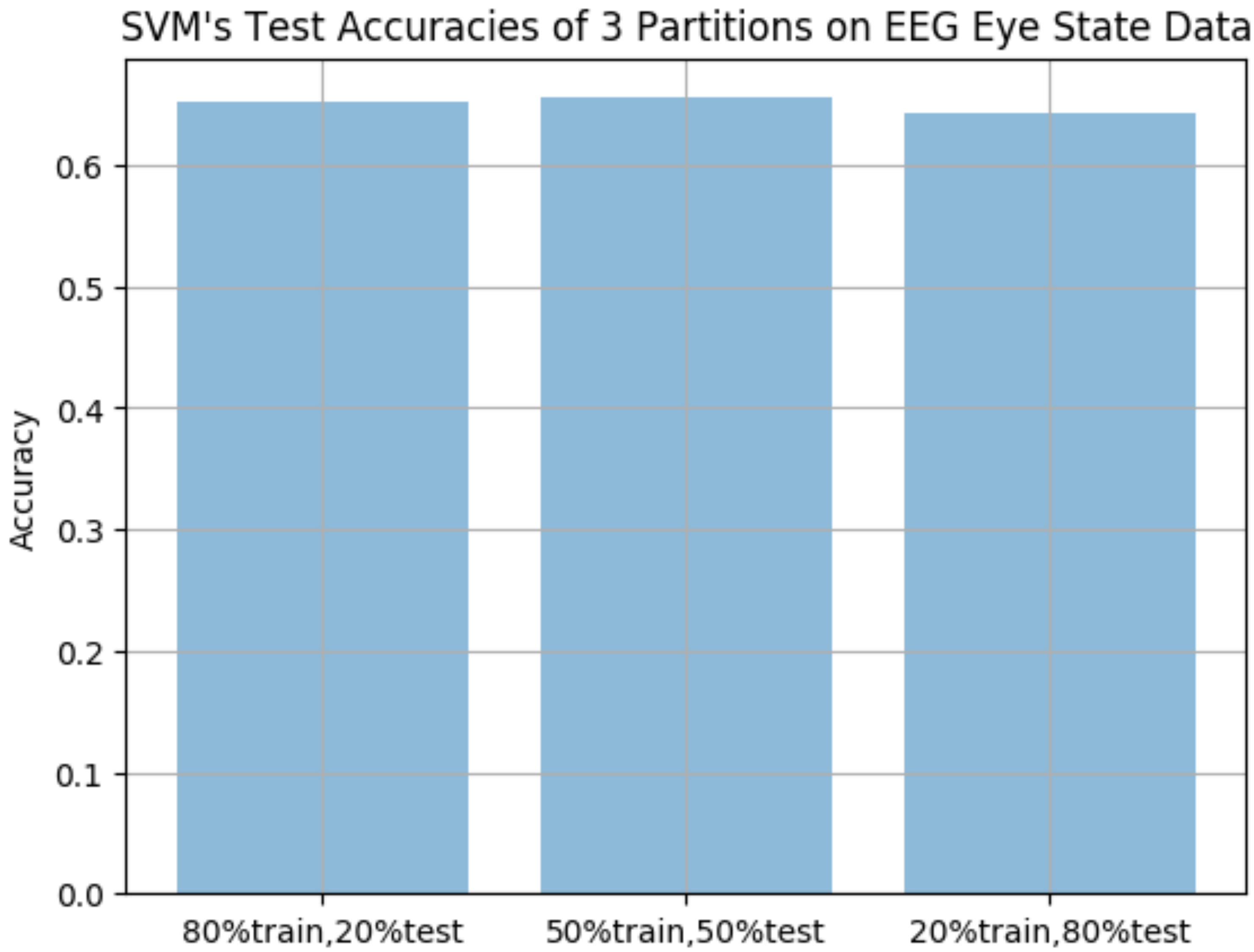
In [46]:

```
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: 0.6515353
80507
SVM_accuracyTestList (50% train, 50% test) partition mean: 0.6554547
07243
SVM_accuracyTestList (20% train, 80% test) partition mean: 0.6425764
10239
```

In [108]:

```
displayAccuracies('SVM', 'EEG Eye State Data', SVM_accuracyAverage_80_20, SVM_ac
curacyAverage_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 :[0.62
483311081441928, 0.66755674232309747, 0.66221628838451263]
Mean Accuracy of SVM on (80% train, 20% test) partition: 0.651535380
507

Accuracy of SVM's 3 trials on (50% train, 50% test) partition :[0.64
709022957821671, 0.66524292578750666, 0.65403096636412172]
Mean Accuracy of SVM on (50% train, 50% test) partition: 0.655454707
243

Accuracy of SVM's 3 trials on (20% train, 80% test) partition:[0.644
192256341789, 0.62950600801068091, 0.65403096636412172]
Mean Accuracy of SVM on (20% train, 80% test) partition: 0.642576410
239
```

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 [48]:

```
#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 [49]:

```
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 [50]:

```
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)
```

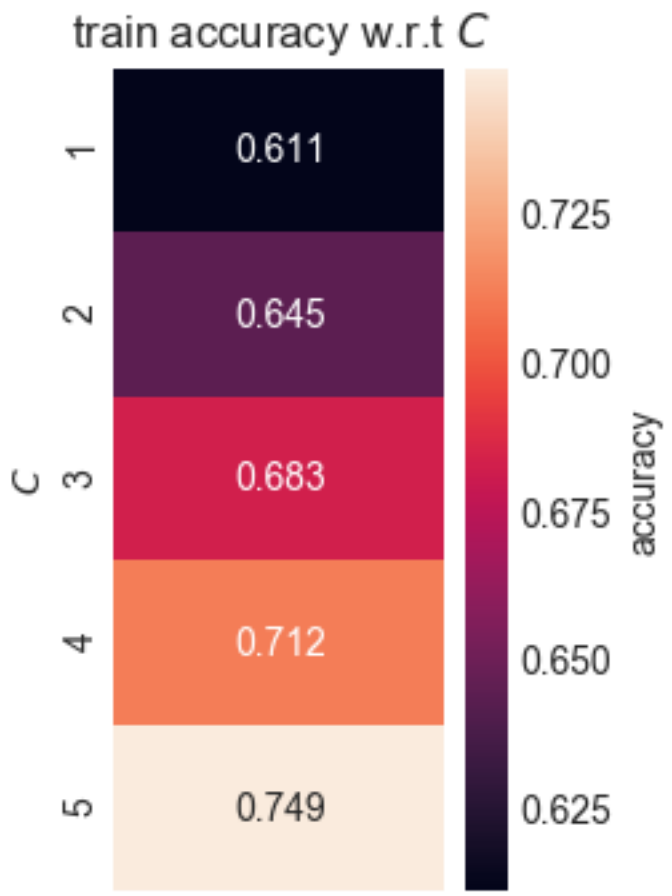
1st Run)

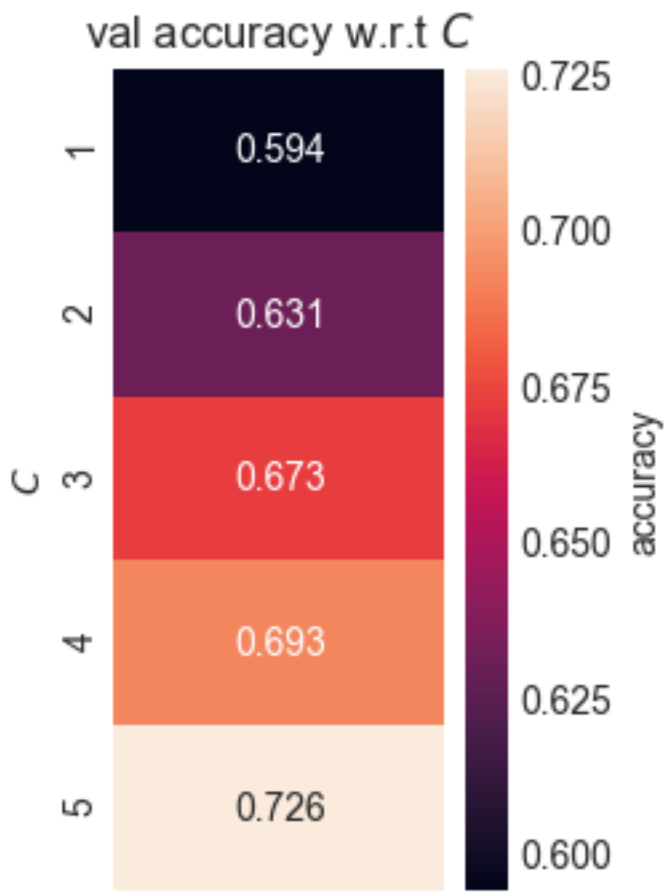
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [51]:

```
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)
```

```
[ 0.61133379  0.6445637   0.68254119  0.71180185  0.74922198]
[ 0.5941255   0.63050734  0.6728972   0.6929239   0.72596796]
```





In [52]:

```
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 0.725967957276 from index 4.
Best D: 5
Test Accuracy Score: 0.691588785047
```

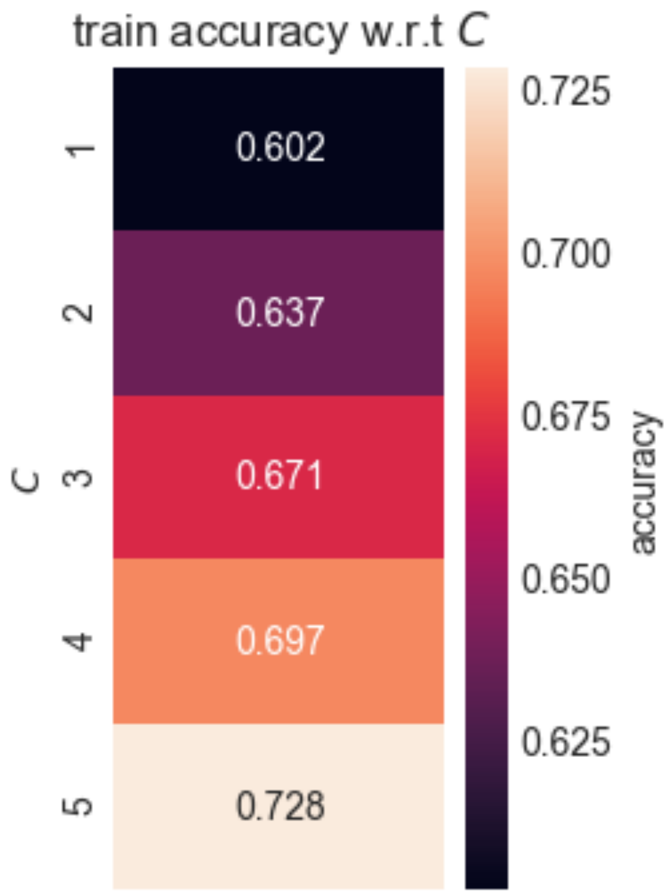
2nd Run)

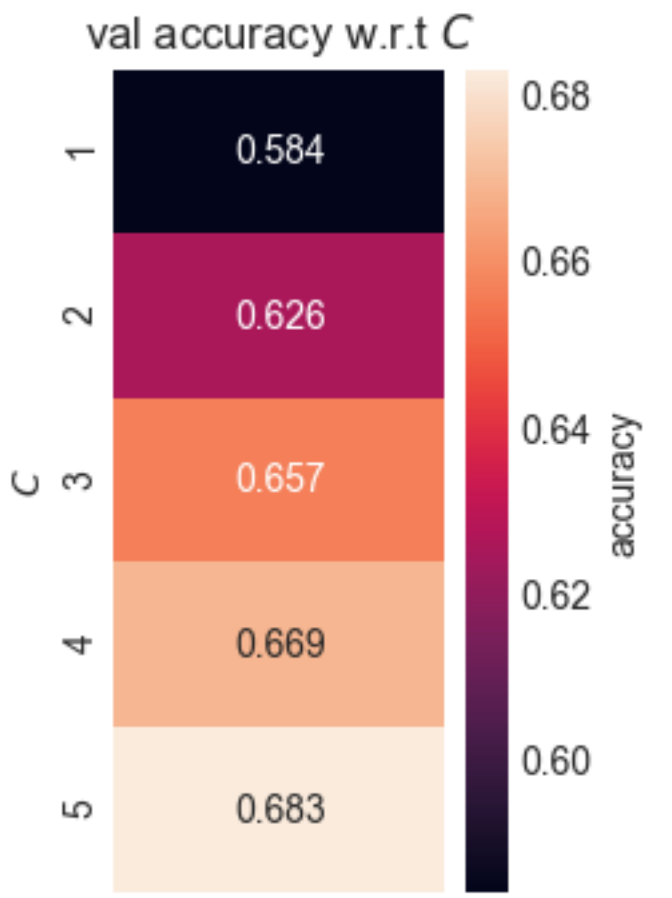
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [53]:

```
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)
```

```
[ 0.60232181  0.63718358  0.67104229  0.69718874  0.72830559]
[ 0.58411215  0.62550067  0.65654206  0.66922563  0.68291055]
```





In [54]:

```
#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 0.682910547397 from index 4.
Best D: 5
Test Accuracy Score: 0.696929238985
```

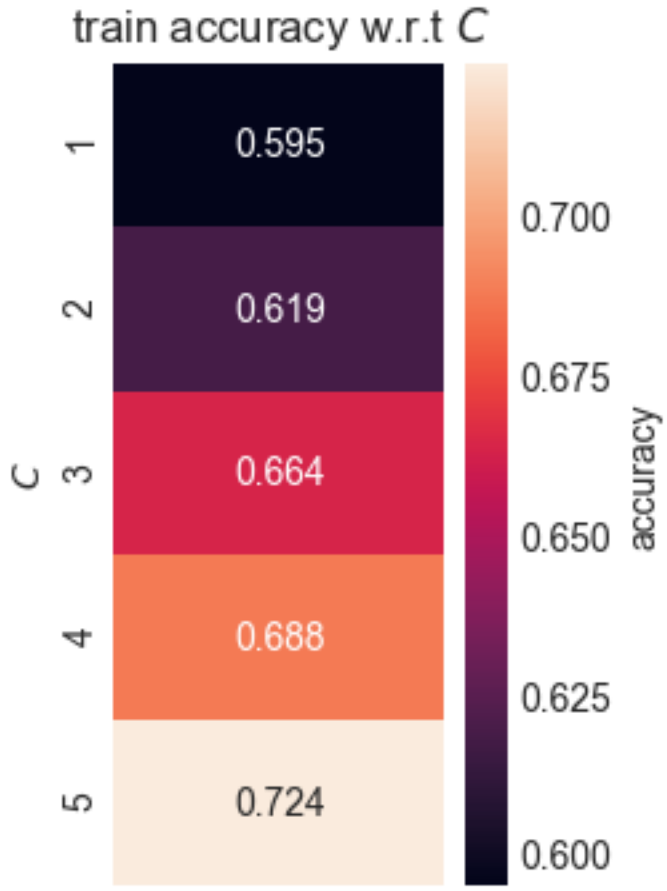
3rd Run)

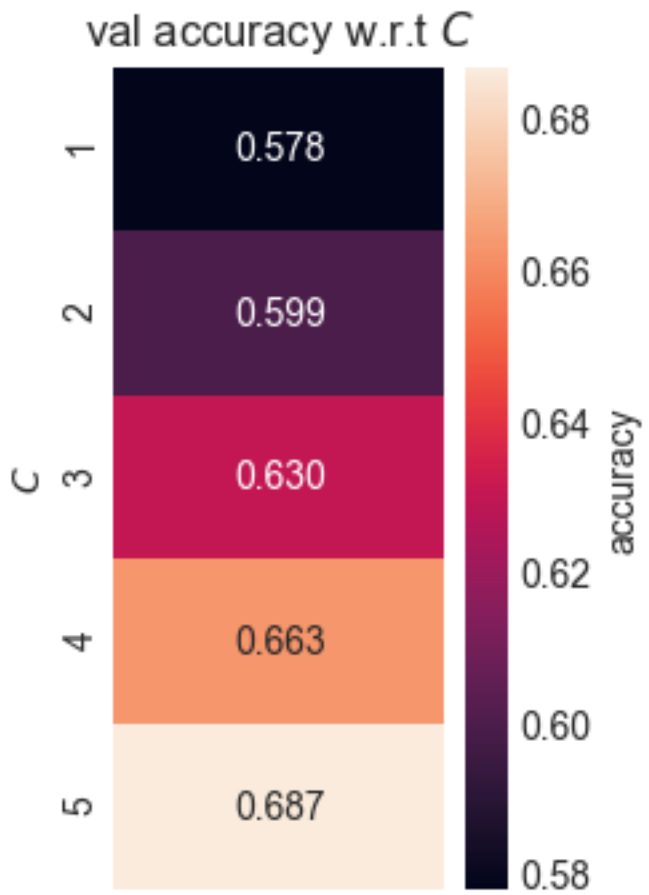
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [55]:

```
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)
```

```
[ 0.59523762  0.61882661  0.66410903  0.6881035   0.72396635]
[ 0.57810414  0.59946595  0.63017356  0.66321762  0.68691589]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.595
0.619
0.664
0.688
0.724
0.700
0.675
0.650
0.625
0.600
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.578
0.599
0.630
0.663
0.687
0.68
0.66
0.64
0.62
0.60
0.58
accuracy

In [56]:

```
#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 0.68691588785 from index 4.
Best D: 5
Test Accuracy Score: 0.702269692924
```

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

In [57]:

```
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:[0.69158878504672894, 0.69692923898531378, 0.702
26969292389851]
DT_accuracyTestList mean: 0.696929238985
```

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 [58]:

```
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)
```

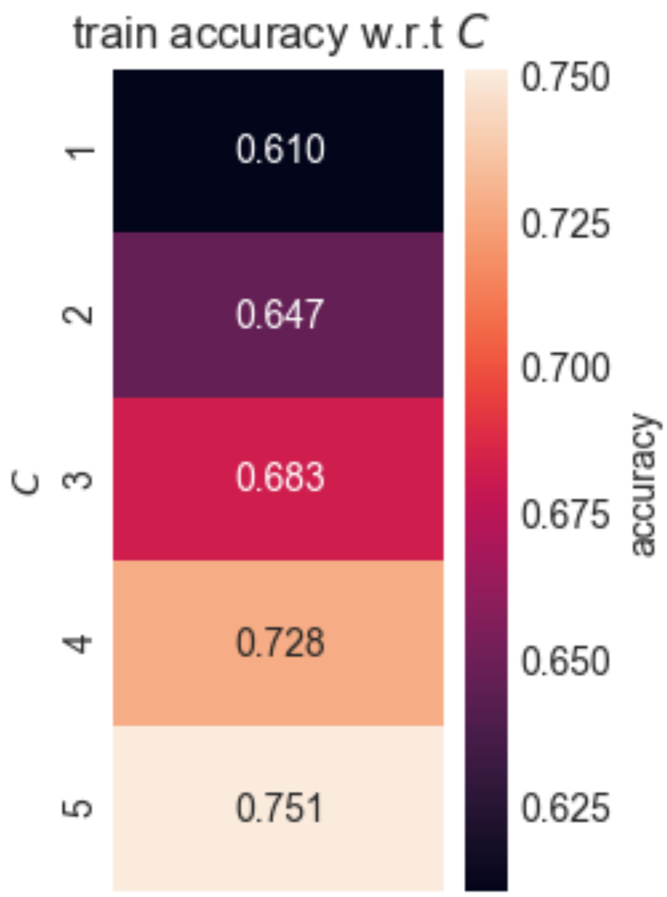
1st Run)

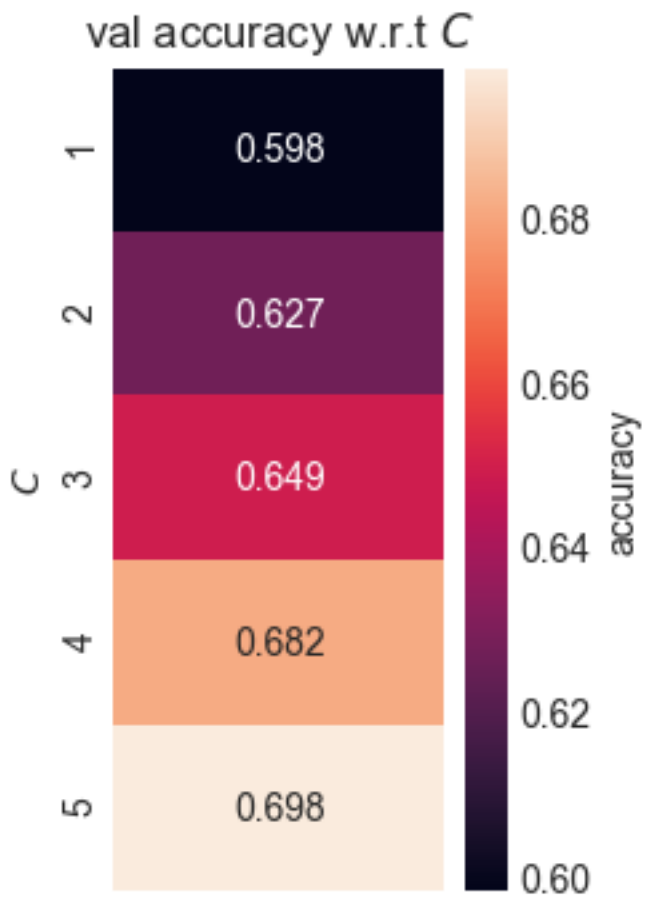
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [59]:

```
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)
```

```
[ 0.61028007  0.64702087  0.68251418  0.72821719  0.75095033]
[ 0.5982906   0.62713675  0.64903846  0.68162393  0.69818376]
```





In [60]:

```
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 0.698183760684 from index 4.
Best D: 5
Test Accuracy Score: 0.720234917245
```

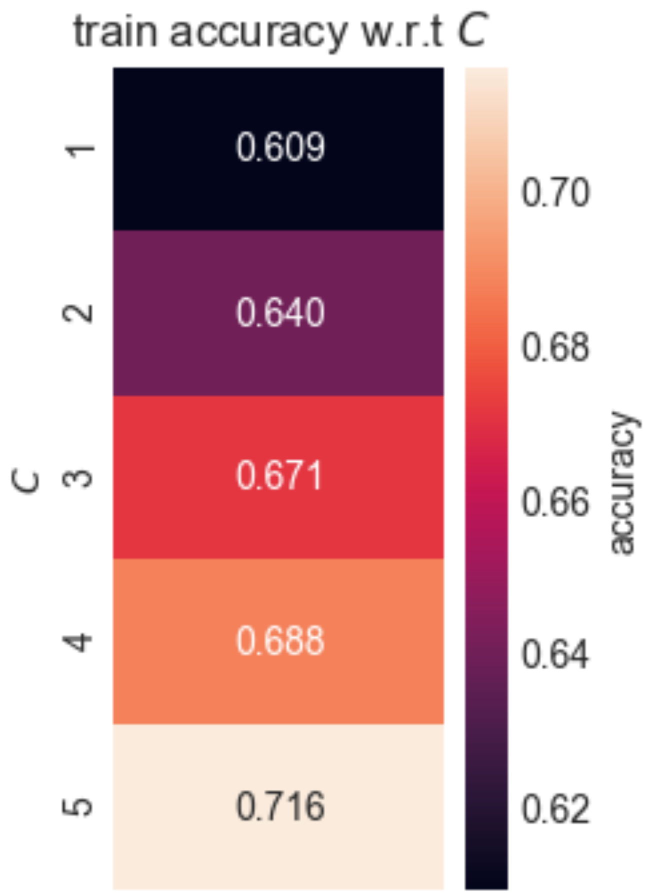
2nd Run)

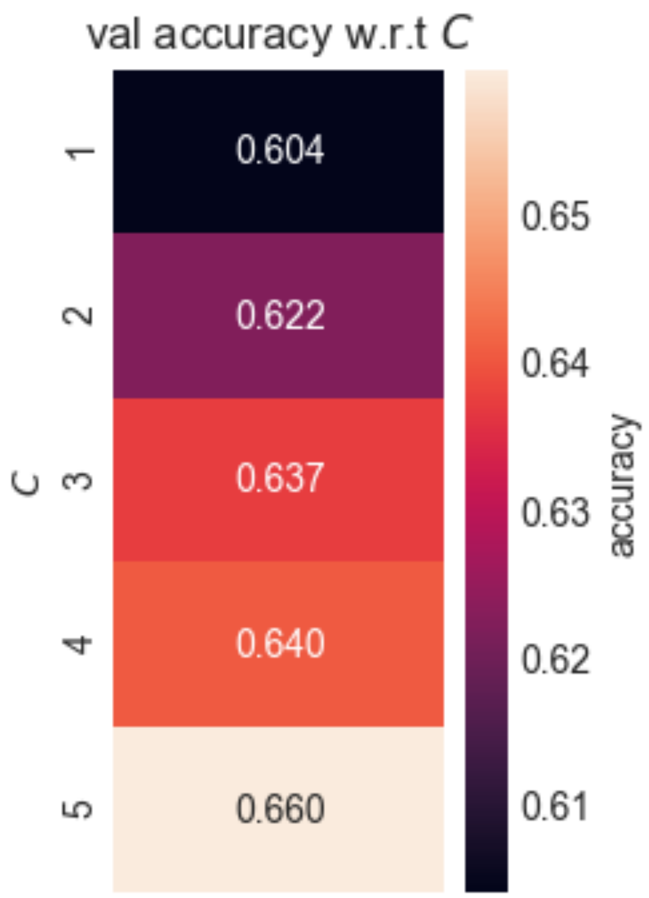
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [61]:

```
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)
```

```
[ 0.60939031  0.64007665  0.67135731  0.68791459  0.71598768]
[ 0.60416667  0.62232906  0.63728632  0.64049145  0.65972222]
```





In [62]:

```
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 0.659722222222 from index 4.
Best D: 5
Test Accuracy Score: 0.659369994661
```

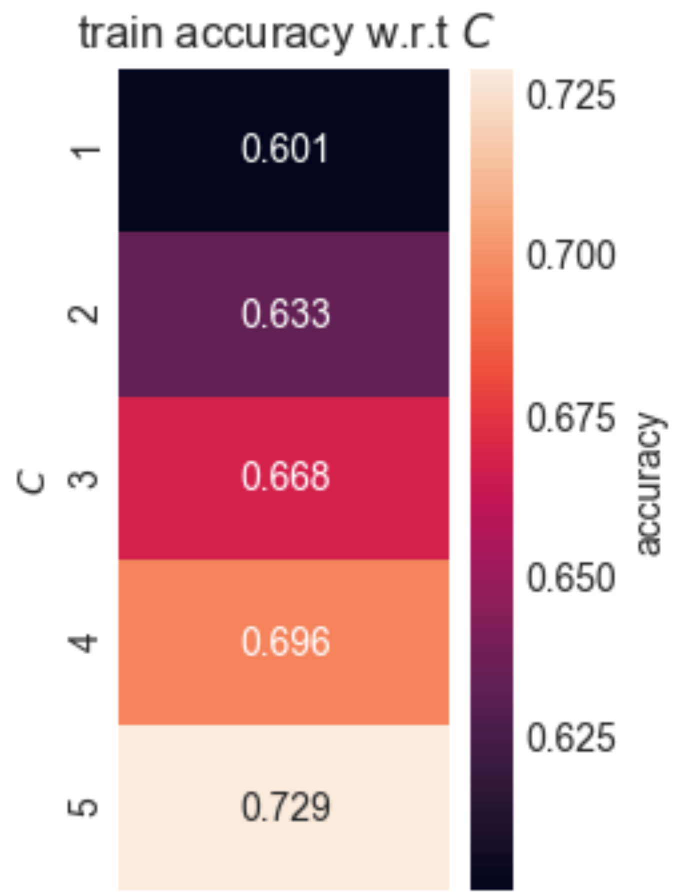
3rd Run)

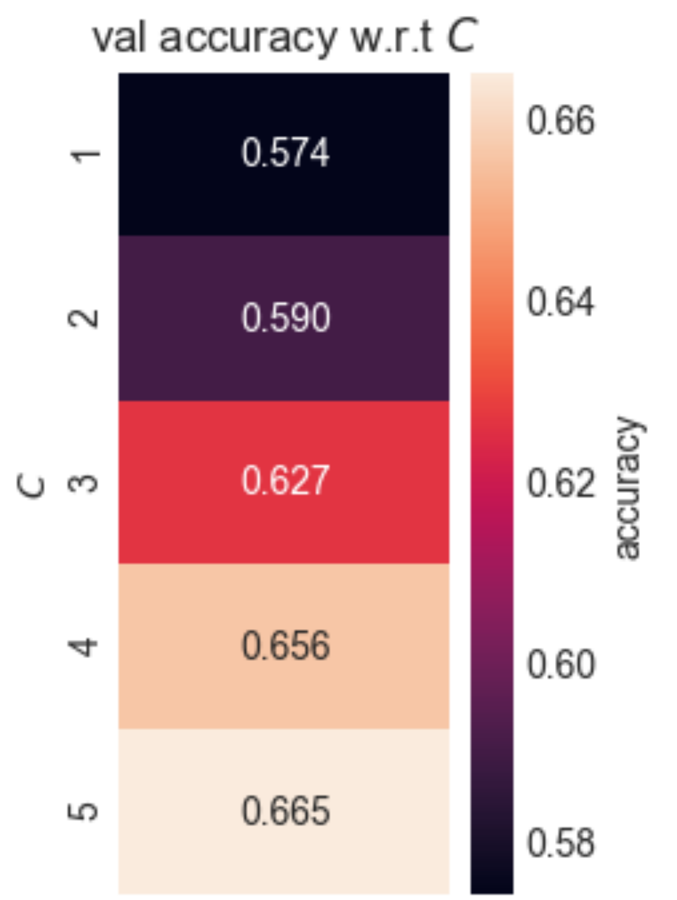
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [63]:

```
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)
```

```
[ 0.60107984  0.63313261  0.66838949  0.69569204  0.72875174]
[ 0.57425214  0.59027778  0.62660256  0.65598291  0.6650641 ]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.601
0.633
0.668
0.696
0.729
0.725
0.700
0.675
0.650
0.625
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.574
0.590
0.627
0.656
0.665
0.66
0.64
0.62
0.60
0.58
accuracy

In [64]:

```
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 0.665064102564 from index 4.
Best D: 5
Test Accuracy Score: 0.675920982381
```

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

In [65]:

```
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:[0.72023491724506139, 0.65936999466097168, 0.675
92098238120657]
DT_accuracyTestList mean: 0.685175298096
```

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 [66]:

```
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)
```

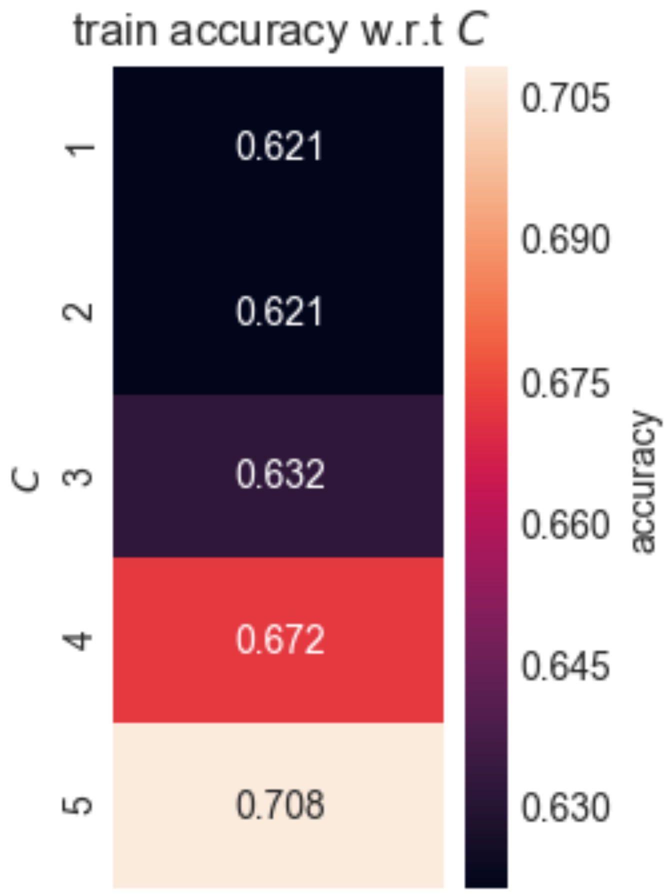
1st Run)

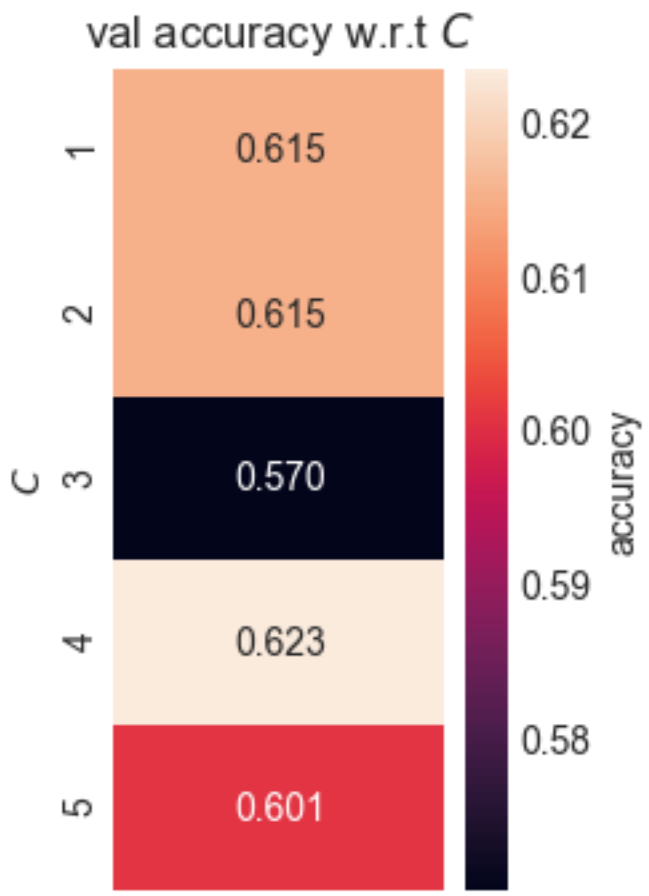
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [67]:

```
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)
```

```
[ 0.62142126  0.62142126  0.63210155  0.67230267  0.7082042 ]
[ 0.61548732  0.61548732  0.57009346  0.623498    0.60080107]
```





In [68]:

```
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 0.62349799733 from index 3.
Best D: 4
Test Accuracy Score: 0.592122830441
```

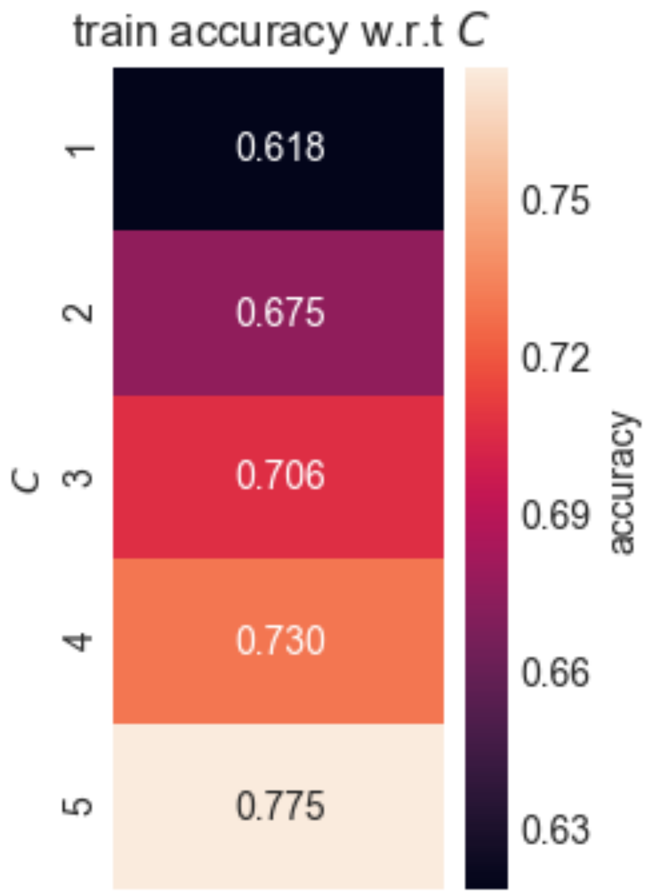
2nd Run)

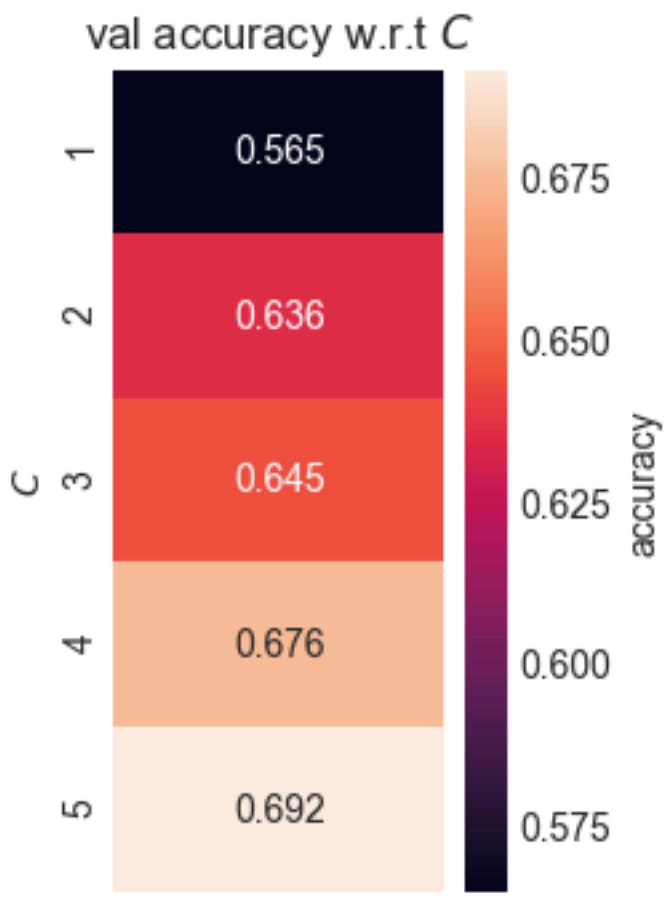
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [69]:

```
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)
```

```
[ 0.61815743  0.67467597  0.70642409  0.73031047  0.77525828]
[ 0.564753    0.63551402  0.64485981  0.67556742  0.69158879]
```





In [70]:

```
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 0.691588785047 from index 4.
Best D: 5
Test Accuracy Score: 0.681575433912
```

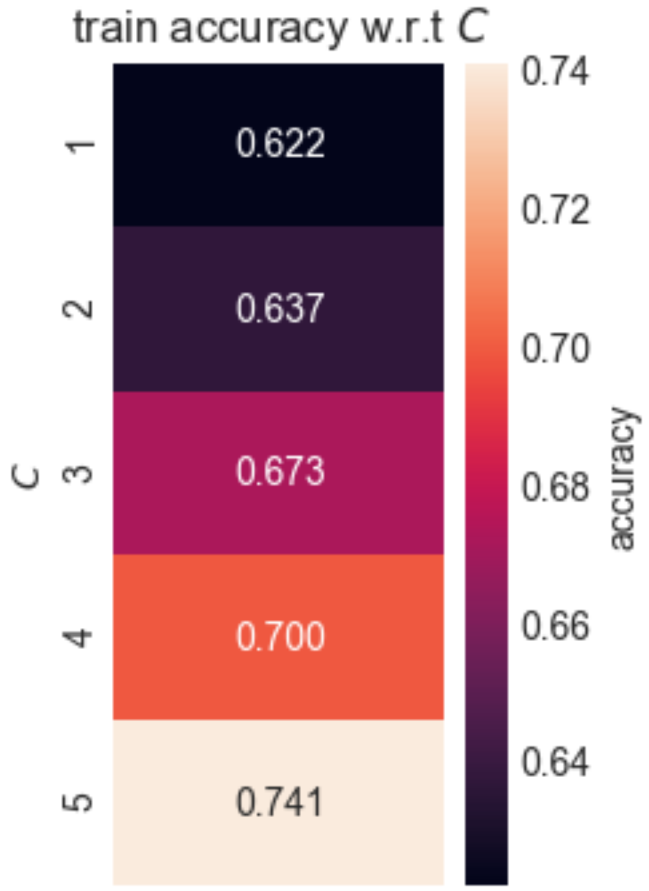
3rd Run)

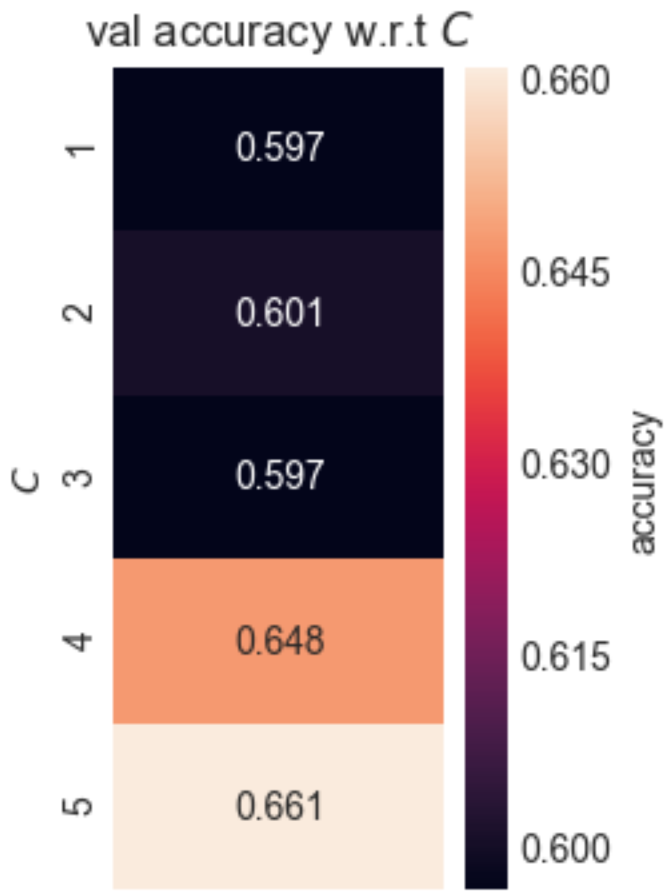
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [71]:

```
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)
```

```
[ 0.62202105  0.63699787  0.67259589  0.6996055   0.74099098]
[ 0.59679573  0.60080107  0.59679573  0.64753004  0.66088117]
```


train accuracy w.r.t C
0.622
0.637
0.673
0.700
0.741
C
1
2
3
4
5
accuracy
0.74
0.72
0.70
0.68
0.66
0.64


val accuracy w.r.t C
0.597
0.601
0.597
0.648
0.661
C
1
2
3
4
5
accuracy
0.660
0.645
0.630
0.615
0.600

In [72]:

```
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 0.6608811749 from index 4.
Best D: 5
Test Accuracy Score: 0.642857142857
```

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

In [73]:

```
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:[0.5921228304405874 4, 0.6815754339118824 7, 0.642
8571428571429]
DT_accuracyTestList mean: 0.638851802403
```

Results of Decision Tree

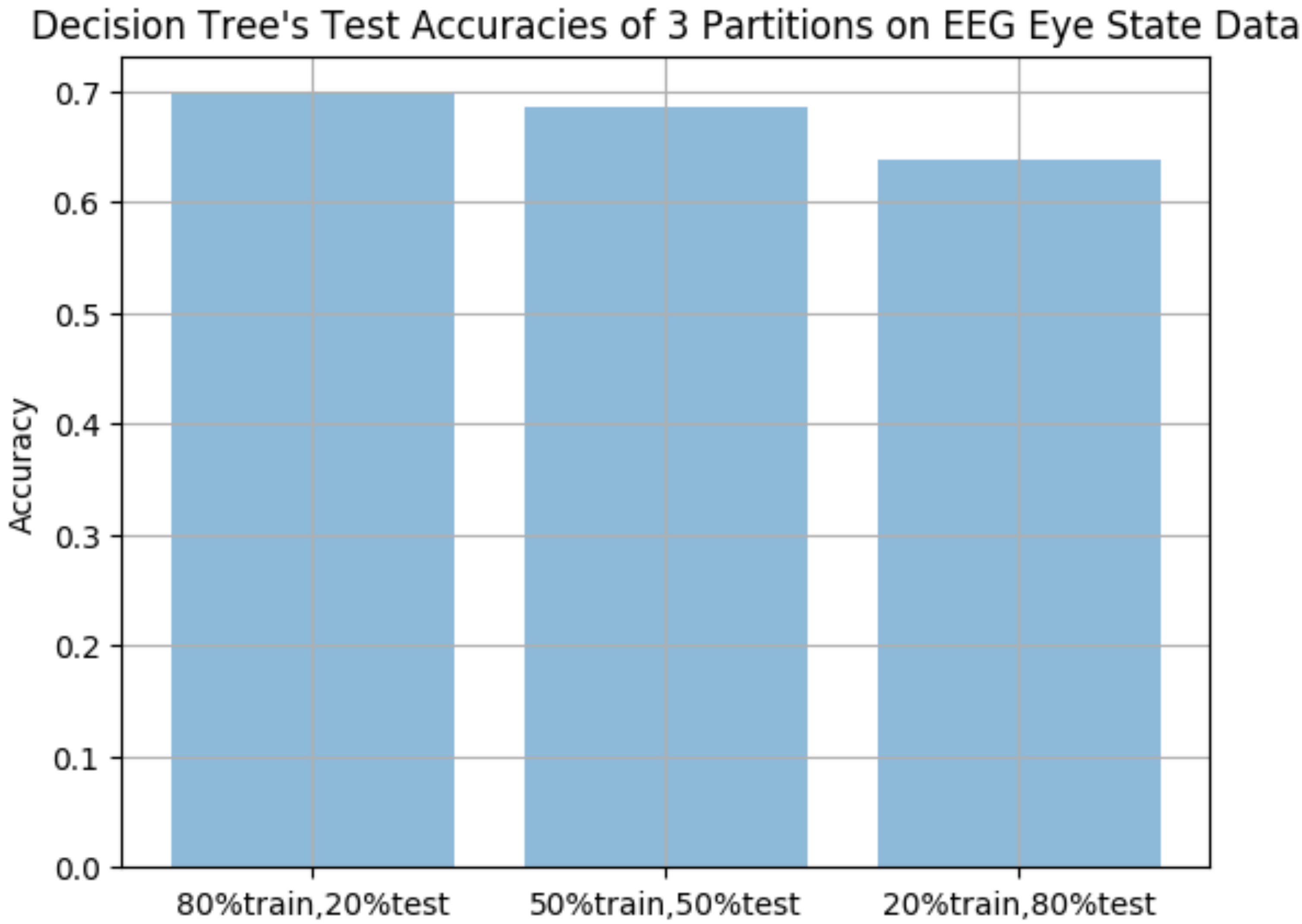
In [74]:

```
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: 0.69692923
8985
DT_accuracyTestList (50% train, 50% test) partition mean: 0.68517529
8096
DT_accuracyTestList (20% train, 80% test) partition mean: 0.63885180
2403
```

In [109]:

```
displayAccuracies('Decision Tree', 'EEG Eye State Data', DT_accuracyAverage_80_2
0, DT_accuracyAverage_50_50, DT_accuracyAverage_20_80)
printAccuracies('DT', DT_accuracyTestList_80_20, DT_accuracyTestList_50_50, DT_a
ccuracyTestList_20_80)
```




```
Accuracy of DT's 3 trials on (80% train, 20% test) partition :[0.691
58878504672894, 0.69692923898531378, 0.70226969292389851]
Mean Accuracy of DT on (80% train, 20% test) partition: 0.6969292389
85

Accuracy of DT's 3 trials on (50% train, 50% test) partition :[0.720
23491724506139, 0.65936999466097168, 0.67592098238120657]
Mean Accuracy of DT on (50% train, 50% test) partition: 0.6851752980
96

Accuracy of DT's 3 trials on (20% train, 80% test) partition:[0.5921
2283044058744, 0.68157543391188247, 0.6428571428571429]
Mean Accuracy of DT on (20% train, 80% test) partition: 0.6388518024
03
```

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 [76]:

```
#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 [77]:

```
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 [78]:

```
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)
```

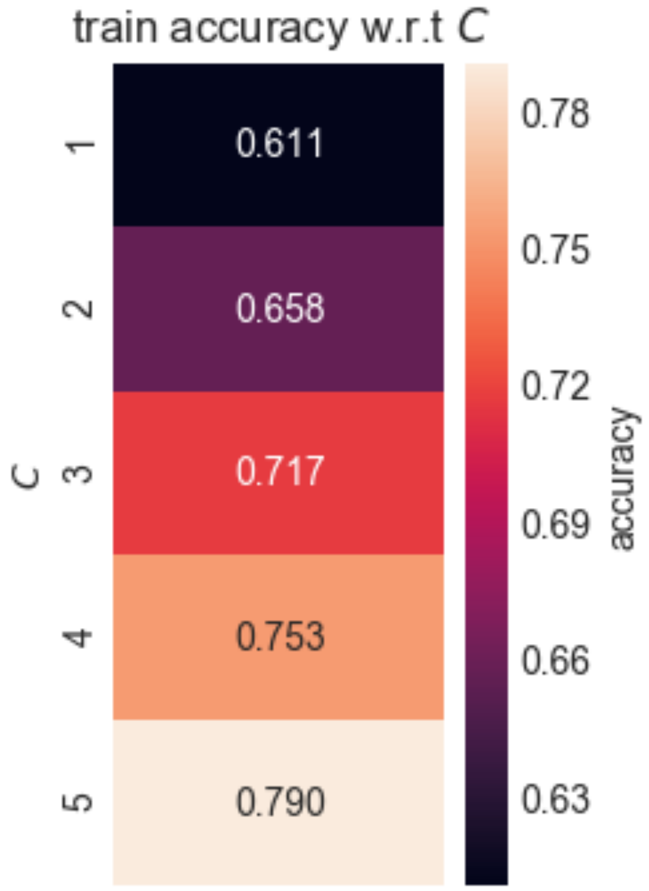
1st Run)

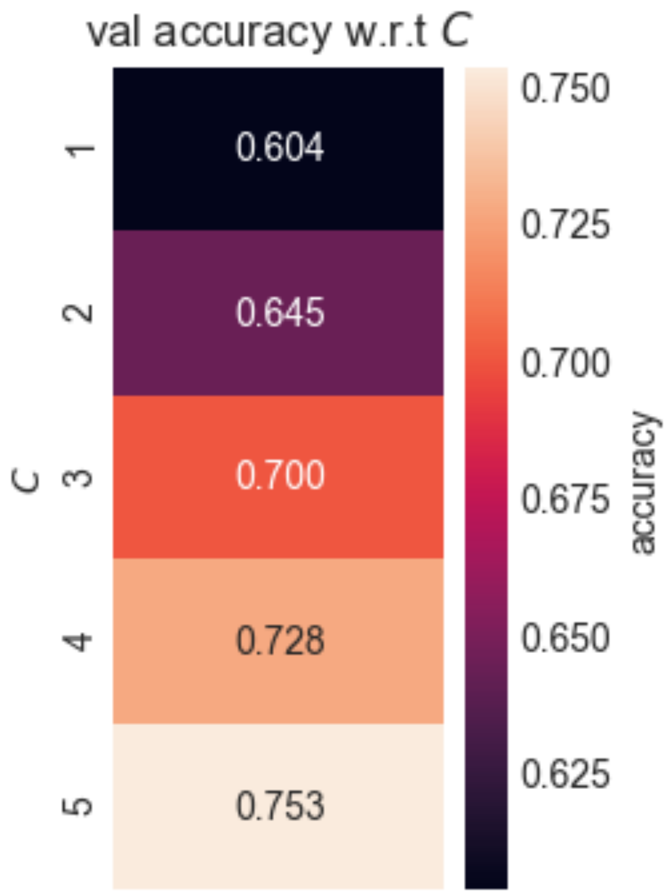
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [79]:

```
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)
```

```
[ 0.61059225  0.65761871  0.71673613  0.75345063  0.79038647]
[ 0.60380507  0.64452603  0.70026702  0.72797063  0.75333778]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.611
0.658
0.717
0.753
0.790
0.78
0.75
0.72
0.69
0.66
0.63
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.604
0.645
0.700
0.728
0.753
0.750
0.725
0.700
0.675
0.650
0.625
accuracy

In [80]:

```
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 0.753337783712 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.748998664887
```

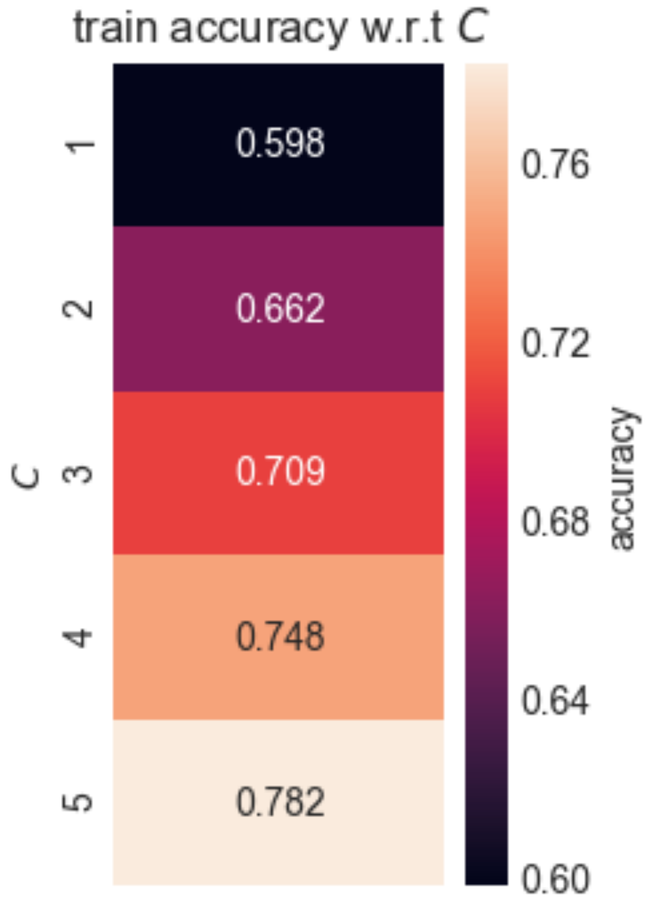
2nd Run)

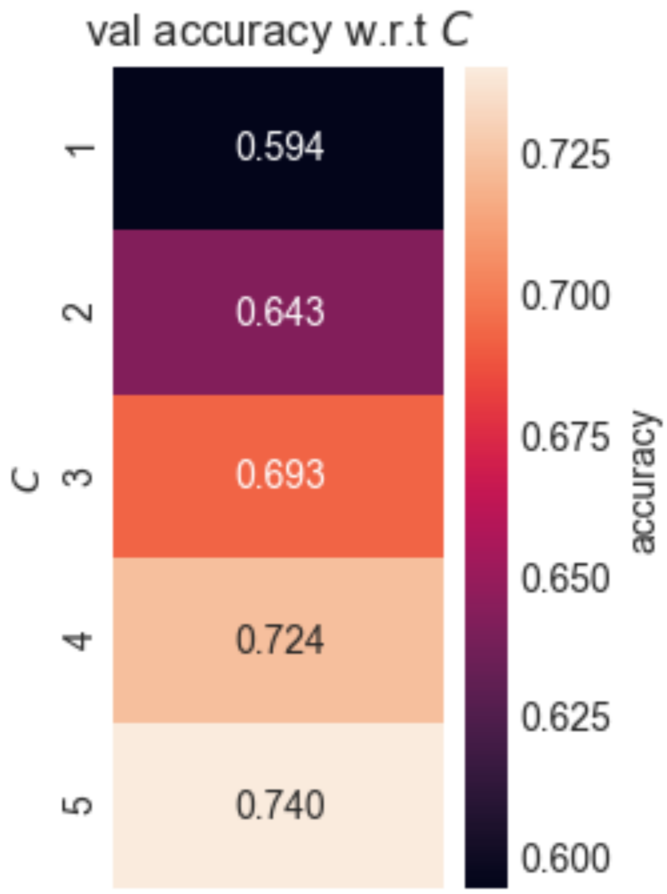
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [81]:

```
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)
```

```
[ 0.59846584  0.66229301  0.70905691  0.74799797  0.78211734]
[ 0.59445928  0.64252336  0.6929239   0.72363151  0.74032043]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.598
0.662
0.709
0.748
0.782
0.76
0.72
0.68
0.64
0.60
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.594
0.643
0.693
0.724
0.740
0.725
0.700
0.675
0.650
0.625
0.600
accuracy

In [82]:

```
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 0.740320427236 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.744993324433
```

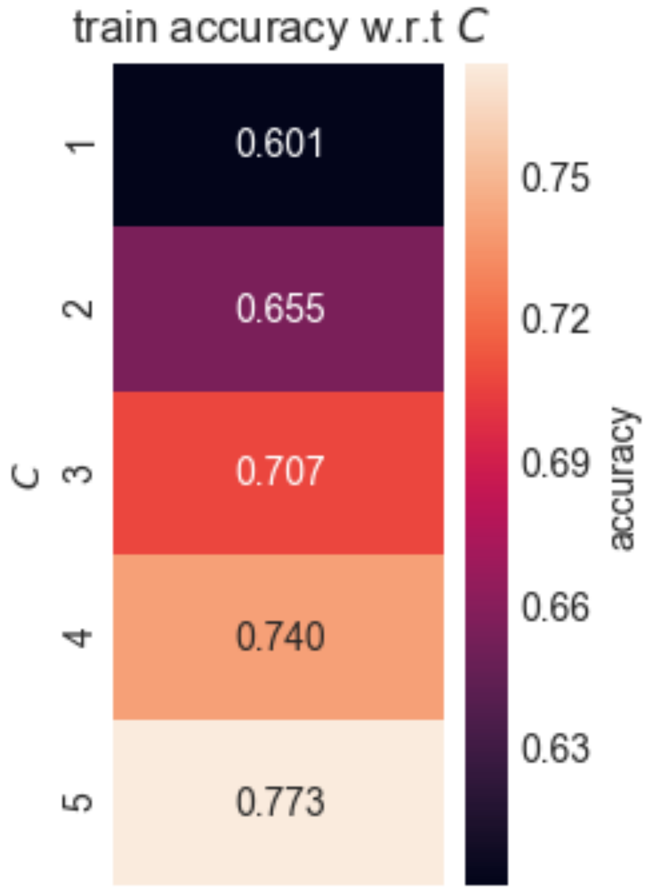
3rd Run)

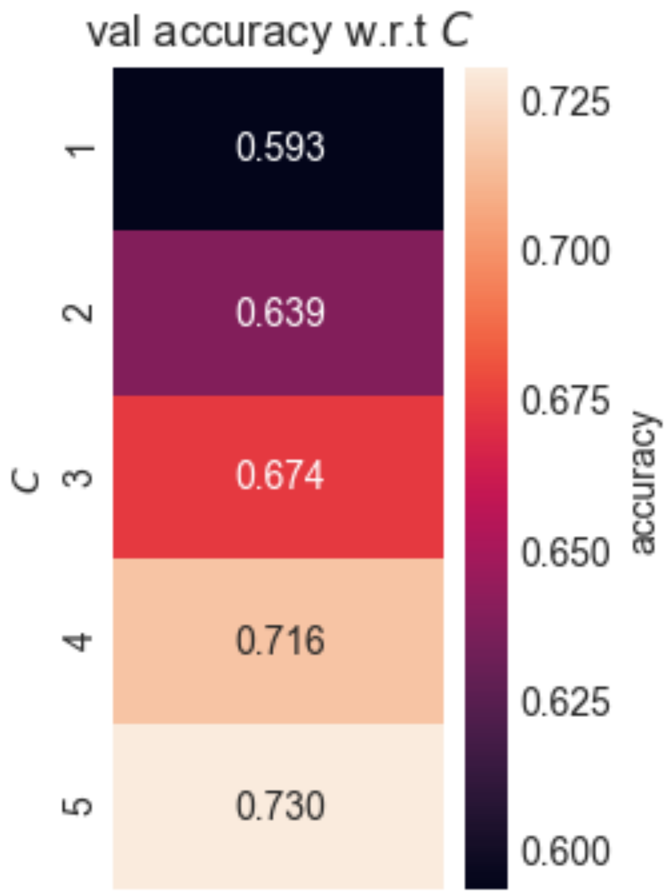
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [83]:

```
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)
```

```
[ 0.60128246  0.65502082  0.70731466  0.7404325   0.77340079]
[ 0.59345794  0.6388518   0.67389853  0.71628838  0.73030708]
```


train accuracy w.r.t C
0.601
0.655
0.707
0.740
0.773
1
2
3
4
5
C
0.75
0.72
0.69
0.66
0.63
accuracy


val accuracy w.r.t C
0.593
0.639
0.674
0.716
0.730
1
2
3
4
5
C
0.725
0.700
0.675
0.650
0.625
0.600
accuracy

In [84]:

```
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 0.730307076101 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.73564753004
```

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

In [85]:

```
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:[0.74899866488651534, 0.74499332443257682, 0.735
64753004005345]
RF_accuracyTestList mean: 0.74321317312
```

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 [86]:

```
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)
```

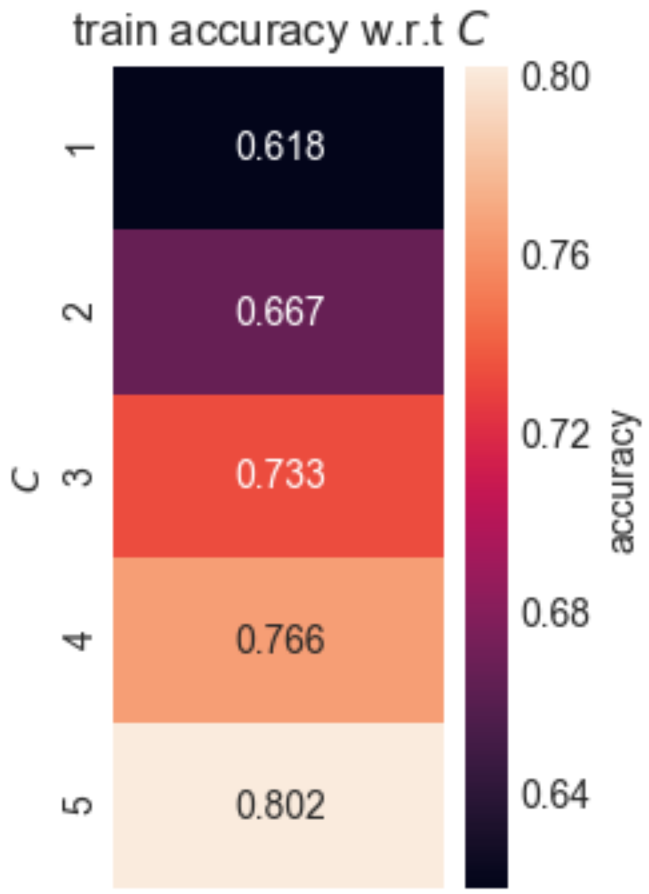
1st Run)

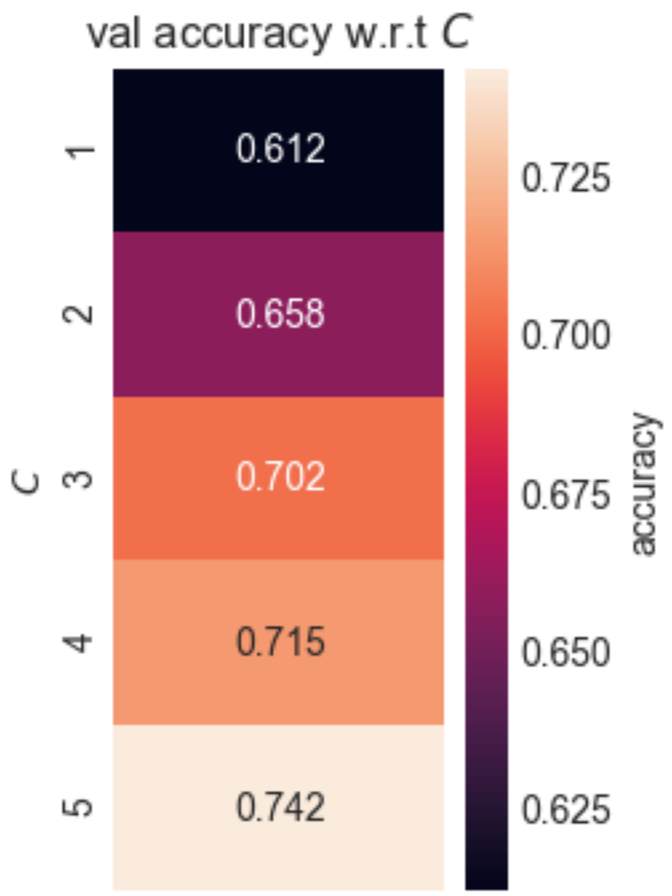
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [87]:

```
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)
```

```
[ 0.61841211  0.66696212  0.73344037  0.76578825  0.8019358 ]
[ 0.61217949  0.65811966  0.70245726  0.71474359  0.74198718]
```





In [88]:

```
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 0.741987179487 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.729845168179
```

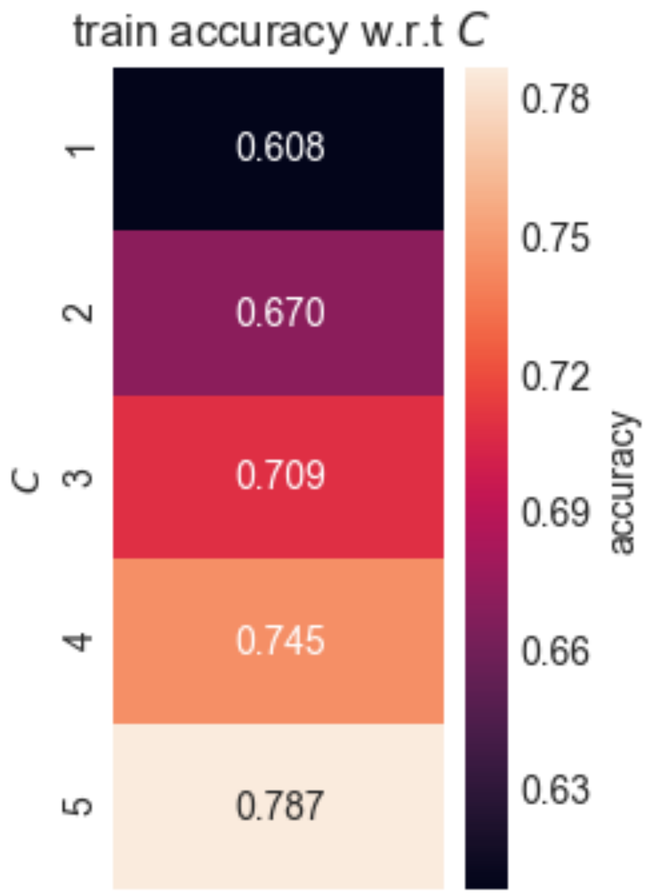
2nd Run)

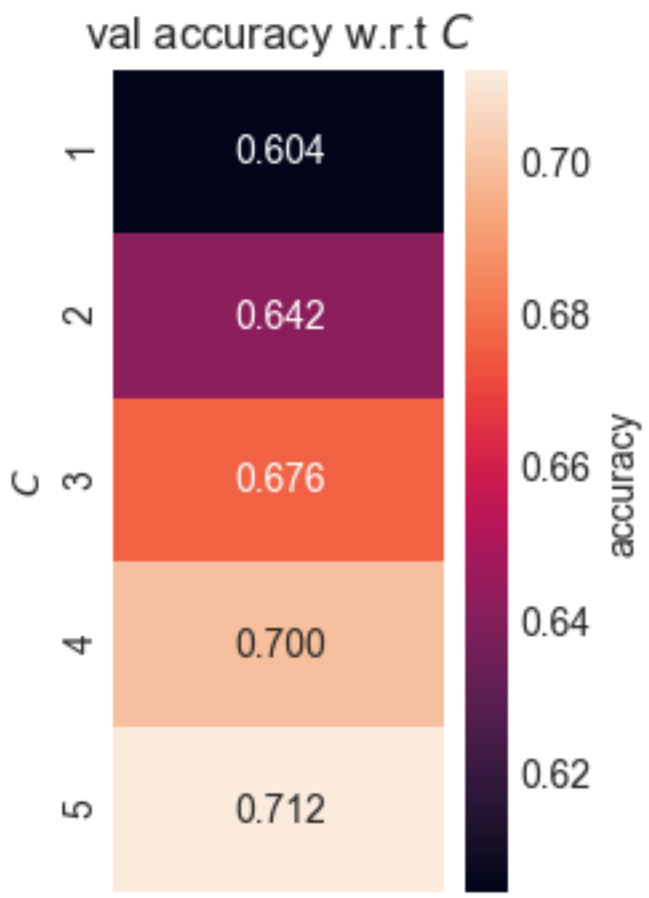
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [89]:

```
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)
```

```
[ 0.60784496  0.67022447  0.70904339  0.74518897  0.78655689]
[ 0.60416667  0.64209402  0.67628205  0.69978632  0.71207265]
```





In [90]:

```
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 0.712072649573 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.736785904965
```

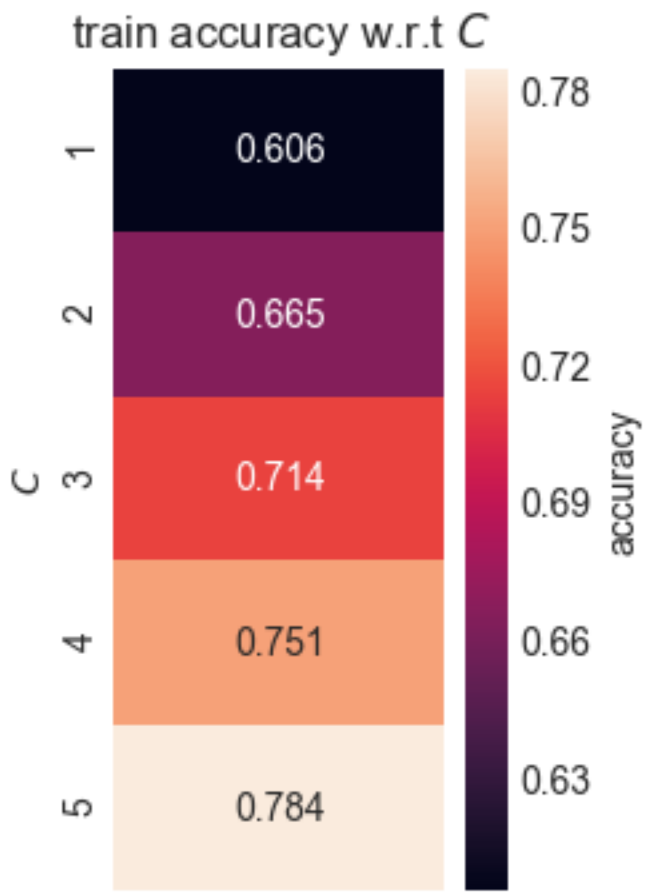
3rd Run)

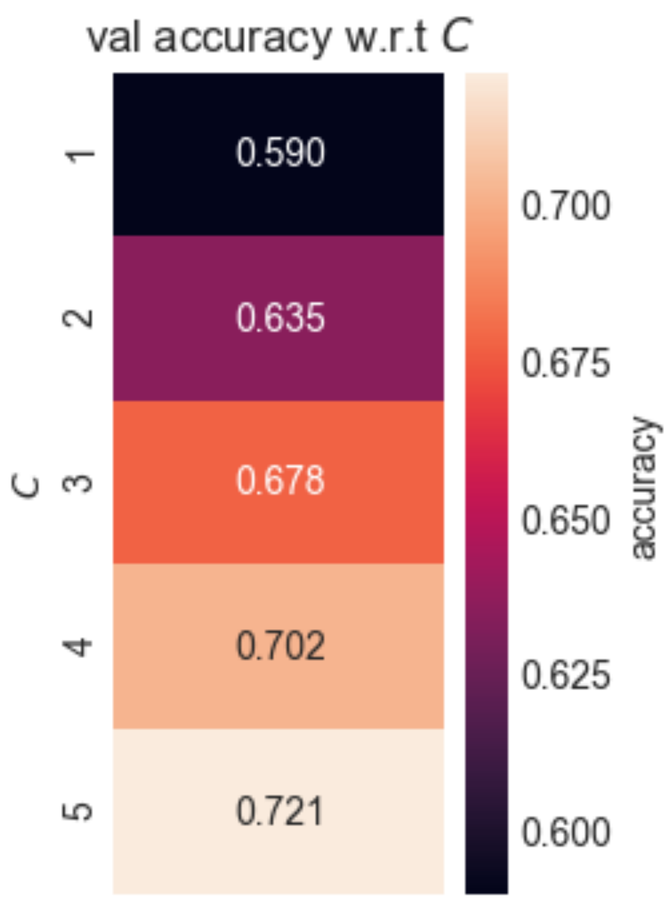
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [91]:

```
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)
```

```
[ 0.60553109  0.6651255   0.71403282  0.75088883  0.78448448]
[ 0.58974359  0.63514957  0.67788462  0.70245726  0.72115385]
```


train accuracy w.r.t C
0.606
0.665
0.714
0.751
0.784
1
2
3
4
5
C
0.78
0.75
0.72
0.69
0.66
0.63
accuracy


val accuracy w.r.t C
0.590
0.635
0.678
0.702
0.721
1
2
3
4
5
C
0.700
0.675
0.650
0.625
0.600
accuracy

In [92]:

```
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 0.721153846154 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.723438334223
```

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

In [93]:

```
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:[0.72984516817939138, 0.73678590496529628, 0.723
43833422317139]
RF_accuracyTestList mean: 0.730023135789
```

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 [94]:

```
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)
```

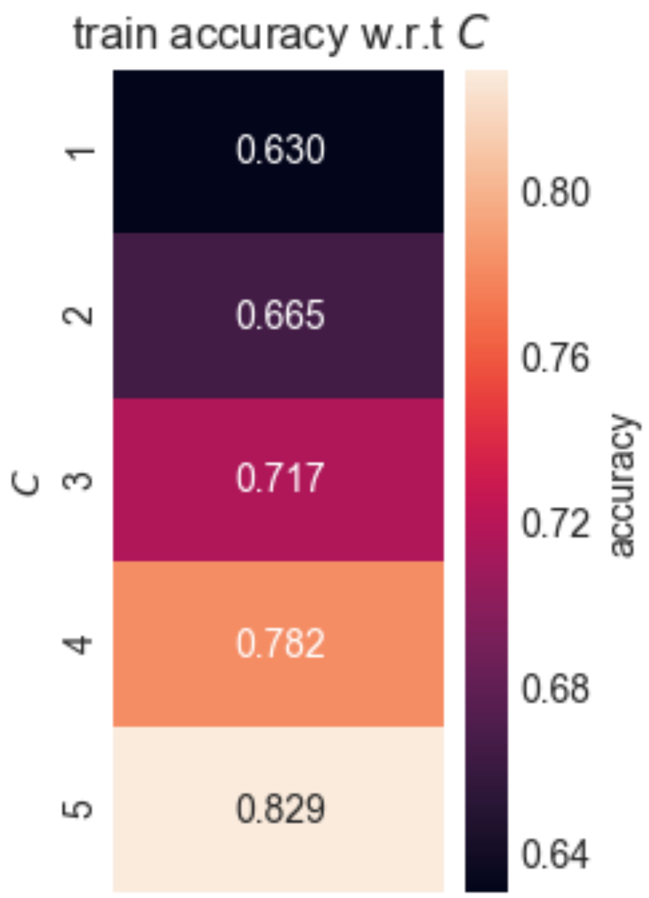
1st Run)

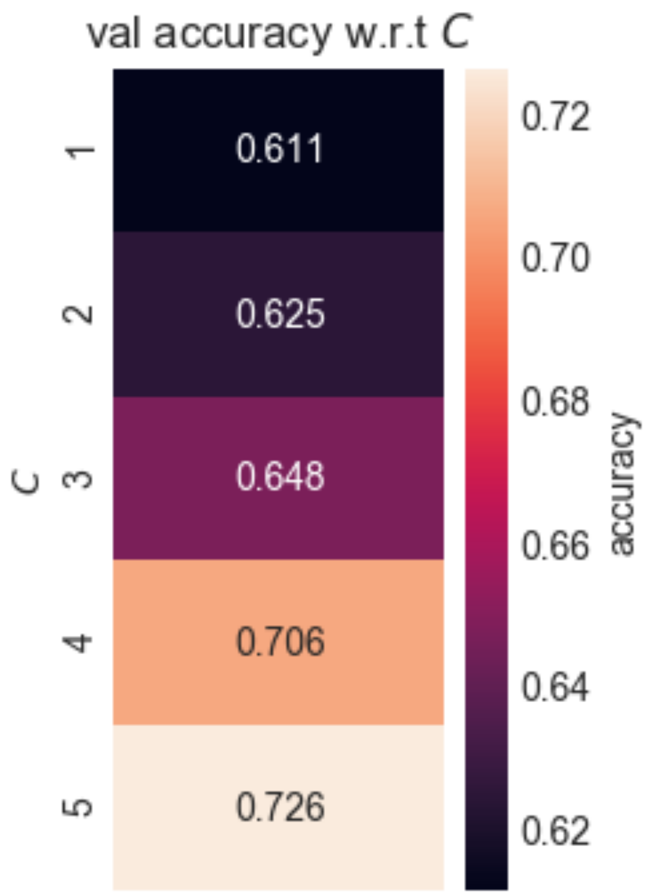
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

In [95]:

```
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)
```

```
[ 0.63017606  0.66548148  0.71680448  0.78222772  0.82895439]
[ 0.61148198  0.62483311  0.64753004  0.70627503  0.72630174]
```





In [96]:

```
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 0.726301735648 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.738651535381
```

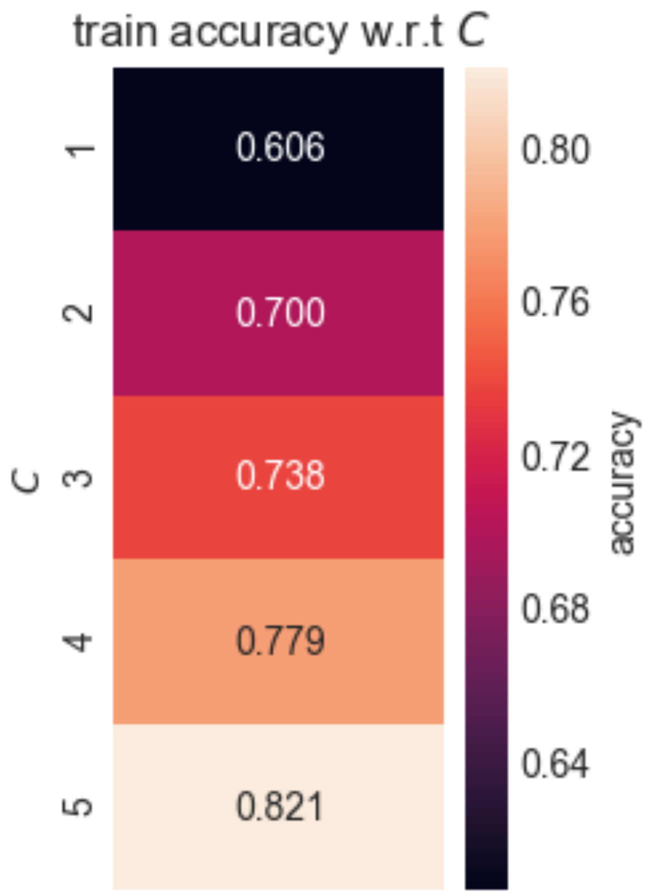
2nd Run)

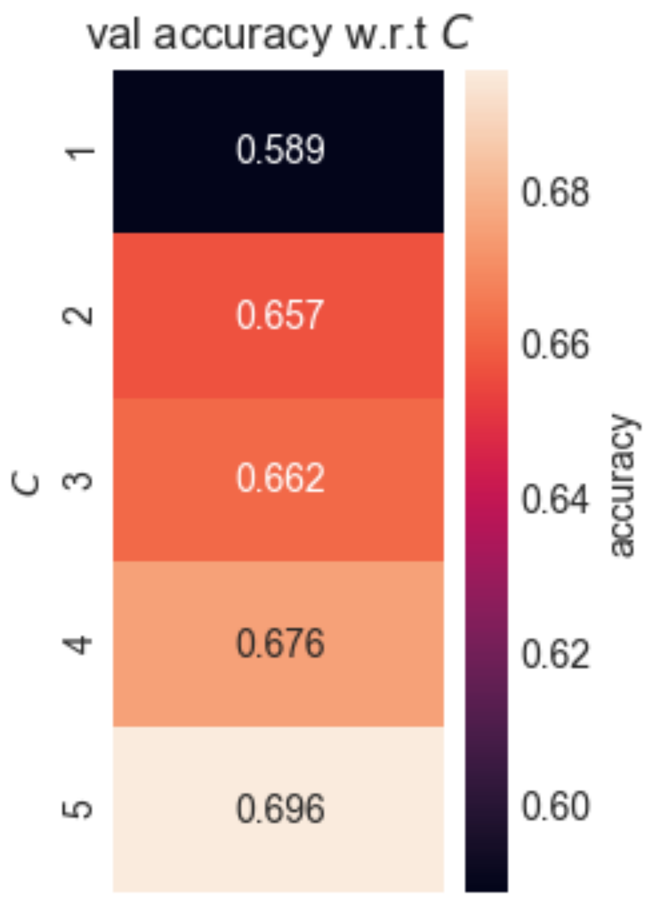
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

In [97]:

```
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)
```

```
[ 0.60644163  0.70019175  0.73801813  0.77881362  0.82109366]
[ 0.58878505  0.65687583  0.66221629  0.67556742  0.69559413]
```





In [98]:

```
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 0.695594125501 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.717957276368
```

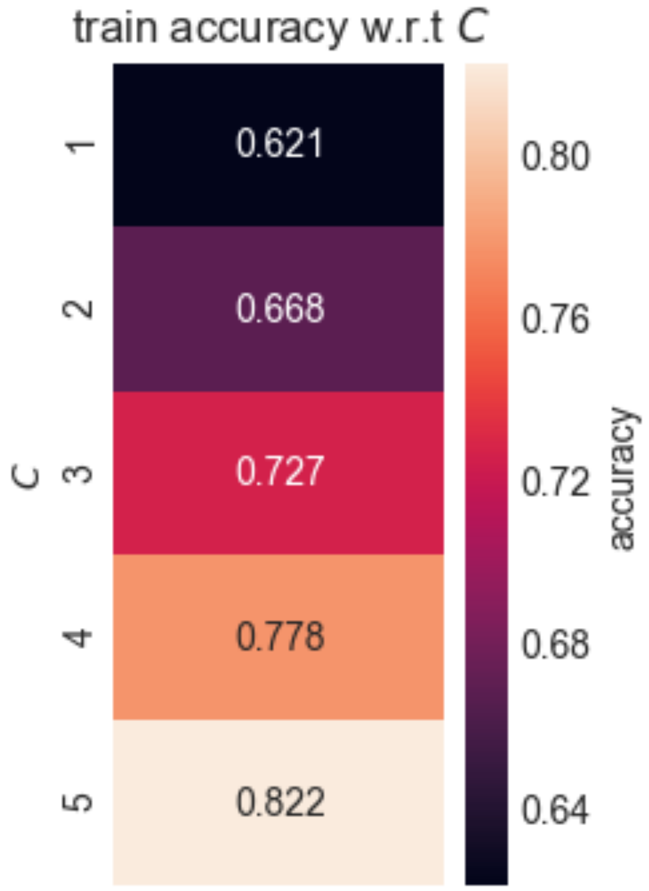
3rd Run)

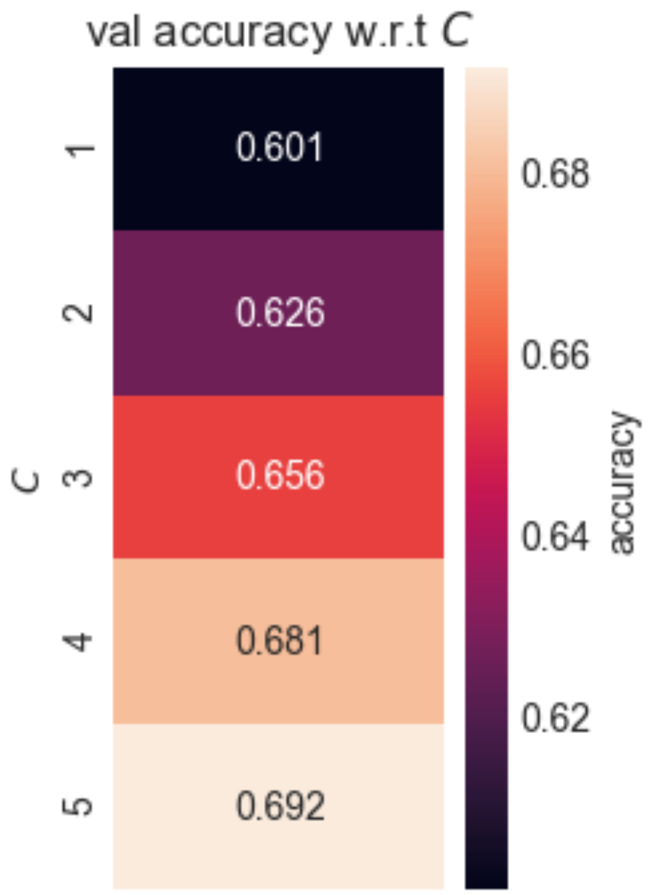
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [99]:

```
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)
```

```
[ 0.62083323  0.66829784  0.72659377  0.77762208  0.82243697]
[ 0.60080107  0.62616822  0.65554072  0.68090788  0.69158879]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.621
0.668
0.727
0.778
0.822
0.80
0.76
0.72
0.68
0.64
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.601
0.626
0.656
0.681
0.692
0.68
0.66
0.64
0.62
accuracy

In [100]:

```
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 0.691588785047 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.714285714286
```

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

In [101]:

```
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:[0.73865153538050732, 0.71795727636849127, 0.714
2857142857143]
RT_accuracyTestList mean: 0.723631508678
```

Results of Random Forest

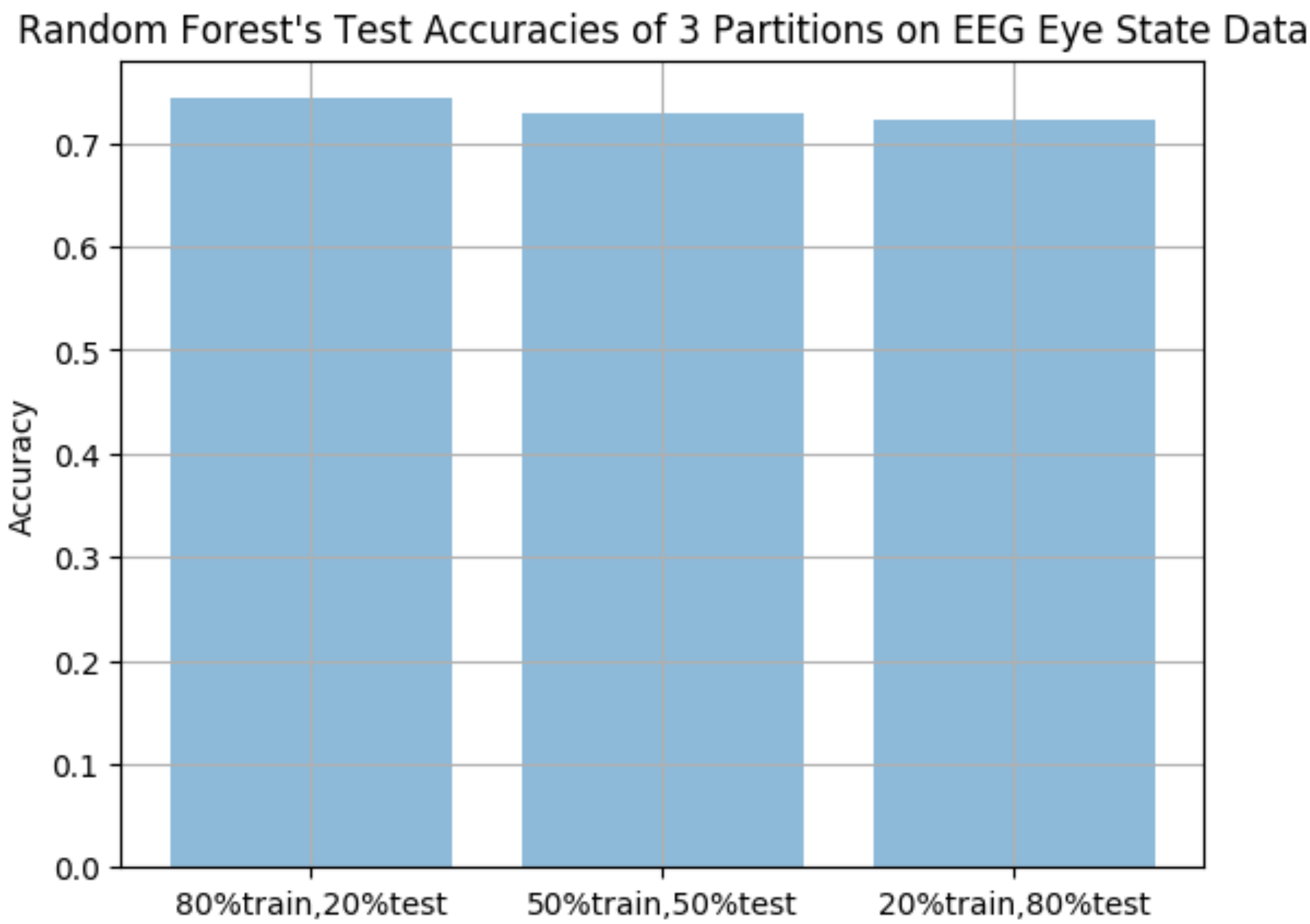
In [102]:

```
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: 0.74321317
312
RT_accuracyTestList (50% train, 50% test) partition mean: 0.73002313
5789
RT_accuracyTestList (20% train, 80% test) partition mean: 0.72363150
8678
```

In [110]:

```
displayAccuracies('Random Forest', 'EEG Eye State Data', RF_accuracyAverage_80_2
0, RF_accuracyAverage_50_50, RF_accuracyAverage_20_80)
printAccuracies('RF', RF_accuracyTestList_80_20, RF_accuracyTestList_50_50, RF_a
ccuracyTestList_20_80)
```




```
Accuracy of RF's 3 trials on (80% train, 20% test) partition :[0.748
99866488651534, 0.74499332443257682, 0.73564753004005345]
Mean Accuracy of RF on (80% train, 20% test) partition: 0.7432131731
2

Accuracy of RF's 3 trials on (50% train, 50% test) partition :[0.729
84516817939138, 0.73678590496529628, 0.72343833422317139]
Mean Accuracy of RF on (50% train, 50% test) partition: 0.7300231357
89

Accuracy of RF's 3 trials on (20% train, 80% test) partition:[0.7386
5153538050732, 0.71795727636849127, 0.7142857142857143]
Mean Accuracy of RF on (20% train, 80% test) partition: 0.7236315086
78
```

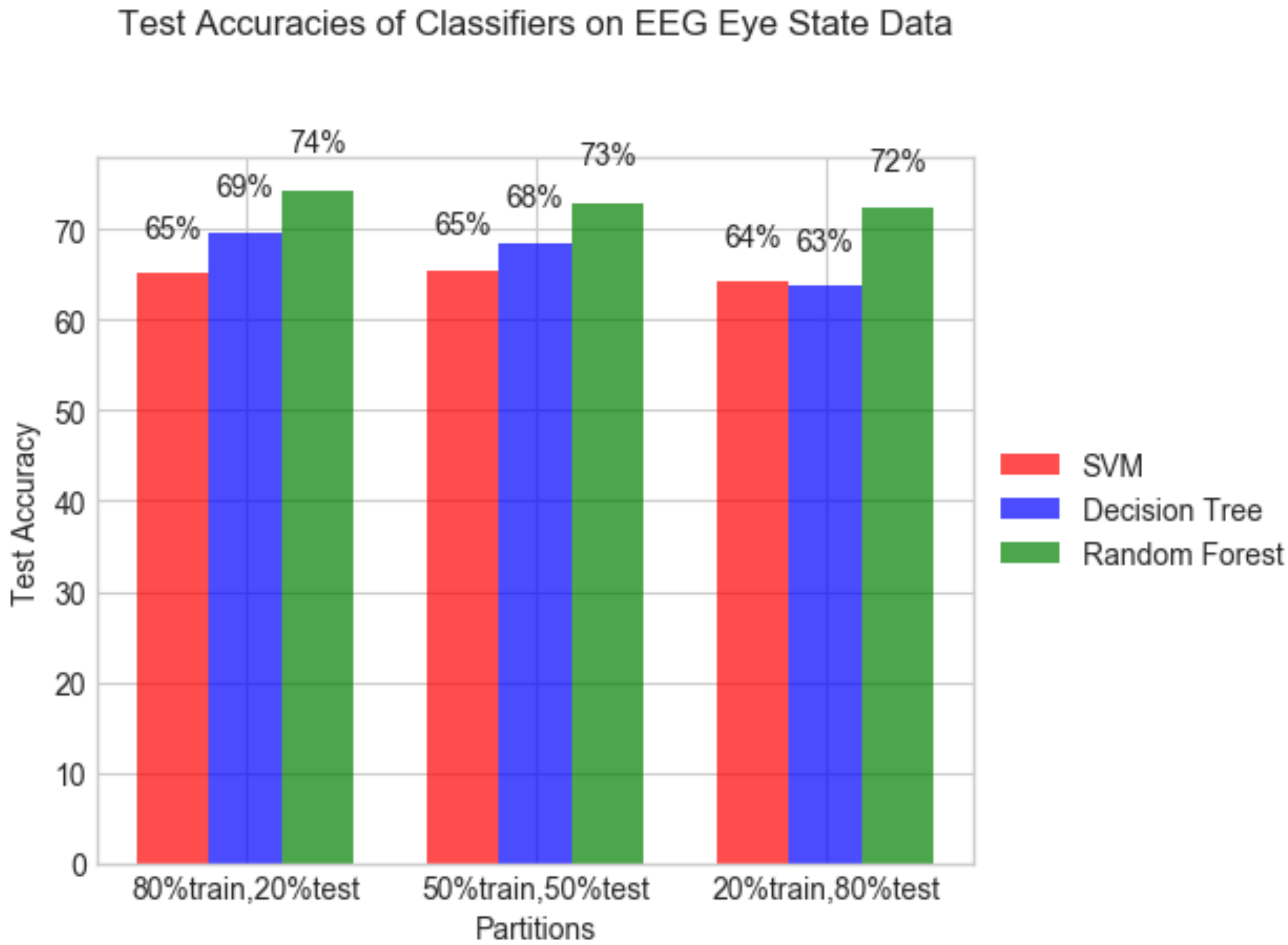
Results

In [104]:

```
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 [106]:

```
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 EEG Eye State 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 EEG Eye State Data
74%
69%
65%
73%
68%
65%
72%
64%
63%
70
60
50
40
30
20
10
0
Test Accuracy
80%train,20%test
50%train,50%test
20%train,80%test
Partitions
SVM
Decision Tree
Random Forest