Wine Data Set

This data is the result of a chemical analysis of wines grown in the same region in Italy but derived from 3 different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

Attribute Information

1. Type <-- (Label which has 3 classes)
2. Alcohol
3. Malic acid
4. Ash
5. Alcalinity of ash
6. Magnesium
7. Total phenols
8. Flavanoids
9. Nonflavanoid phenols
10. Proanthocyanins
11. Color intensity
12. Hue
13. OD280/OD315 of diluted wines
14. Proline

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/Wine/wine.dat
a.csv')
df.columns = ['Type', 'Alcohol','Malic Acid','Ash','Alcalinity of Ash','Magnesiu
m','Total phenols','Flavanoids','Nonflavanoid phenols','Proanthocyanins','Color
intensity', 'Hue', 'OD280/OD315', 'Proline']
```

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: (177, 14)
```

Out[3]:

	Type	Alcohol	Malic Acid	Ash	Alcalinity of Ash	Magnesium	Total phenols	Flavanoids	Nonflavanoic phenols
0	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26
1	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30
2	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24
3	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39
4	1	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34

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]:

```
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: 58
Class 2 count: 71
Class 3 count: 48
```

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 order to
slice into X and Y.

X1 = df_array1[:,1:df_array1.shape[1]]  #Second Column to last column. Features
are all numerical.
Y1 = df_array1[:, 0]                    #First Column. This feature represents t
he classes or type of wine.
```

In [9]:

```
df_array2 = np.array(df_shuffle2)      #Convert dataframe to array in order to s
lice into X and Y.

X2 = df_array2[:,1:df_array2.shape[1]]  #Second Column to last column. Features
are all numerical.
Y2 = df_array2[:, 0]                    #First Column. This feature represents t
he classes or type of wine.
```

In [10]:

```
df_array3 = np.array(df_shuffle3)      #Convert dataframe to array in order to s
lice into X and Y.

X3 = df_array3[:,1:df_array3.shape[1]]  #Second Column to last column. Features
are all numerical.
Y3 = df_array3[:, 0]                    #First Column. This feature represents t
he classes or type of wine.
#print('X3 shape: ' + str(X3.shape))
#print('Y3 shape: ' + str(Y3.shape))
#print(X3[:, 0])
#print(Y3)
```

Functions Used For All Classifiers

1. partitionData
2. viewSplit
3. draw_heatmap_linear
4. bestValue **(EXTRA CREDIT)**
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 [106]:

```
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('Random Forest\'s Test Accuracies for 3 Partitions')
    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)
```

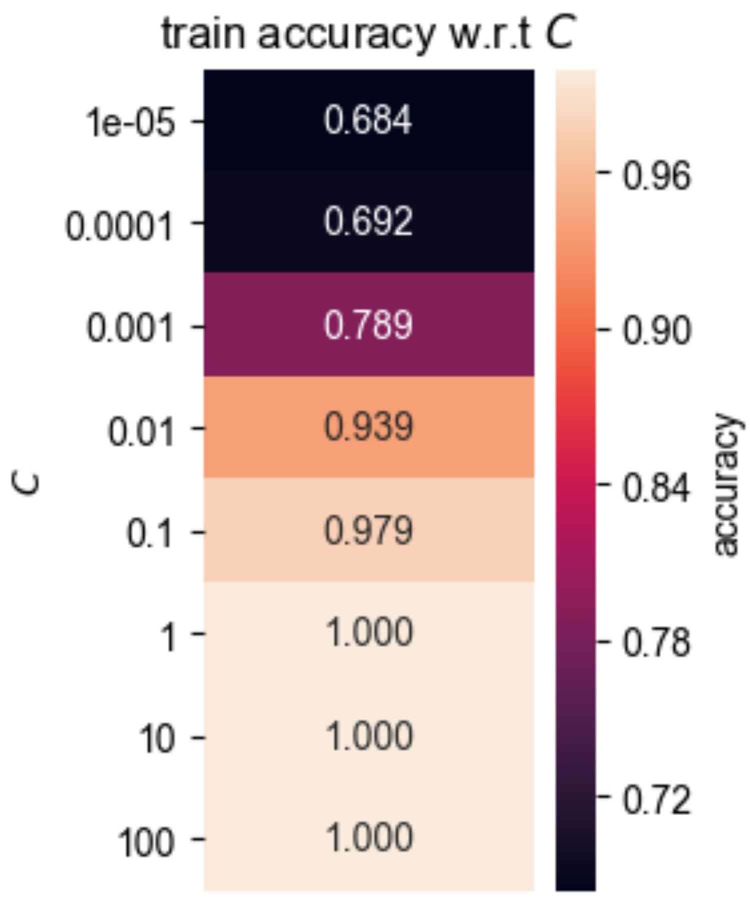
1st Run)

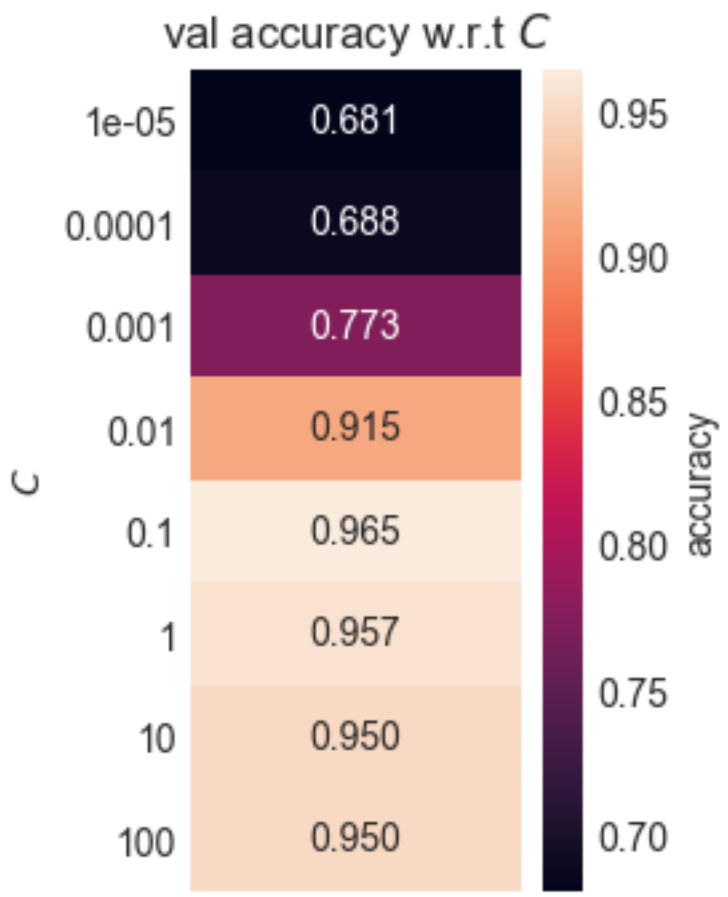
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

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)
```

```
[ 0.68398668  0.69186128  0.78875117  0.93933333  0.97872579  1.
1.
  1.        ]
[ 0.68085106  0.68794326  0.77304965  0.91489362  0.96453901  0.9574
4681
  0.95035461  0.95035461]
```





In [23]:

```
#EXTRA CREDIT
#Instead of creating my own implementation to pick out the best hyper.
#I could have used this to find the best hyperparameter.
print(SVM_clfGridSearch.best_params_)

#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))
```

```
{'C': 0.1}
Largest value in accuracyValidation is 0.964539007092 from index 4.
Best C: 0.1
Test Accuracy Score: 0.972222222222
```

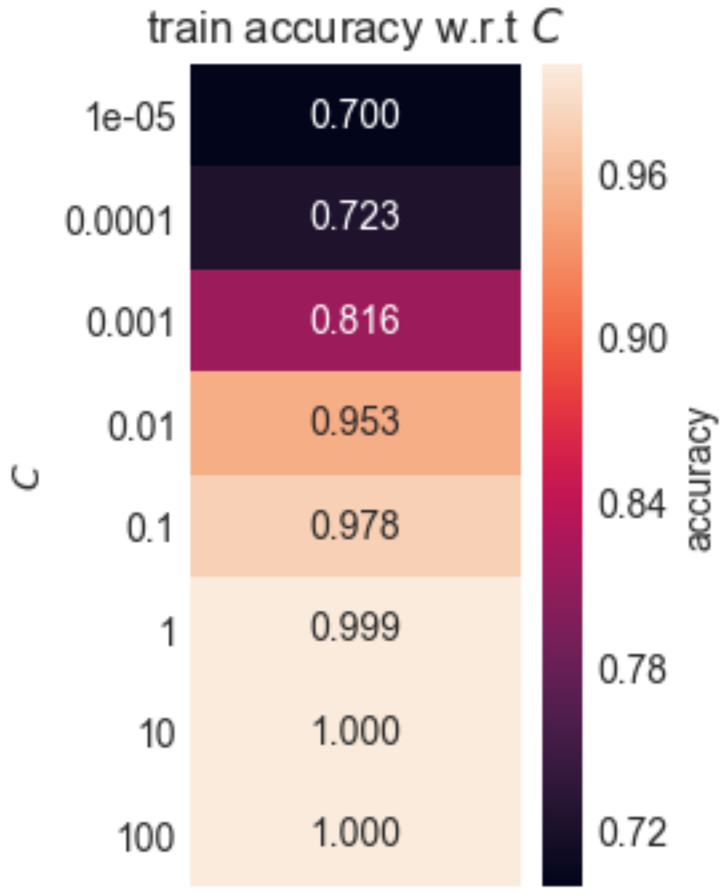
2nd Run)

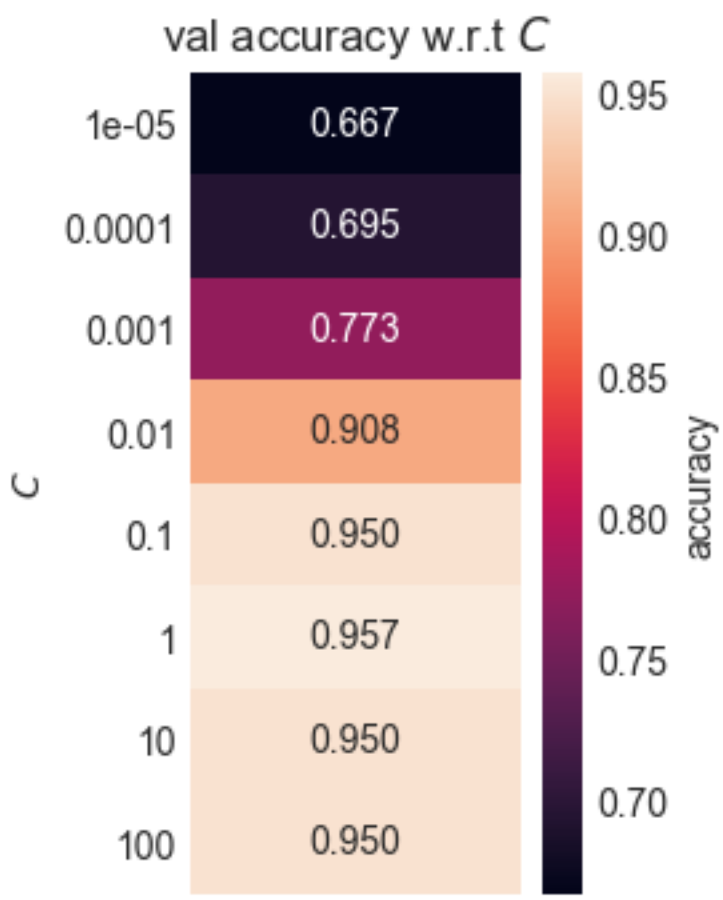
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

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)
```

```
[ 0.69974585  0.72334506  0.81635927  0.95272756  0.97791993  0.9992
1.
  1.        ]
[ 0.66666667  0.69503546  0.77304965  0.90780142  0.95035461  0.9574
4681
  0.95035461  0.95035461]
```





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 0.957446808511 from index 5.
Best C: 1
Test Accuracy Score: 0.972222222222
```

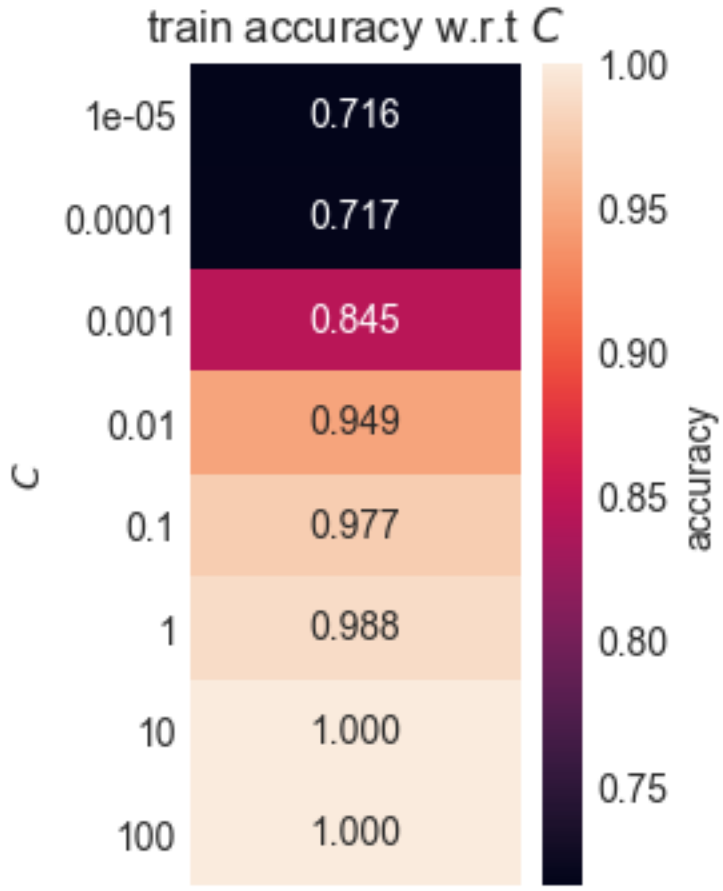
3rd Run)

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

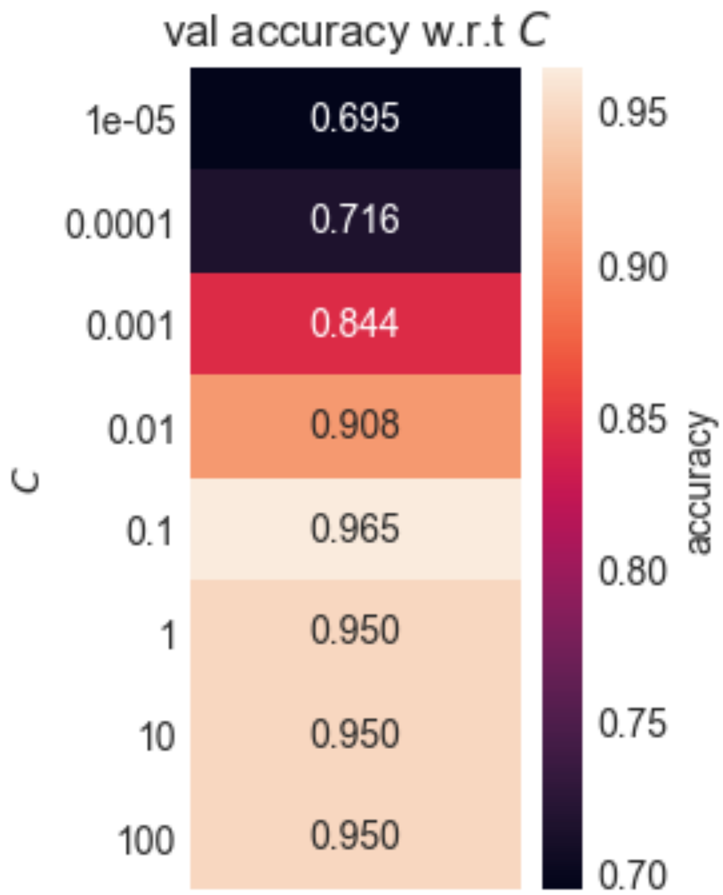
In [26]:

```
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)
```

```
[ 0.71556132  0.71714257  0.8447841   0.94871629  0.9771513   0.9881
9425
  1.          1.        ]
[ 0.69503546  0.71631206  0.84397163  0.90780142  0.96453901  0.9503
5461
  0.95035461  0.95035461]
```



train accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.716
0.717
0.845
0.949
0.977
0.988
1.000
1.000
1.00
0.95
0.90
0.85
0.80
0.75
accuracy



val accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.695
0.716
0.844
0.908
0.965
0.950
0.950
0.950
0.95
0.90
0.85
0.80
0.75
0.70
accuracy

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 0.964539007092 from index 4.
Best C: 0.1
Test Accuracy Score: 0.972222222222
```

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:[0.9722222222222221, 0.9722222222222221, 0.97
22222222222221]
SVM_accuracyTestList mean: 0.972222222222
```

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)
```

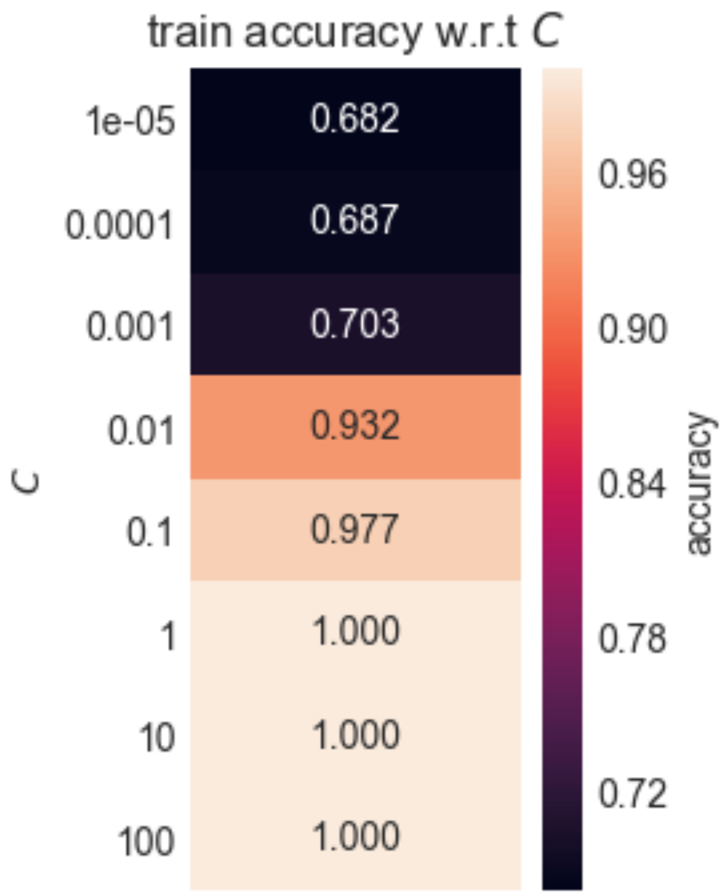
1st Run)

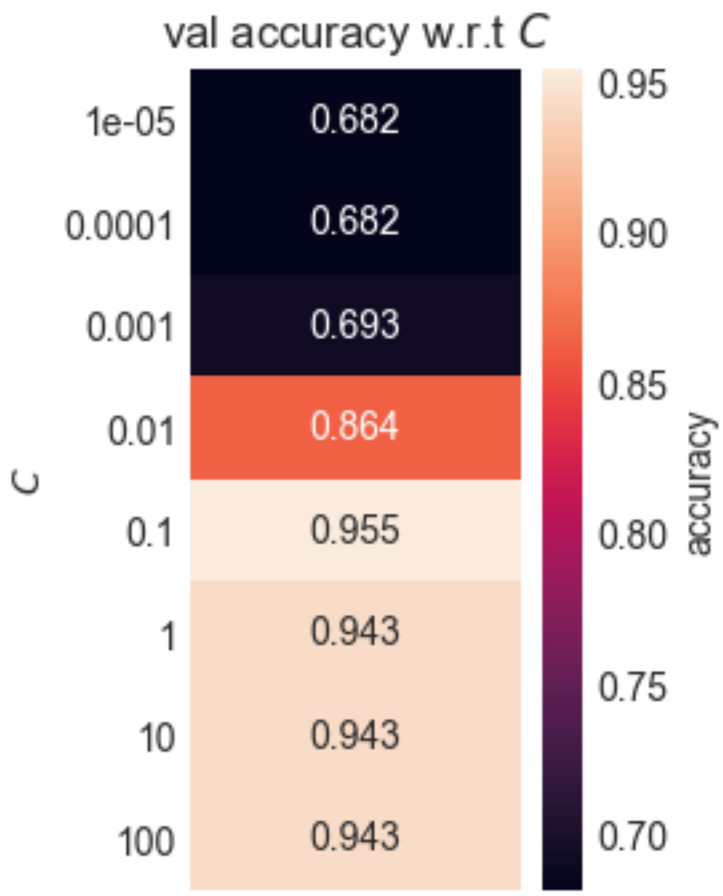
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

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)
```

```
[ 0.6817953   0.68681113  0.7031577   0.93182725  0.97727561  1.
1.
  1.        ]
[ 0.68181818  0.68181818  0.69318182  0.86363636  0.95454545  0.9431
8182
  0.94318182  0.94318182]
```





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 0.954545454545 from index 4.
Best C: 0.1
Test Accuracy Score: 0.966292134831
```

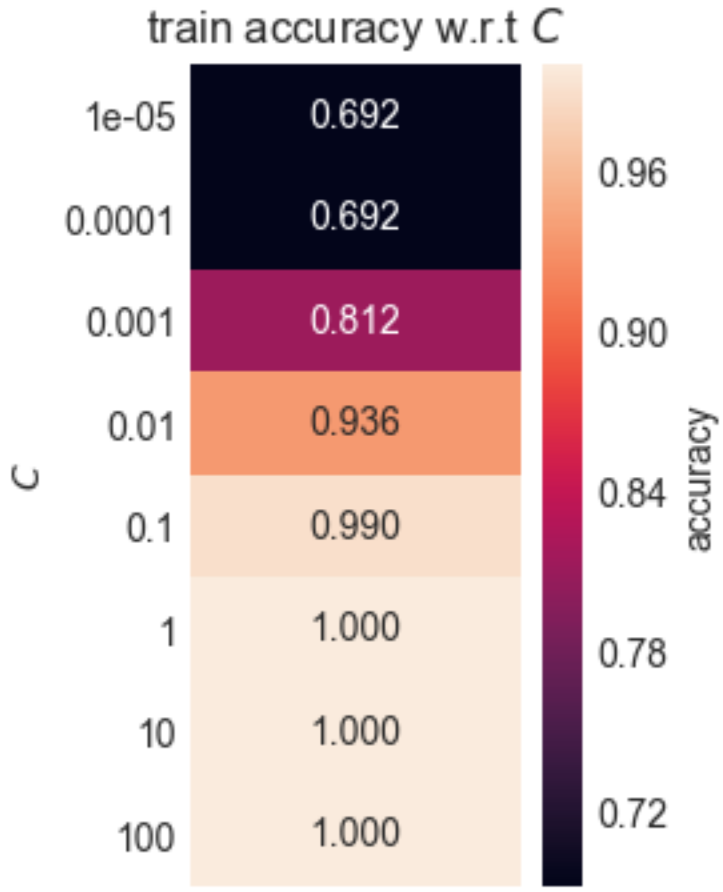
2nd Run)

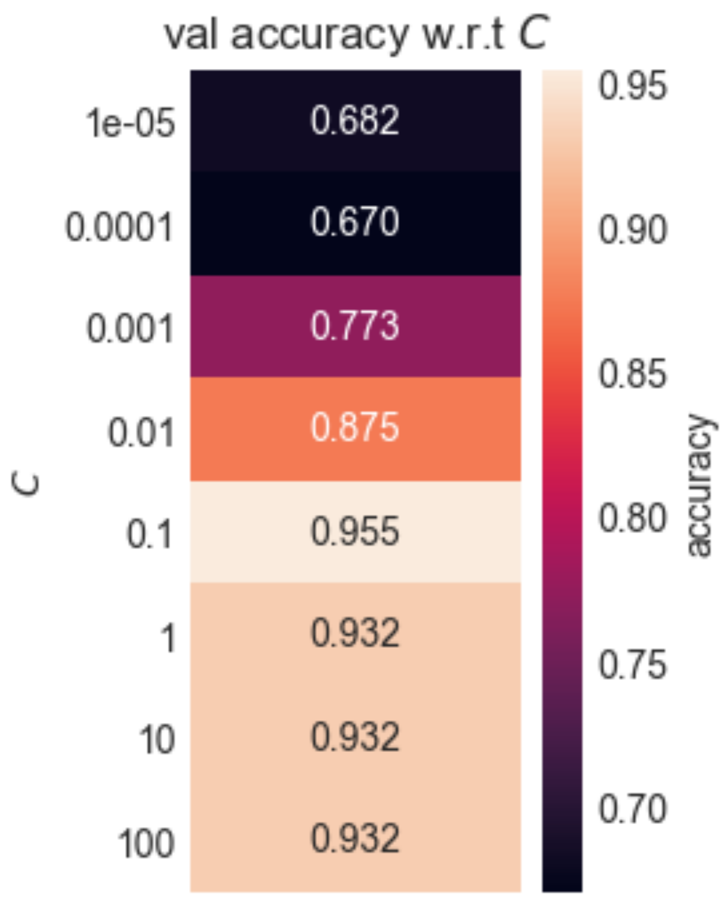
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

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)
```

```
[ 0.69187678  0.69194088  0.81169753  0.93565052  0.98990266  1.
1.
  1.        ]
[ 0.68181818  0.67045455  0.77272727  0.875       0.95454545  0.9318
1818
  0.93181818  0.93181818]
```





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 0.954545454545 from index 4.
Best C: 0.1
Test Accuracy Score: 0.955056179775
```

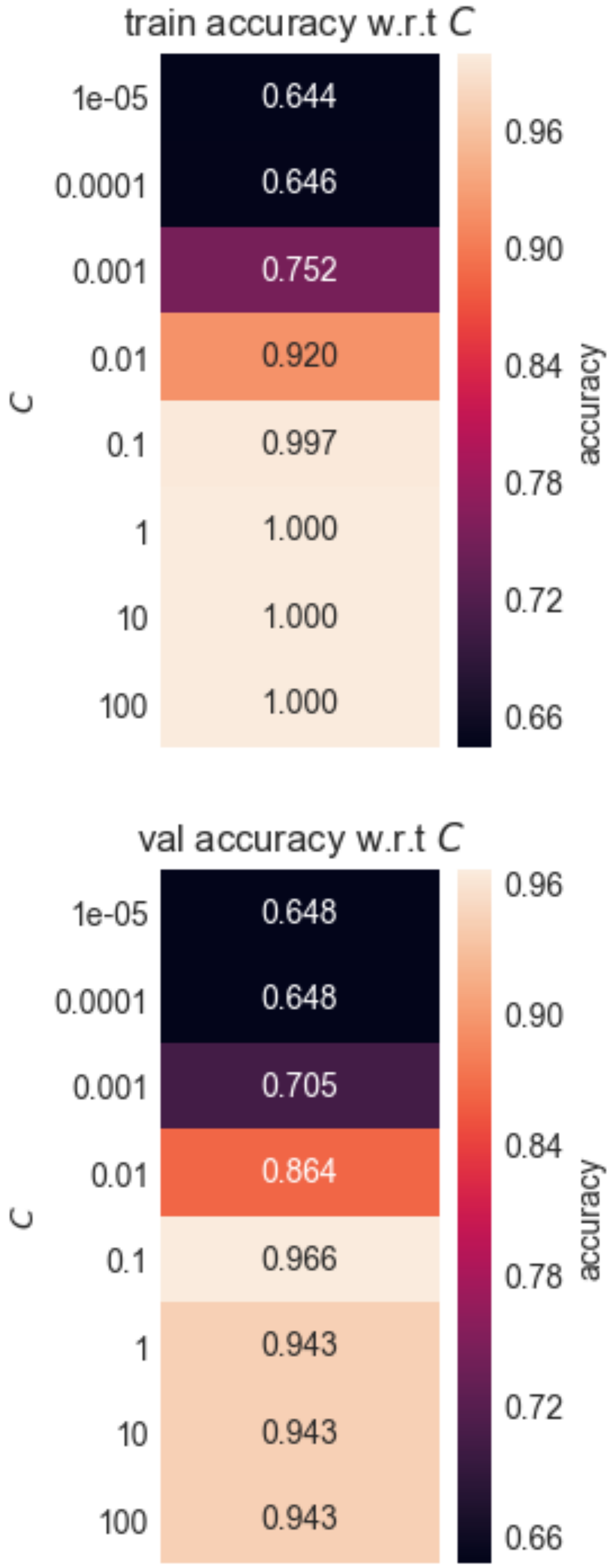
3rd Run)

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)
```

```
[ 0.64404256  0.6464476   0.75249178  0.92035102  0.99746795  1.
1.
  1.        ]
[ 0.64772727  0.64772727  0.70454545  0.86363636  0.96590909  0.9431
8182
  0.94318182  0.94318182]
```


train accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.644
0.646
0.752
0.920
0.997
1.000
1.000
1.000
0.96
0.90
0.84
0.78
0.72
0.66
accuracy
val accuracy w.r.t C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
C
0.648
0.648
0.705
0.864
0.966
0.943
0.943
0.943
0.96
0.90
0.84
0.78
0.72
0.66
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 0.965909090909 from index 4.
Best C: 0.1
Test Accuracy Score: 0.966292134831
```

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:[0.9662921348314607, 0.9550561797752809, 0.9662
921348314607]
SVM_accuracyTestList mean: 0.962546816479
```

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)
```

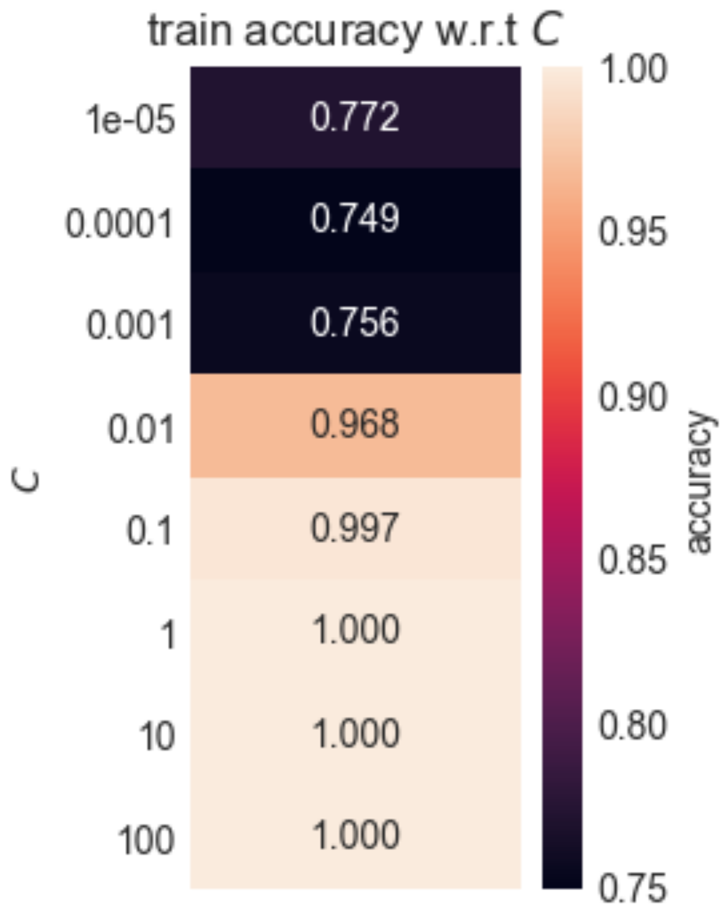
1st Run)

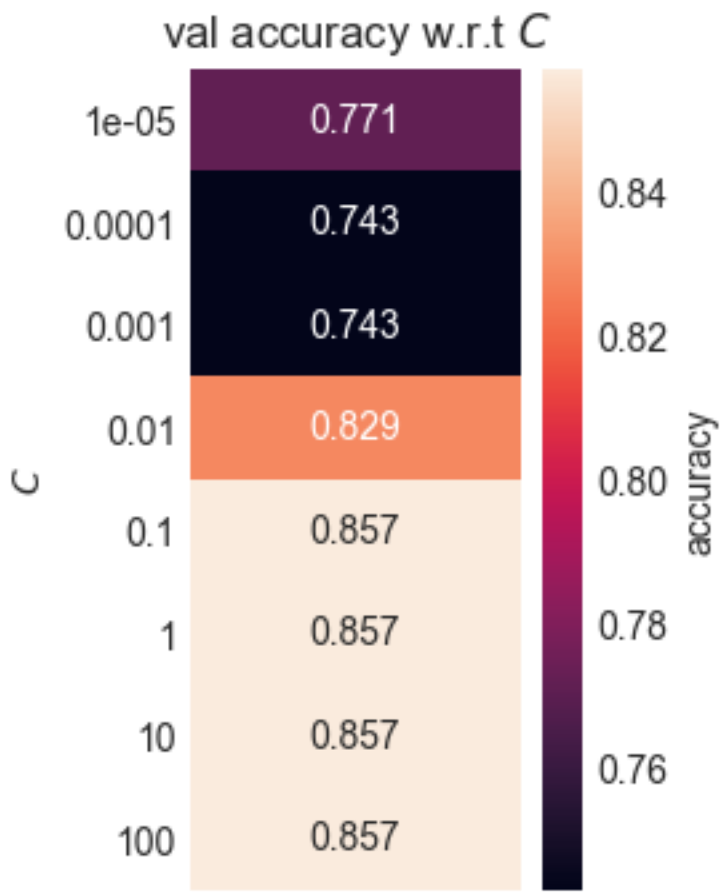
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

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)
```

```
/Users/rod/anaconda3/lib/python3.6/site-packages/sklearn/model_selec
tion/_split.py:605: Warning: The least populated class in y has only
7 members, which is too few. The minimum number of members in any cl
ass cannot be less than n_splits=10.
  % (min_groups, self.n_splits)), Warning)
```





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))
```

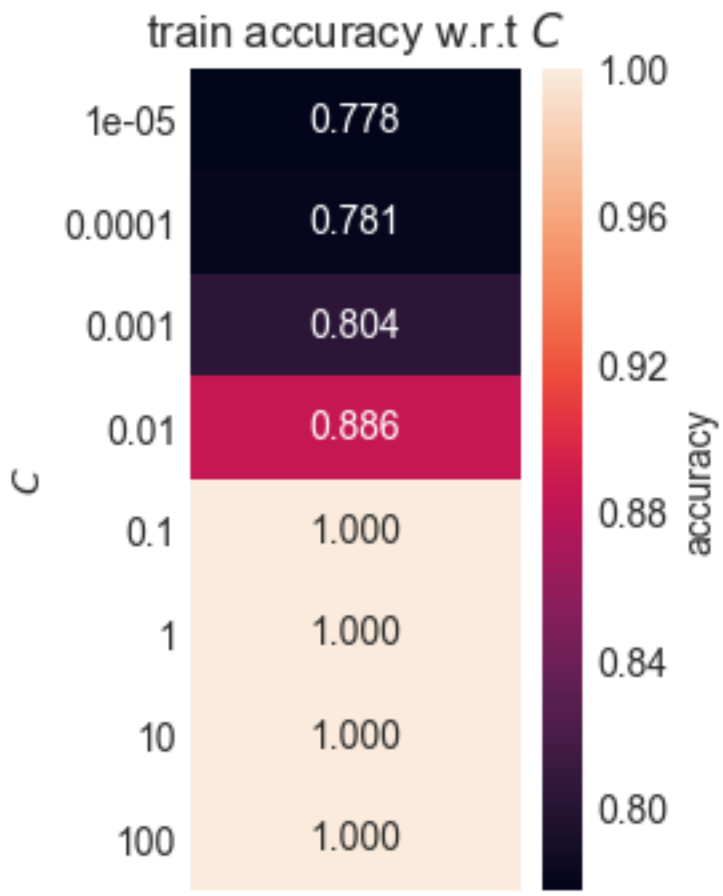
```
Largest value in accuracyValidation is 0.857142857143 from index 4.
Best C: 0.1
Test Accuracy Score: 0.859154929577
```

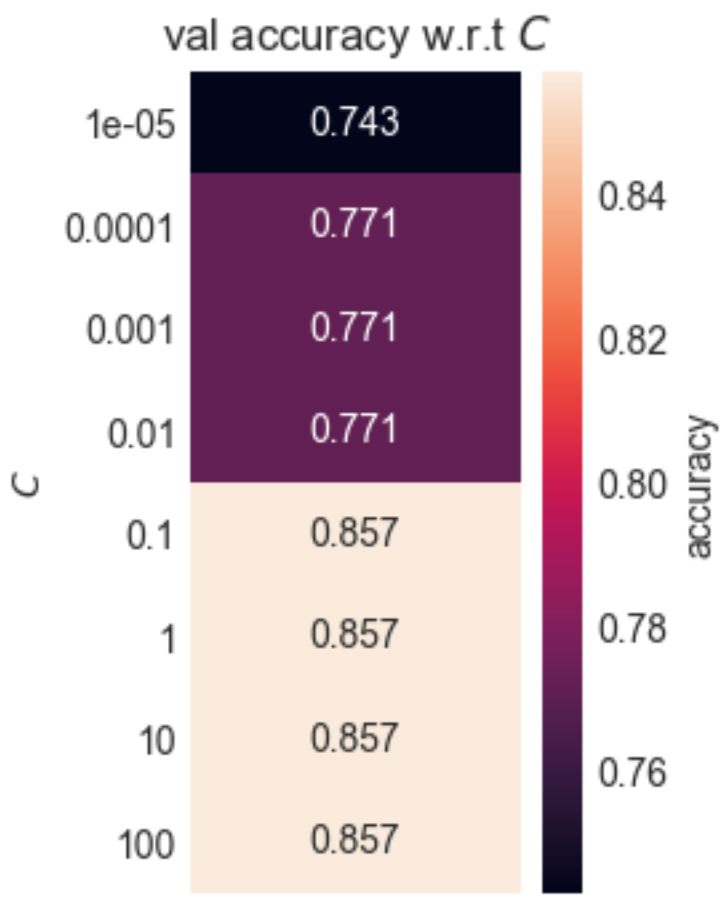
2nd Run)

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)
```





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 0.857142857143 from index 4.
Best C: 0.1
Test Accuracy Score: 0.880281690141
```

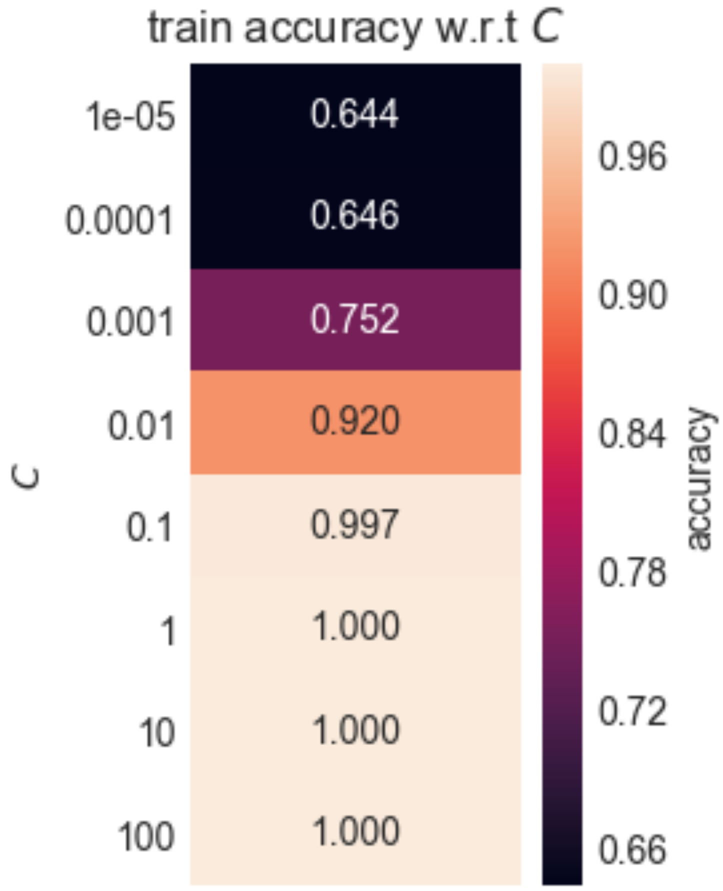
3rd Run)

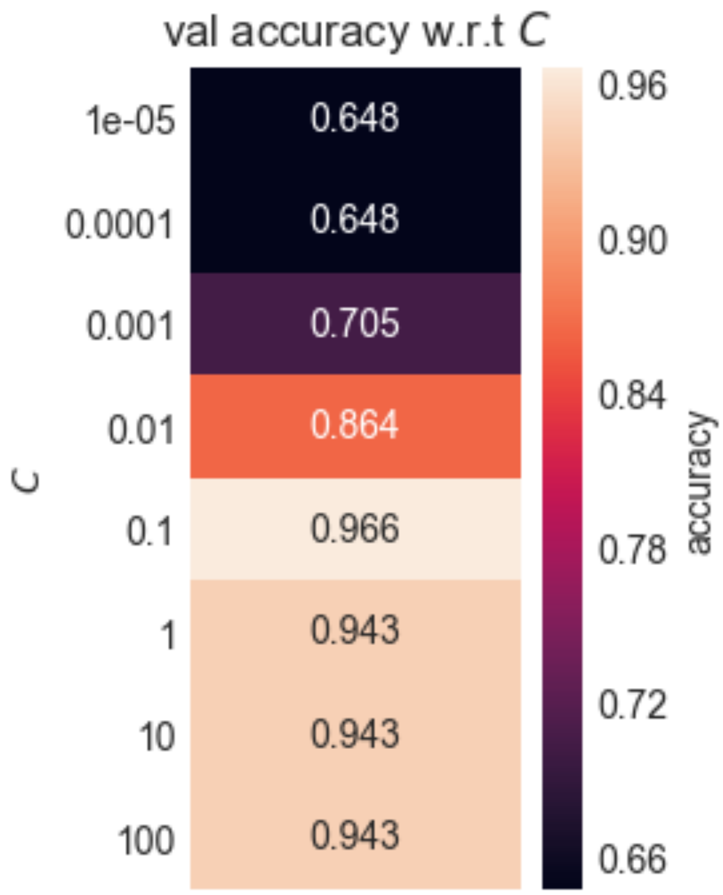
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

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)
```

```
[ 0.64404256  0.6464476   0.75249178  0.92035102  0.99746795  1.
1.
  1.        ]
[ 0.64772727  0.64772727  0.70454545  0.86363636  0.96590909  0.9431
8182
  0.94318182  0.94318182]
```


train accuracy w.r.t C
C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
0.644
0.646
0.752
0.920
0.997
1.000
1.000
1.000
accuracy
0.96
0.90
0.84
0.78
0.72
0.66


val accuracy w.r.t C
C
1e-05
0.0001
0.001
0.01
0.1
1
10
100
0.648
0.648
0.705
0.864
0.966
0.943
0.943
0.943
accuracy
0.96
0.90
0.84
0.78
0.72
0.66

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 0.965909090909 from index 4.
Best C: 0.1
Test Accuracy Score: 0.966292134831
```

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:[0.85915492957746475, 0.88028169014084512, 0.96
62921348314607]
SVM_accuracyTestList mean: 0.90190958485
```

Results of SVM

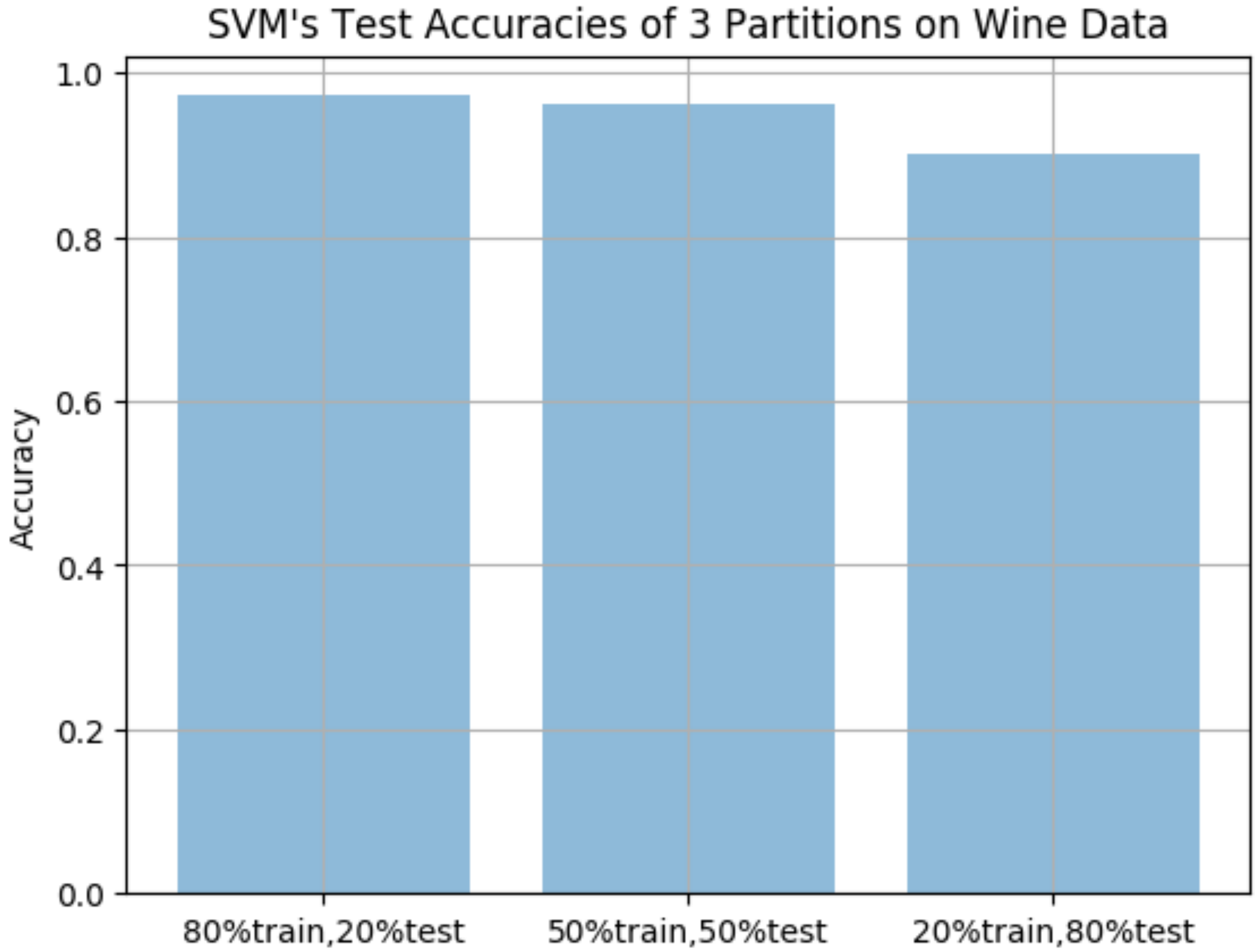
In [45]:

```
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: 0.9722222
22222
SVM_accuracyTestList (50% train, 50% test) partition mean: 0.9625468
16479
SVM_accuracyTestList (20% train, 80% test) partition mean: 0.9019095
8485
```

In [107]:

```
displayAccuracies('SVM', 'Wine Data',  SVM_accuracyAverage_80_20, SVM_accuracyAv
erage_50_50, SVM_accuracyAverage_20_80)

printAccuracies('SVM', SVM_accuracyTestList_80_20, SVM_accuracyTestList_50_50, S
VM_accuracyTestList_20_80)
# print('Accuracy of SVM\'s 3 trials on (80% train, 20% test) partition :' + str
(SVM_accuracyTestList_80_20))
# SVM_accuracyAverage_80_20 = statistics.mean(SVM_accuracyTestList_80_20)
# print('Mean Accuracy of SVM on (80% train, 20% test) partition: ' + str(SVM_ac
curacyAverage_80_20))

# print('\nAccuracy of SVM\'s 3 trials on (50% train, 50% test) partition :' + s
tr(SVM_accuracyTestList_50_50))
# SVM_accuracyAverage_50_50 = statistics.mean(SVM_accuracyTestList_50_50)
# print('Mean Accuracy of SVM on (50% train, 50% test) partition: ' + str(SVM_ac
curacyAverage_50_50))

# print('\nAccuracy of SVM\'s 3 trials on (20% train, 80% test) partition:' + st
r(SVM_accuracyTestList_20_80))
# SVM_accuracyAverage_20_80 = statistics.mean(SVM_accuracyTestList_20_80)
# print('Mean Accuracy of SVM on (20% train, 80% test) partition: ' + str(SVM_ac
curacyAverage_20_80))
```




```
Accuracy of SVM's 3 trials on (80% train, 20% test) partition :[0.97
22222222222221, 0.9722222222222221, 0.9722222222222221]
Mean Accuracy of SVM on (80% train, 20% test) partition: 0.972222222
222

Accuracy of SVM's 3 trials on (50% train, 50% test) partition :[0.96
62921348314607, 0.9550561797752809, 0.9662921348314607]
Mean Accuracy of SVM on (50% train, 50% test) partition: 0.962546816
479

Accuracy of SVM's 3 trials on (20% train, 80% test) partition:[0.859
15492957746475, 0.88028169014084512, 0.9662921348314607]
Mean Accuracy of SVM on (20% train, 80% test) partition: 0.901909584
85
```

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)
```

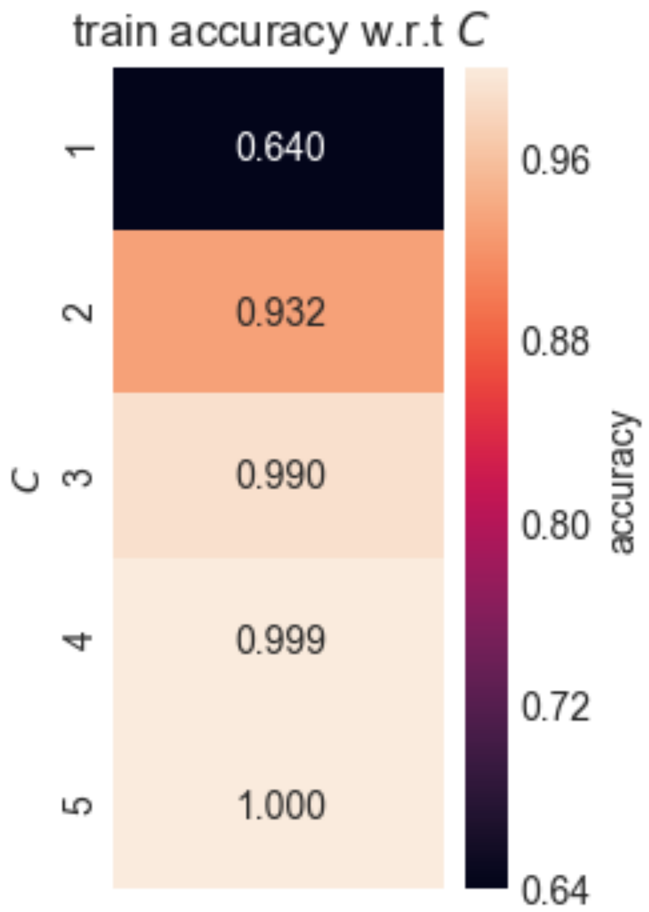
1st Run)

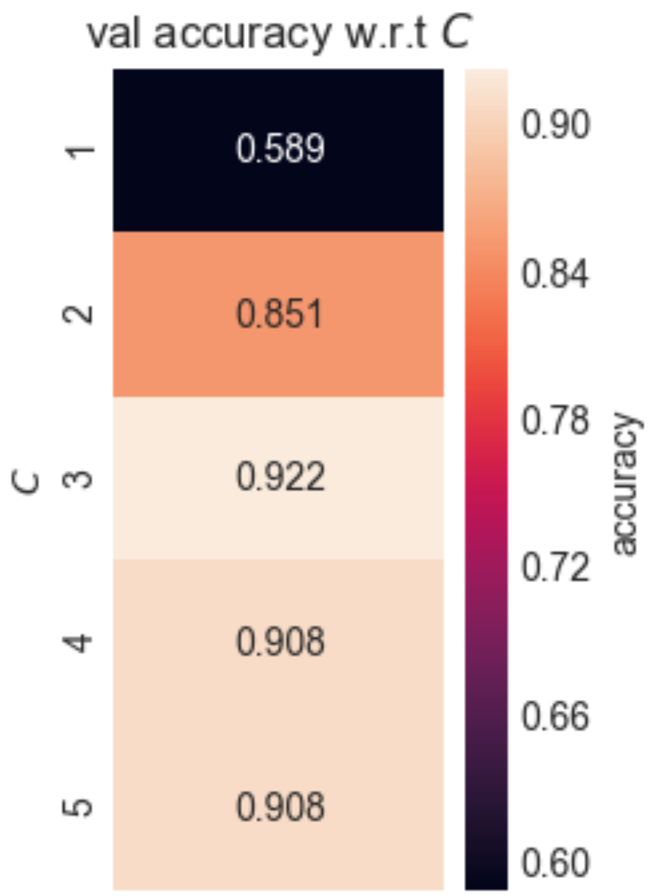
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

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)
```

```
[ 0.63989997  0.93207172  0.98974424  0.9992126   1.        ]
[ 0.58865248  0.85106383  0.92198582  0.90780142  0.90780142]
```





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 0.921985815603 from index 2.
Best D: 3
Test Accuracy Score: 1.0
```

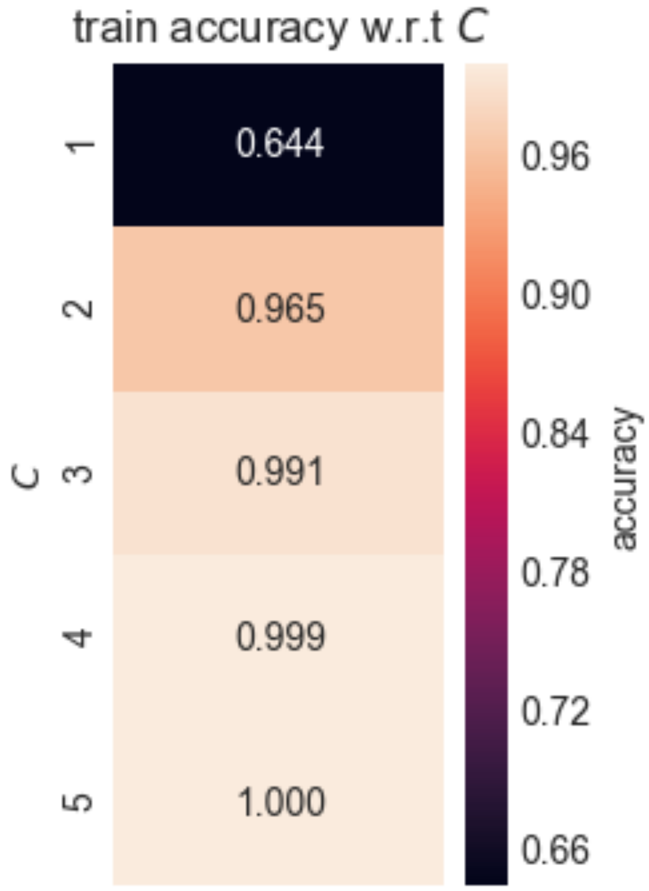
2nd Run)

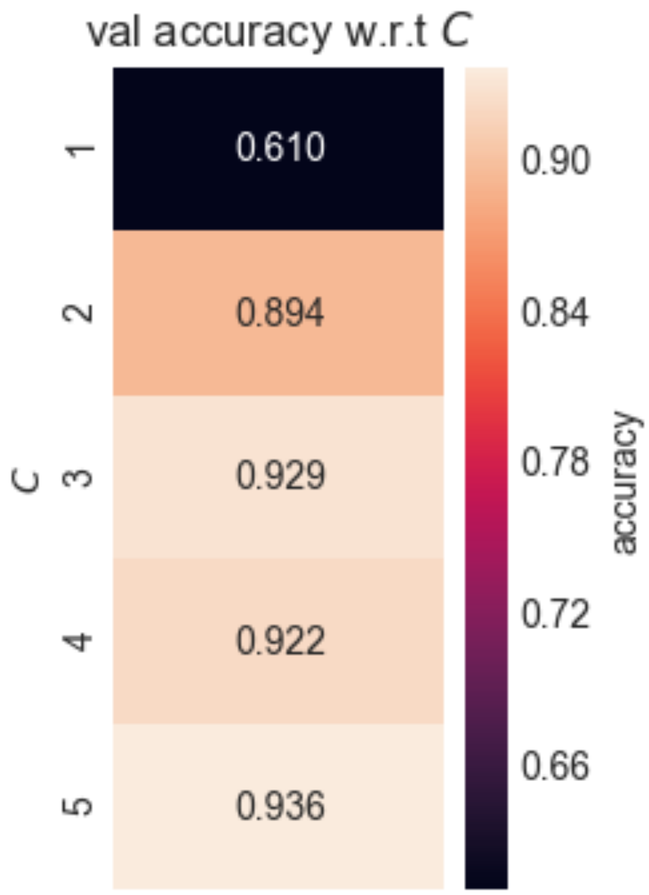
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

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)
```

```
[ 0.64394344  0.96524523  0.99131914  0.99921875  1.        ]
[ 0.60992908  0.89361702  0.92907801  0.92198582  0.93617021]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.644
0.965
0.991
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
0.610
0.894
0.929
0.922
0.936
0.90
0.84
0.78
0.72
0.66
accuracy

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 0.936170212766 from index 4.
Best D: 5
Test Accuracy Score: 0.916666666667
```

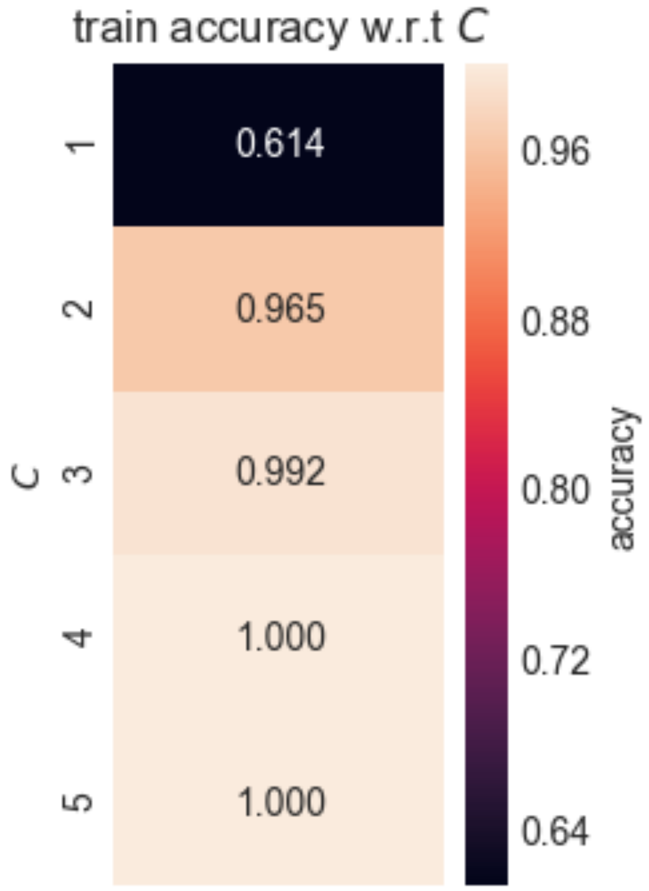
3rd Run)

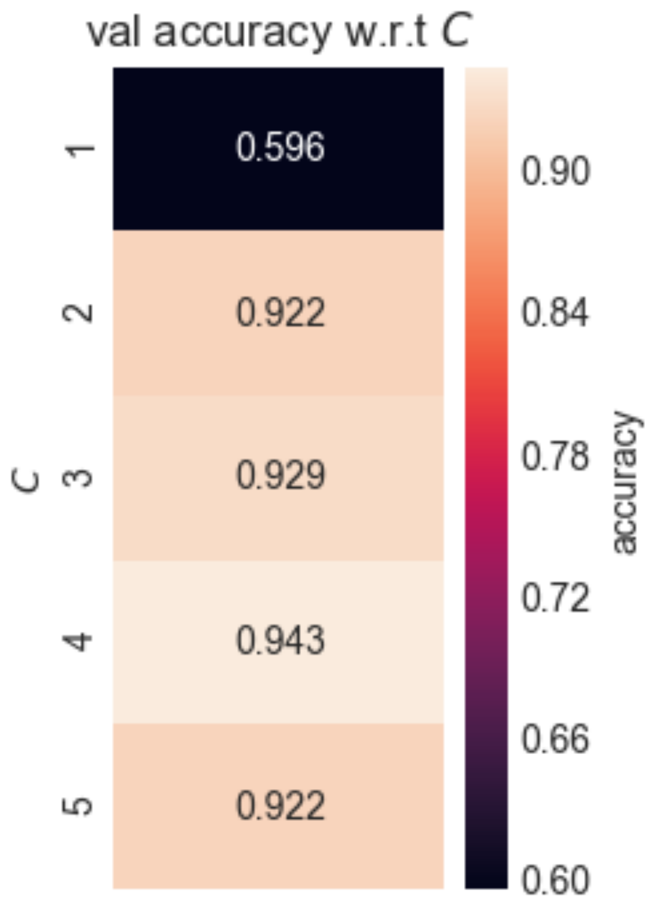
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

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)
```

```
[ 0.61387406  0.96455835  0.9921255   1.          1.        ]
[ 0.59574468  0.92198582  0.92907801  0.94326241  0.92198582]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.614
0.965
0.992
1.000
1.000
0.96
0.88
0.80
0.72
0.64
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.596
0.922
0.929
0.943
0.922
0.90
0.84
0.78
0.72
0.66
0.60
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 0.943262411348 from index 3.
Best D: 4
Test Accuracy Score: 1.0
```

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:[1.0, 0.91666666666666663, 1.0]
DT_accuracyTestList mean: 0.972222222222
```

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)
```

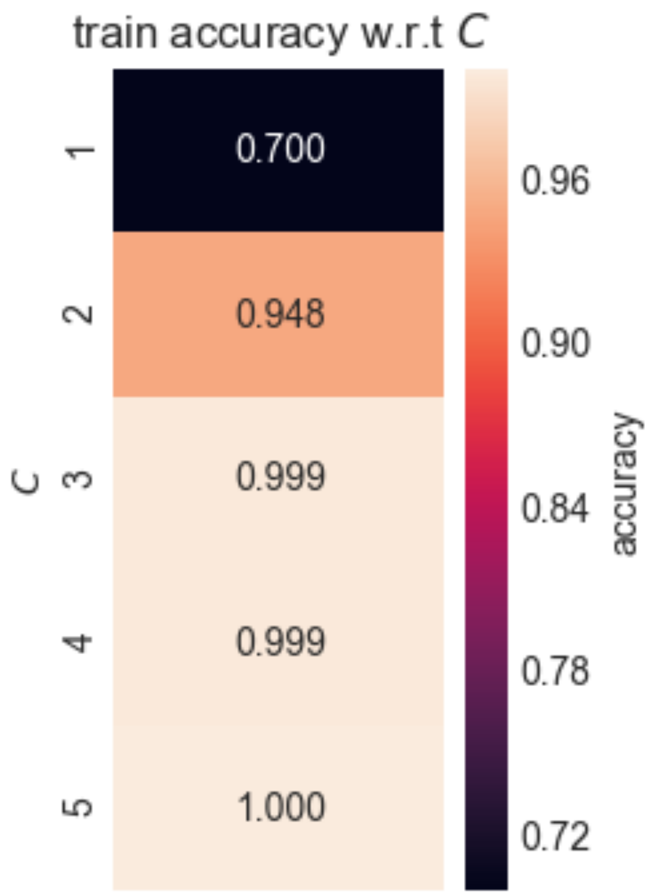
1st Run)

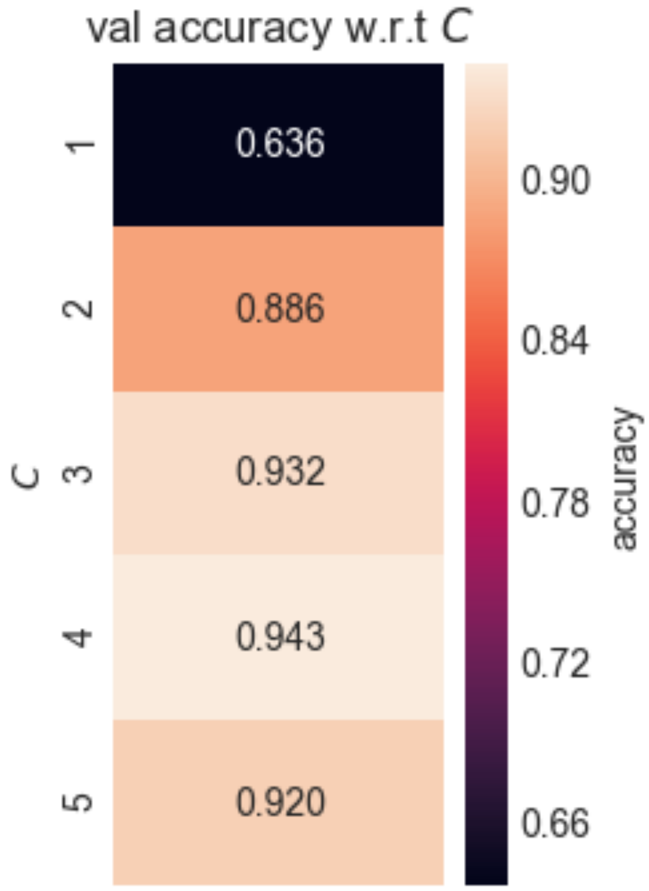
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

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)
```

```
[ 0.69951804  0.94835034  0.99873418  0.99873418  1.        ]
[ 0.63636364  0.88636364  0.93181818  0.94318182  0.92045455]
```





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 0.943181818182 from index 3.
Best D: 4
Test Accuracy Score: 0.898876404494
```

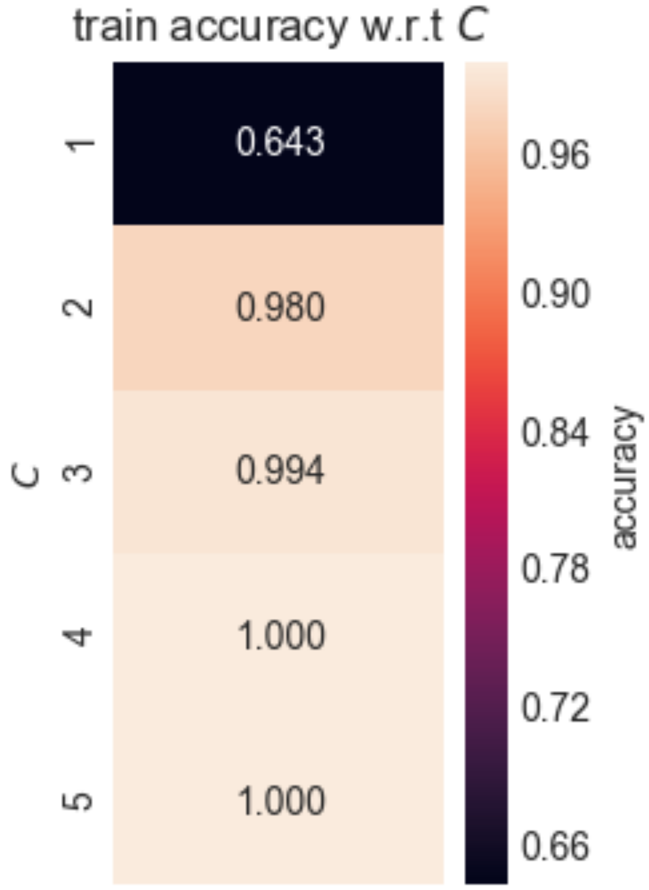
2nd Run)

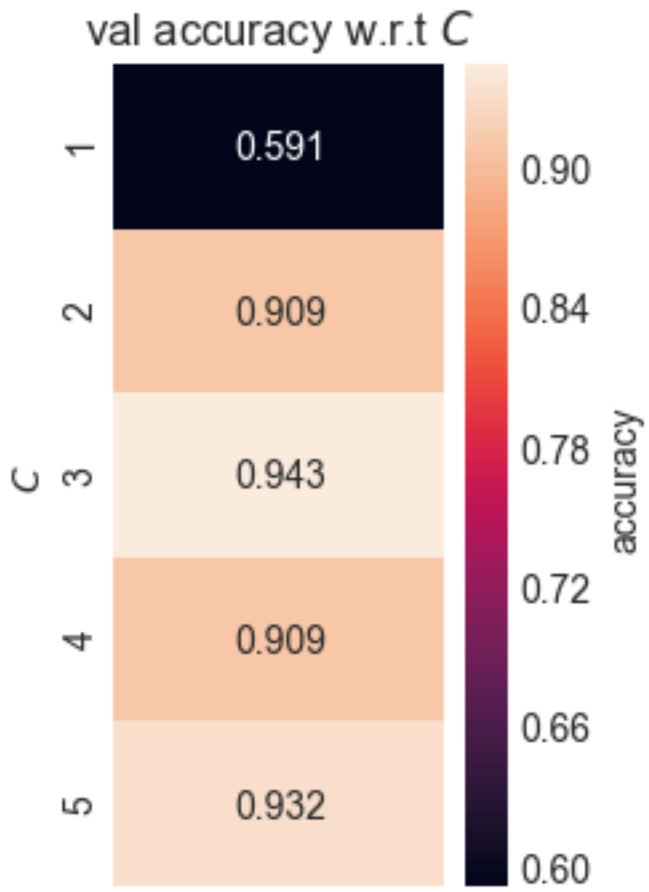
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

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)
```

```
[ 0.64273742  0.97982194  0.99370133  1.          1.        ]
[ 0.59090909  0.90909091  0.94318182  0.90909091  0.93181818]
```





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 0.943181818182 from index 2.
Best D: 3
Test Accuracy Score: 0.876404494382
```

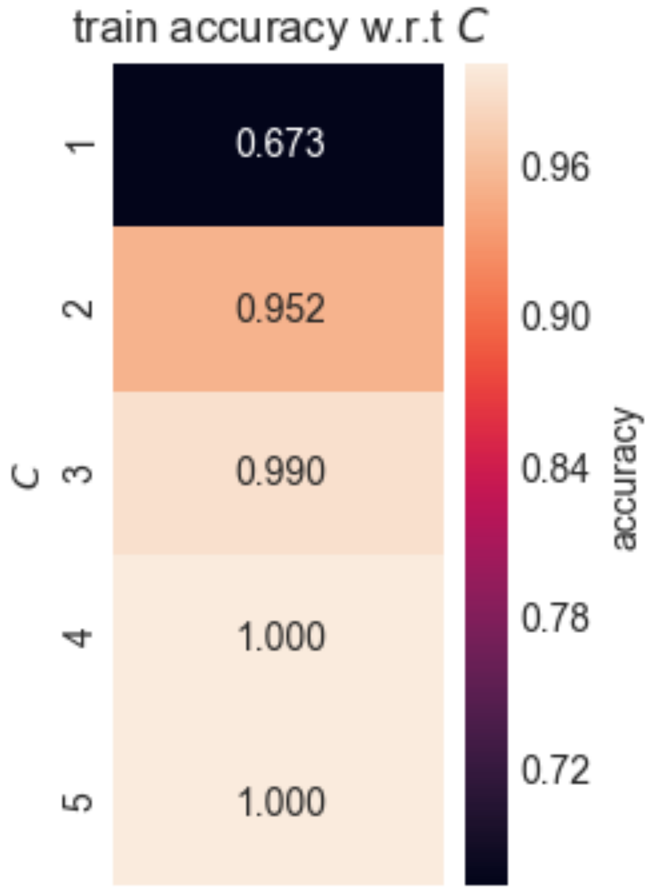
3rd Run)

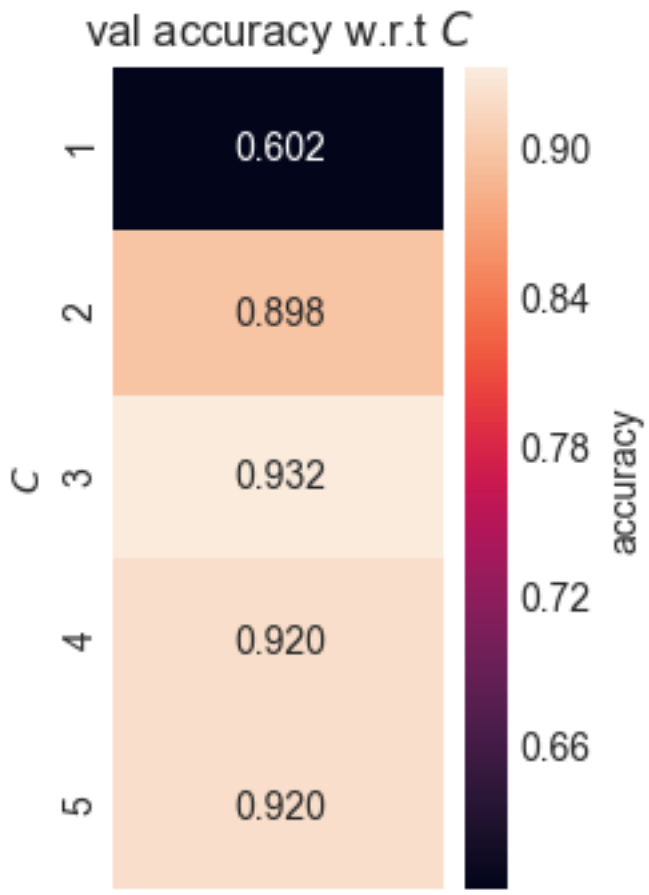
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

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)
```

```
[ 0.67308179  0.95208136  0.98991928  1.          1.        ]
[ 0.60227273  0.89772727  0.93181818  0.92045455  0.92045455]
```


train accuracy w.r.t C
0.673
0.952
0.990
1.000
1.000
1
2
3
4
5
C
0.96
0.90
0.84
0.78
0.72
accuracy


val accuracy w.r.t C
0.602
0.898
0.932
0.920
0.920
1
2
3
4
5
C
0.90
0.84
0.78
0.72
0.66
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 0.931818181818 from index 2.
Best D: 3
Test Accuracy Score: 0.955056179775
```

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:[0.898876404494382, 0.8764044943820225, 0.955056
1797752809]
DT_accuracyTestList mean: 0.910112359551
```

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)
```

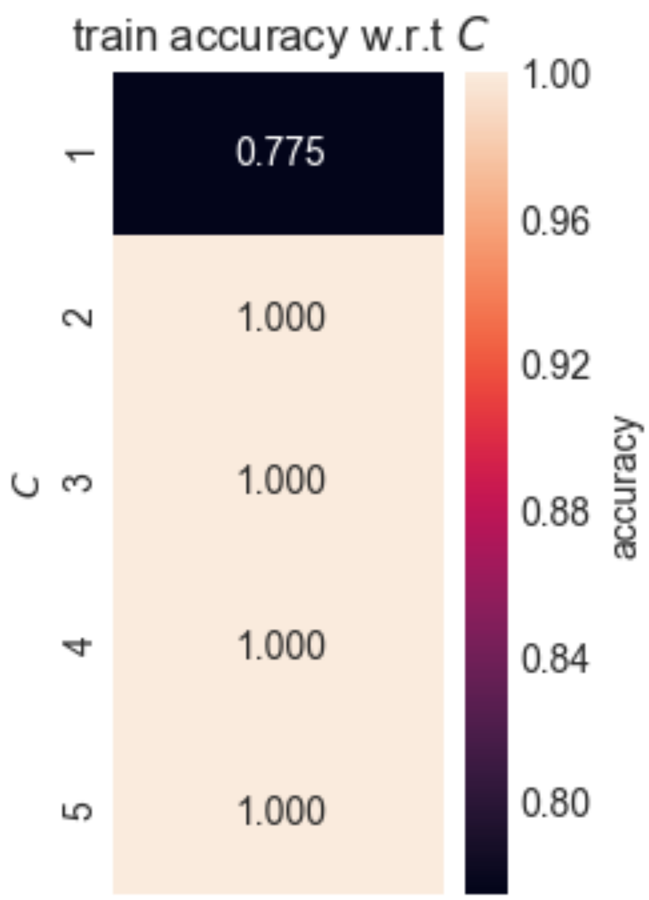
1st Run)

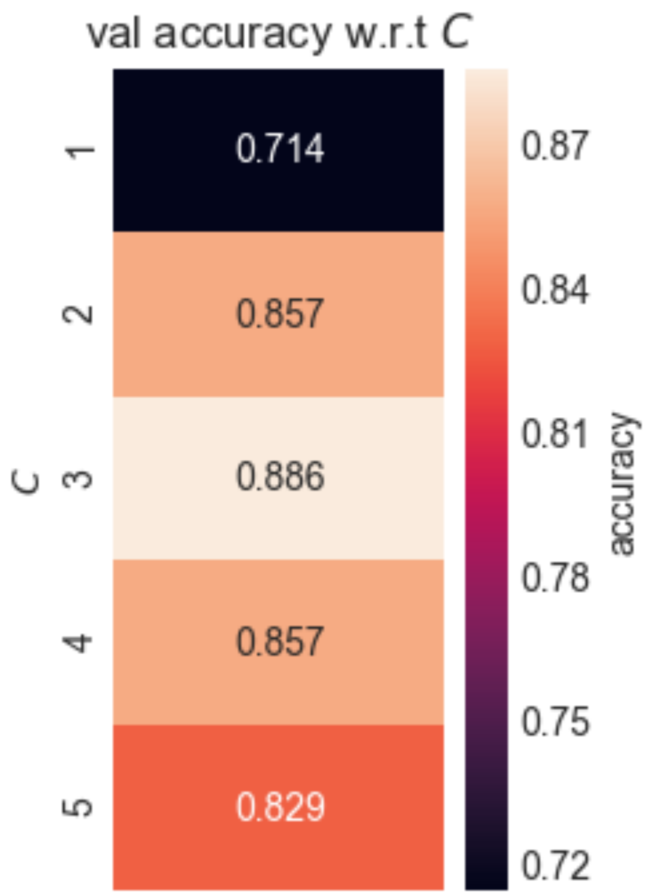
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

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)
```

```
[ 0.77475688  1.          1.          1.          1.        ]
[ 0.71428571  0.85714286  0.88571429  0.85714286  0.82857143]

/Users/rod/anaconda3/lib/python3.6/site-packages/sklearn/model_selec
tion/_split.py:605: Warning: The least populated class in y has only
7 members, which is too few. The minimum number of members in any cl
ass cannot be less than n_splits=10.
  % (min_groups, self.n_splits)), Warning)
```





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 0.885714285714 from index 2.
Best D: 3
Test Accuracy Score: 0.830985915493
```

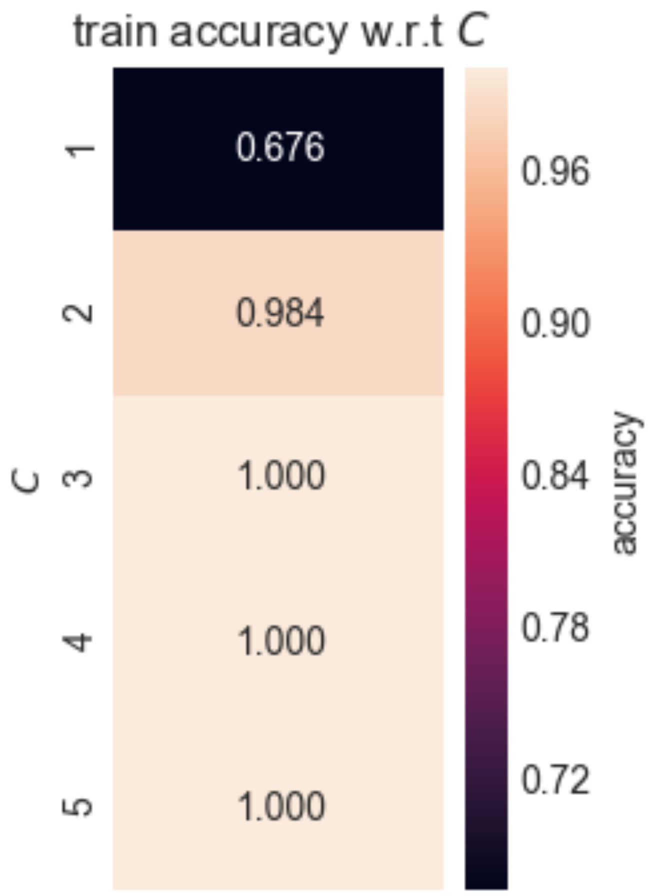
2nd Run)

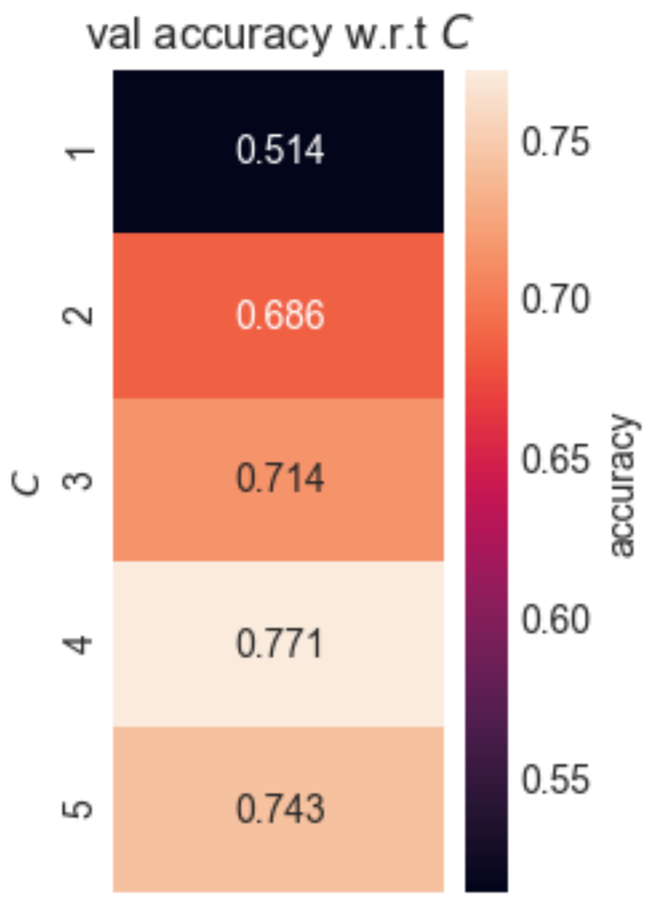
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

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)
```

```
[ 0.67635081  0.98386425  1.          1.          1.        ]
[ 0.51428571  0.68571429  0.71428571  0.77142857  0.74285714]
```





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 0.771428571429 from index 3.
Best D: 4
Test Accuracy Score: 0.901408450704
```

3rd Run)

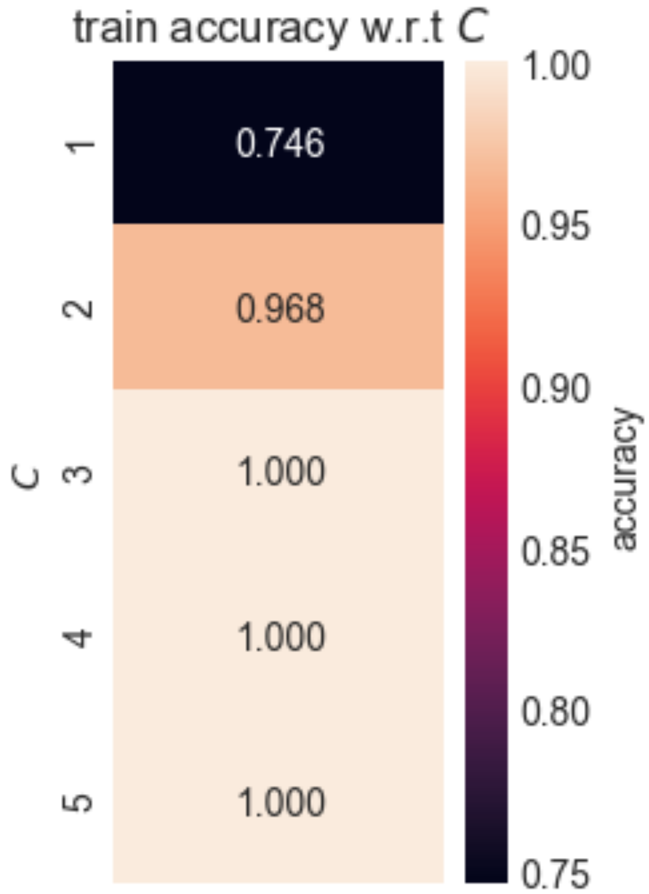
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

In [70]:

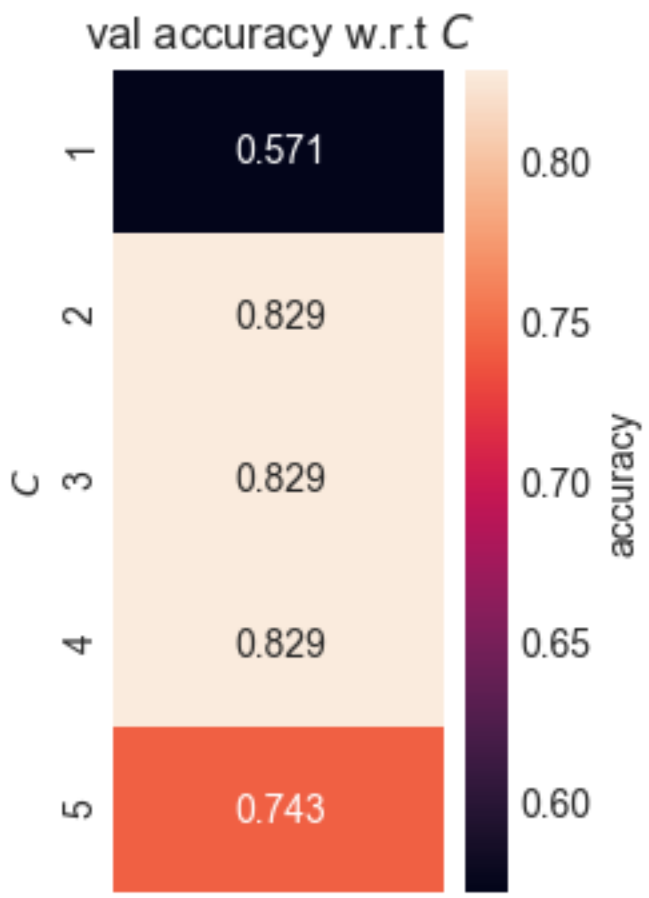
```
DT_clfGridSearch = decisionTreeTrainValidation(X3_train_val, Y3_train_val, D_lis
t, CV)
accuracyTrain = DT_clfGridSearch.cv_results_['mean_train_score']
accuracyValidation = DT_clfGridSearch.cv_results_['mean_test_score']
print(accuracyTrain)        #This is what shows up in the heat maps.
print(accuracyValidation)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

```
[ 0.7459708   0.96823375  1.          1.          1.        ]
[ 0.57142857  0.82857143  0.82857143  0.82857143  0.74285714]
```

```
/Users/rod/anaconda3/lib/python3.6/site-packages/sklearn/model_selec
tion/_split.py:605: Warning: The least populated class in y has only
7 members, which is too few. The minimum number of members in any cl
ass cannot be less than n_splits=10.
  % (min_groups, self.n_splits)), Warning)
```



train accuracy w.r.t C
C
1
2
3
4
5
0.746
0.968
1.000
1.000
1.000
1.00
0.95
0.90
0.85
0.80
0.75
accuracy



val accuracy w.r.t C
C
1
2
3
4
5
0.571
0.829
0.829
0.829
0.743
0.80
0.75
0.70
0.65
0.60
accuracy

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 0.828571428571 from index 1.
Best D: 2
Test Accuracy Score: 0.852112676056
```

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:[0.83098591549295775, 0.90140845070422537, 0.852
112676056338]
DT_accuracyTestList mean: 0.861502347418
```

Results of Decision Tree

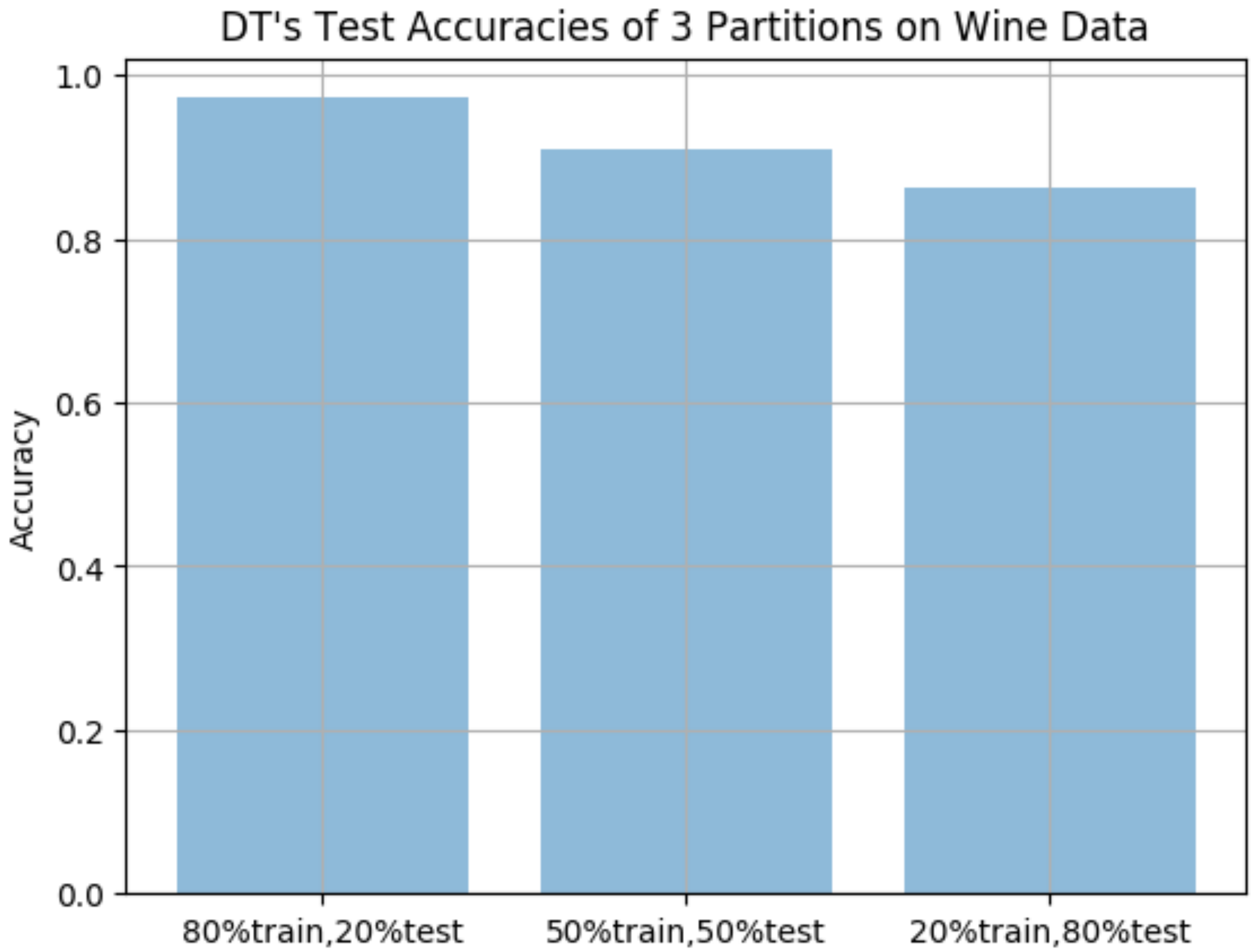
In [73]:

```
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: 0.97222222
2222
DT_accuracyTestList (50% train, 50% test) partition mean: 0.91011235
9551
DT_accuracyTestList (20% train, 80% test) partition mean: 0.86150234
7418
```

In [108]:

```
displayAccuracies('DT', 'Wine Data', DT_accuracyAverage_80_20, DT_accuracyAverag
e_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 :[1.0,
0.9166666666666663, 1.0]
Mean Accuracy of DT on (80% train, 20% test) partition: 0.9722222222
22

Accuracy of DT's 3 trials on (50% train, 50% test) partition :[0.898
876404494382, 0.8764044943820225, 0.9550561797752809]
Mean Accuracy of DT on (50% train, 50% test) partition: 0.9101123595
51

Accuracy of DT's 3 trials on (20% train, 80% test) partition:[0.8309
8591549295775, 0.9014084507042537, 0.852112676056338]
Mean Accuracy of DT on (20% train, 80% test) partition: 0.8615023474
18
```

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)
```

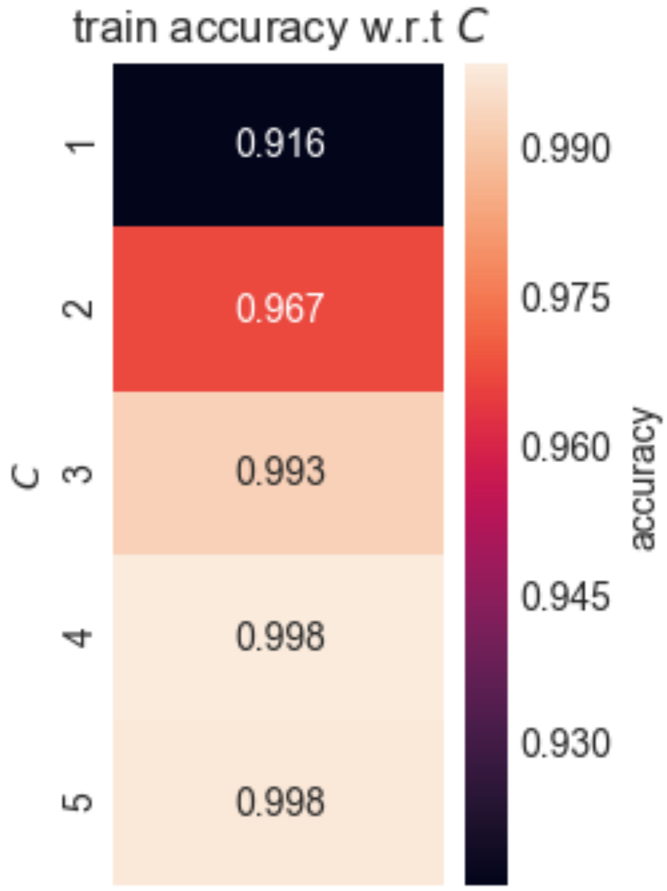
1st Run)

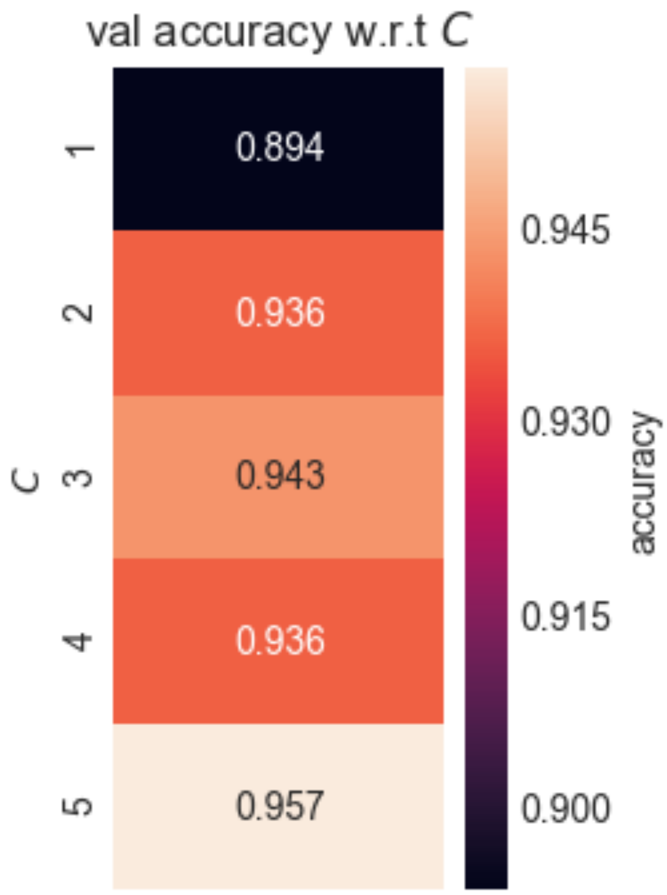
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

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)
```

```
[ 0.91564771  0.96695734  0.9928877   0.9984251   0.99763135]
[ 0.89361702  0.93617021  0.94326241  0.93617021  0.95744681]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.916
0.967
0.993
0.998
0.998
0.990
0.975
0.960
0.945
0.930
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.894
0.936
0.943
0.936
0.957
0.945
0.930
0.915
0.900
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 0.957446808511 from index 4.
Best max_depth: 5
Test Accuracy Score: 1.0
```

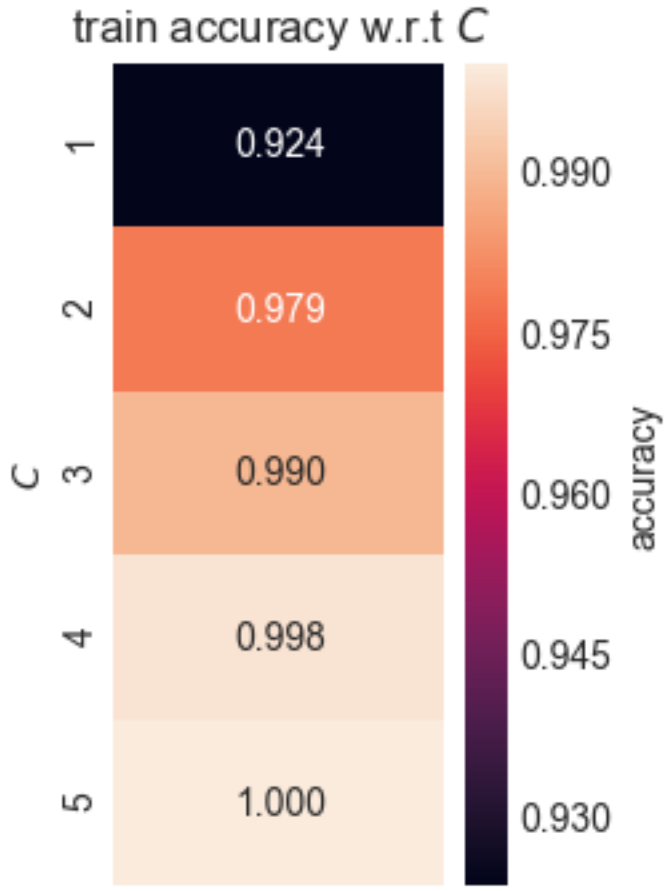
2nd Run)

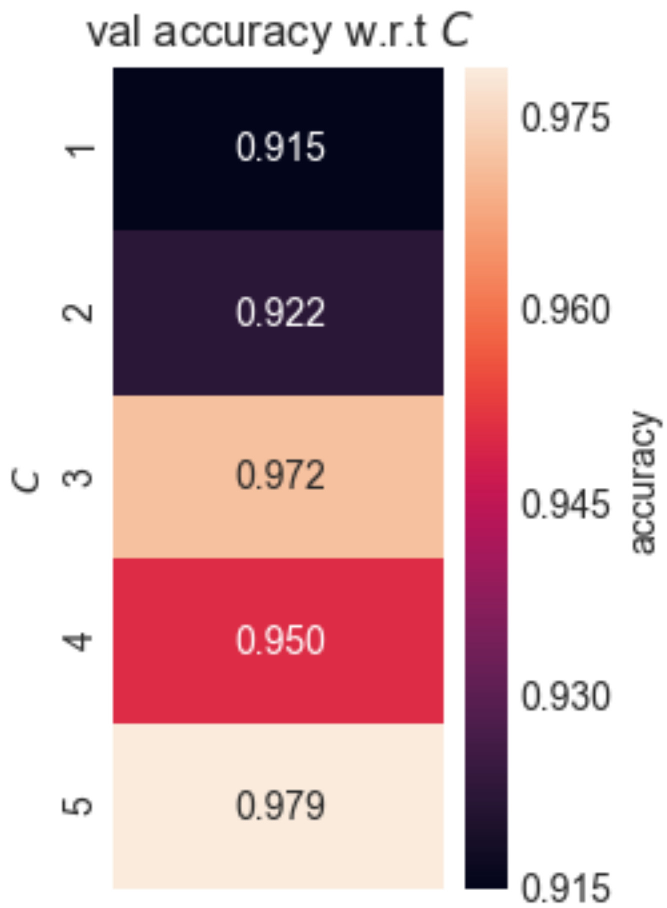
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

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)
```

```
[ 0.92354125  0.97870089  0.98973174  0.9984      1.        ]
[ 0.91489362  0.92198582  0.97163121  0.95035461  0.9787234 ]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.924
0.979
0.990
0.998
1.000
0.990
0.975
0.960
0.945
0.930
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.915
0.922
0.972
0.950
0.979
0.975
0.960
0.945
0.930
0.915
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 0.978723404255 from index 4.
Best max_depth: 5
Test Accuracy Score: 1.0
```

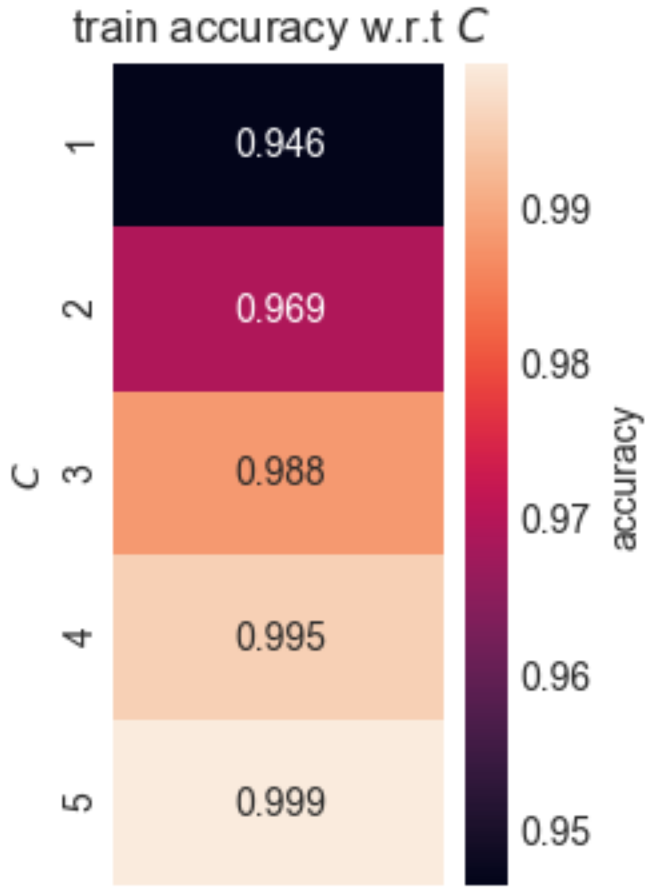
3rd Run)

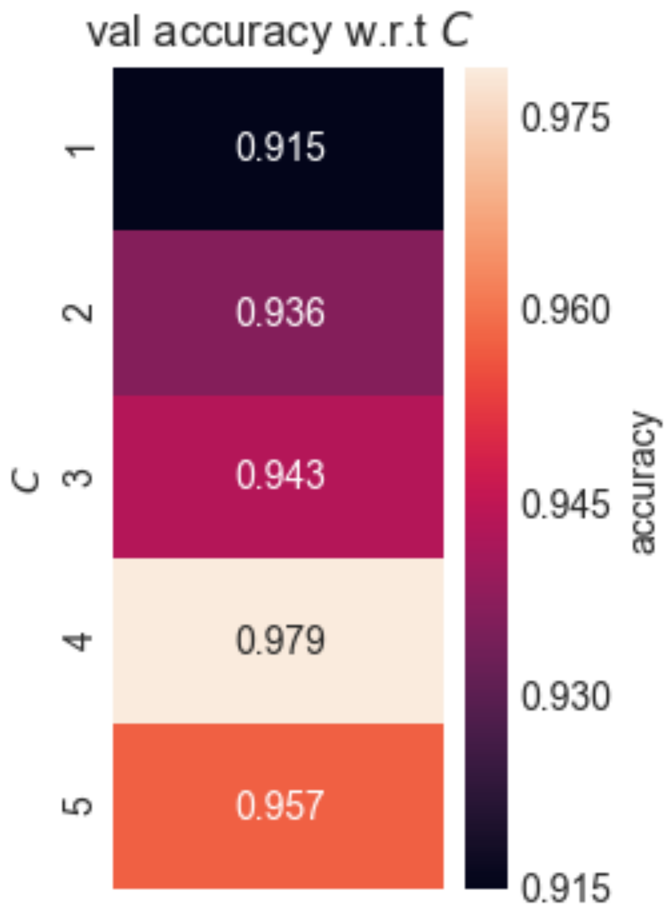
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

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)
```

```
[ 0.9464033   0.96927094  0.98818195  0.99530606  0.99920635]
[ 0.91489362  0.93617021  0.94326241  0.9787234   0.95744681]
```


train accuracy w.r.t C
0.946
0.969
0.988
0.995
0.999
C
1
2
3
4
5
0.99
0.98
0.97
0.96
0.95
accuracy


val accuracy w.r.t C
0.915
0.936
0.943
0.979
0.957
C
1
2
3
4
5
0.975
0.960
0.945
0.930
0.915
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 0.978723404255 from index 3.
Best max_depth: 4
Test Accuracy Score: 1.0
```

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:[1.0, 1.0, 1.0]
RF_accuracyTestList mean: 1.0
```

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)
```

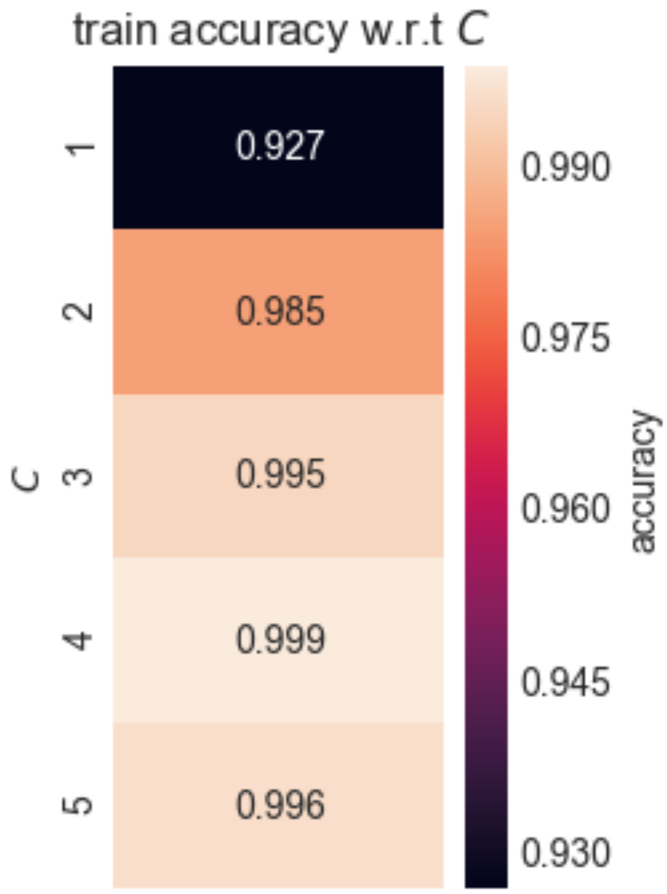
1st Run)

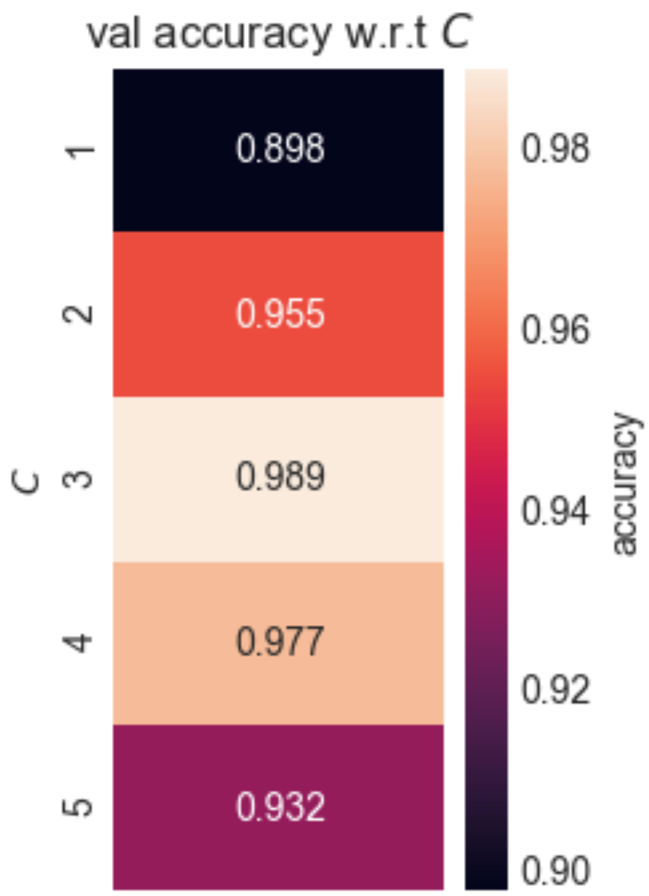
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

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)
```

```
[ 0.92670068  0.98488801  0.99488677  0.9987013   0.9962013 ]
[ 0.89772727  0.95454545  0.98863636  0.97727273  0.93181818]
```





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 0.988636363636 from index 2.
Best max_depth: 3
Test Accuracy Score: 0.955056179775
```

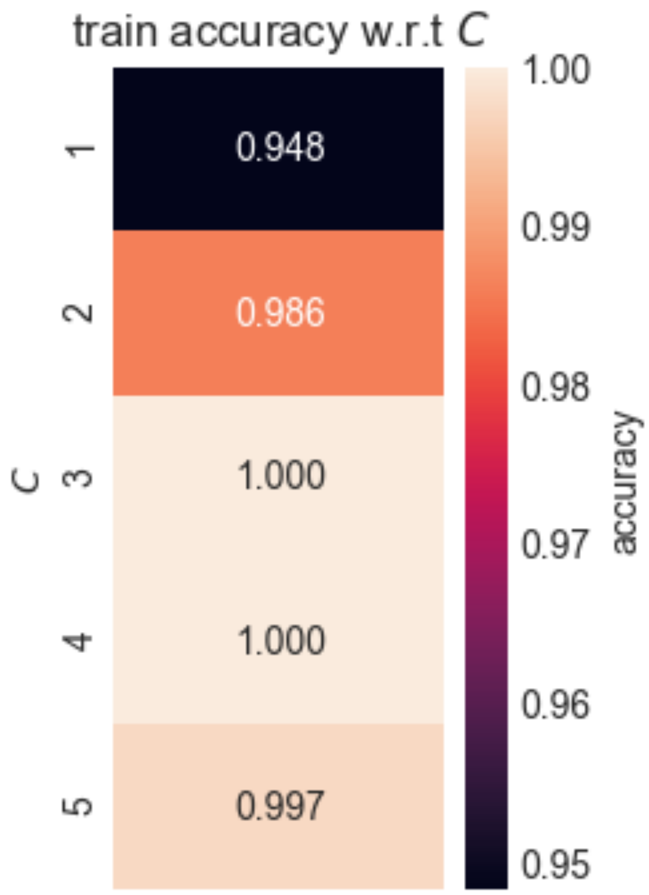
2nd Run)

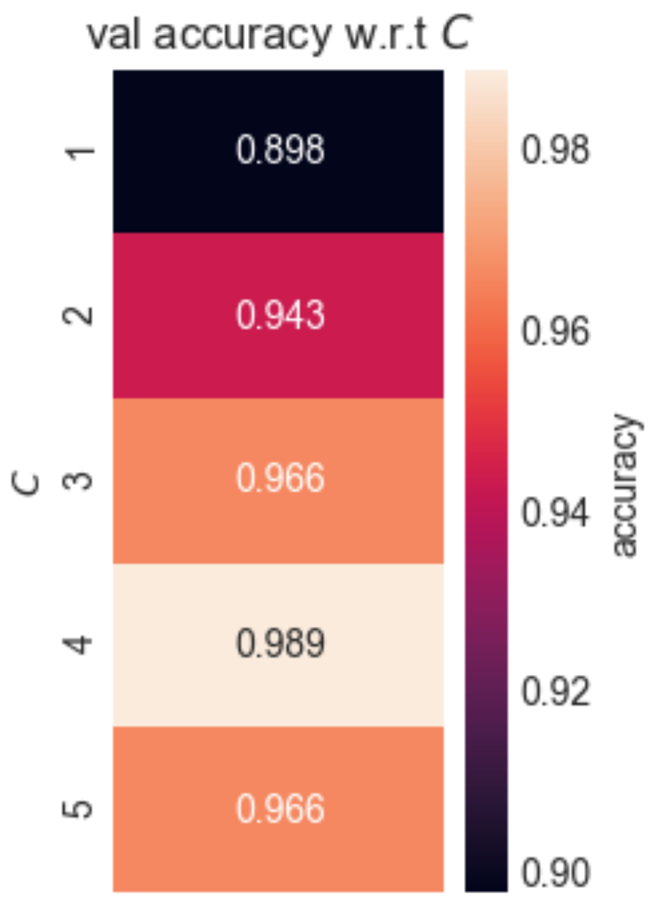
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

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)
```

```
[ 0.94827873  0.98620014  1.          1.          0.99746795]
[ 0.89772727  0.94318182  0.96590909  0.98863636  0.96590909]
```





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 0.988636363636 from index 3.
Best max_depth: 4
Test Accuracy Score: 0.876404494382
```

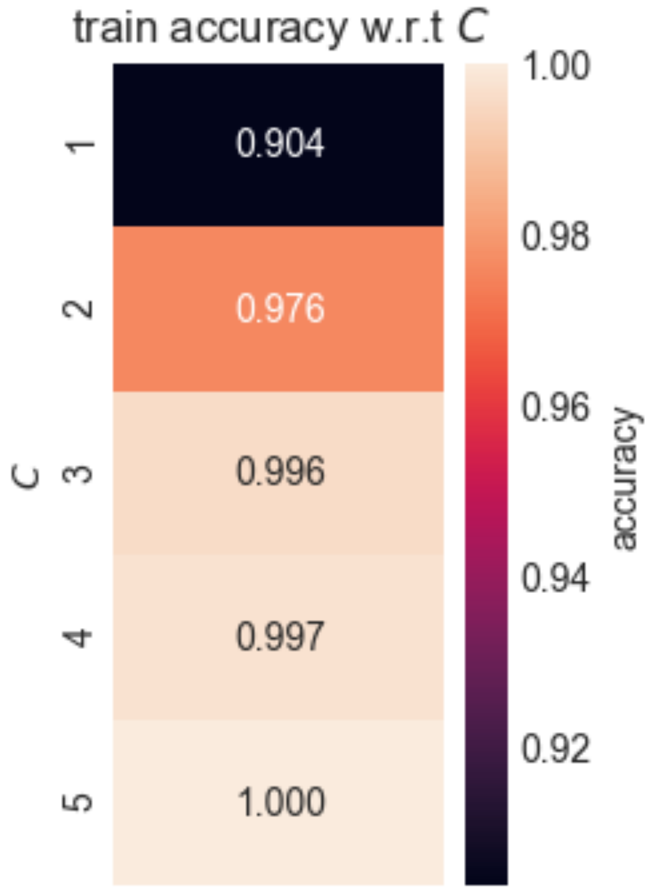
3rd Run)

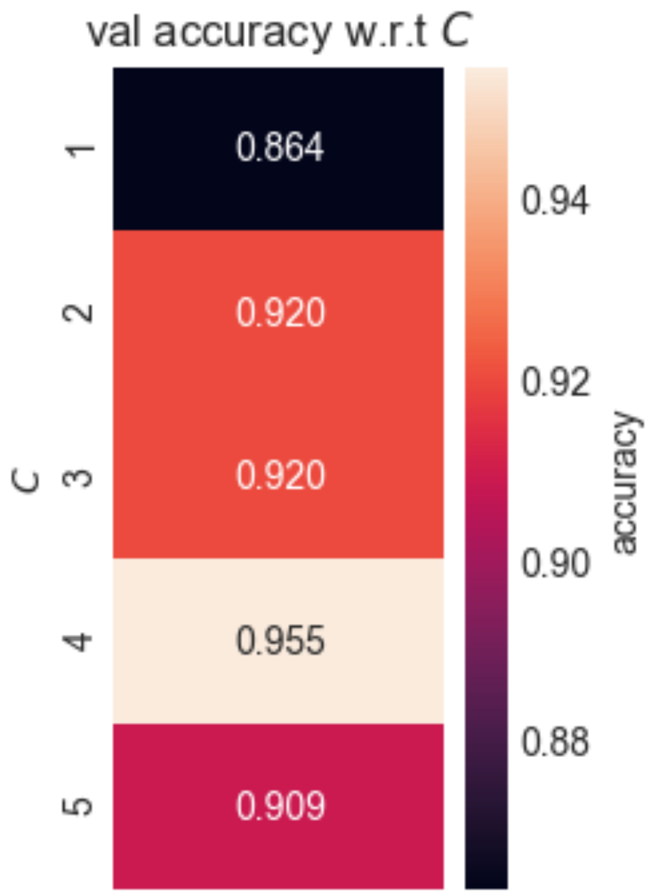
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

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)
```

```
[ 0.90376832  0.97613485  0.99623338  0.99746795  1.        ]
[ 0.86363636  0.92045455  0.92045455  0.95454545  0.90909091]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.904
0.976
0.996
0.997
1.000
1.00
0.98
0.96
0.94
0.92
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.864
0.920
0.920
0.955
0.909
0.94
0.92
0.90
0.88
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 0.954545454545 from index 3.
Best max_depth: 4
Test Accuracy Score: 0.966292134831
```

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:[0.9550561797752809, 0.8764044943820225, 0.96629
21348314607]
RF_accuracyTestList mean: 0.932584269663
```

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)
```

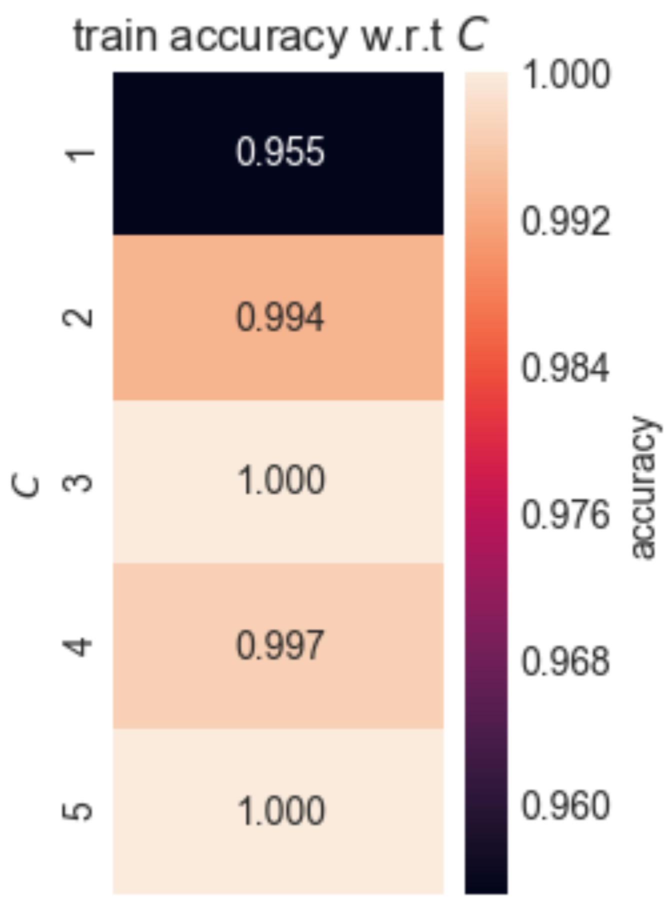
1st Run)

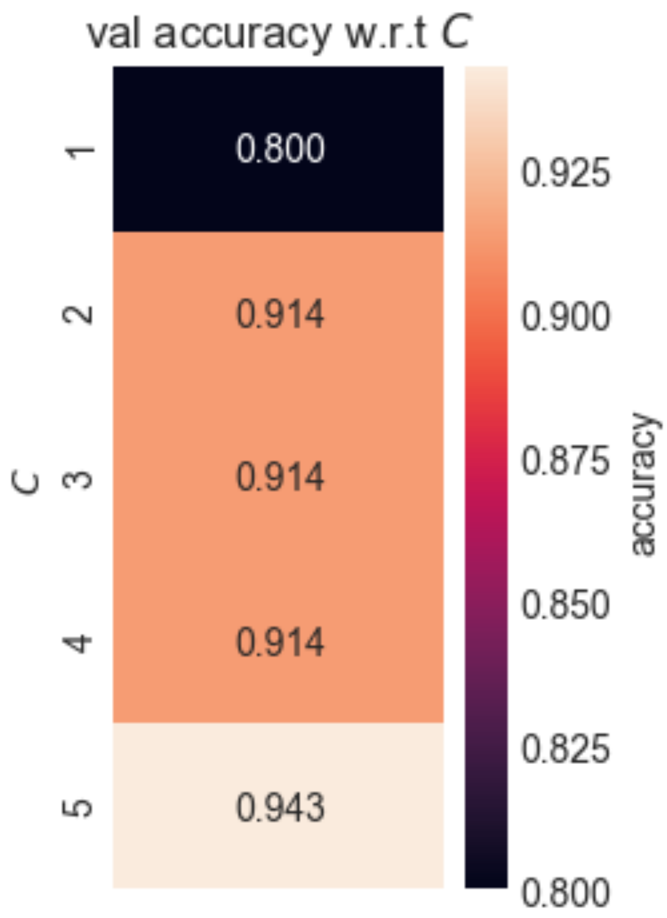
First run uses the variables; X1_train_val, Y1_train_val, X1_test, Y1_test

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)
```

```
/Users/rod/anaconda3/lib/python3.6/site-packages/sklearn/model_selec
tion/_split.py:605: Warning: The least populated class in y has only
7 members, which is too few. The minimum number of members in any cl
ass cannot be less than n_splits=10.
  % (min_groups, self.n_splits)), Warning)

[ 0.95514113  0.99354839  1.          0.99677419  1.        ]
[ 0.8         0.91428571  0.91428571  0.91428571  0.94285714]
```





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 0.942857142857 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.894366197183
```

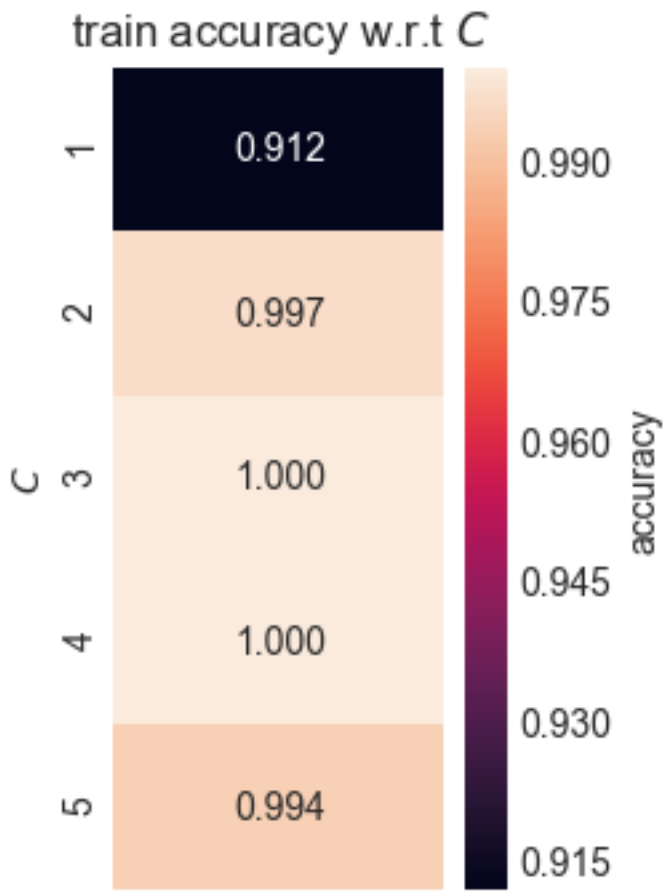
2nd Run)

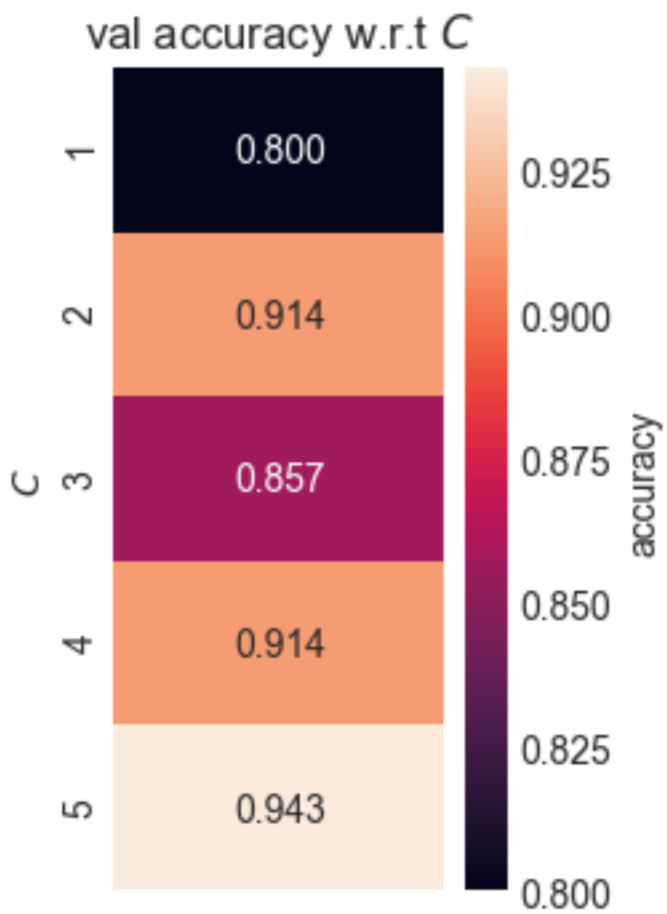
Second run uses the variables; X2_train_val, Y2_train_val, X2_test, Y2_test

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)
```

```
[ 0.91188844  0.996875    1.          1.          0.99375   ]
[ 0.8         0.91428571  0.85714286  0.91428571  0.94285714]
```





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 0.942857142857 from index 4.
Best max_depth: 5
Test Accuracy Score: 0.887323943662
```

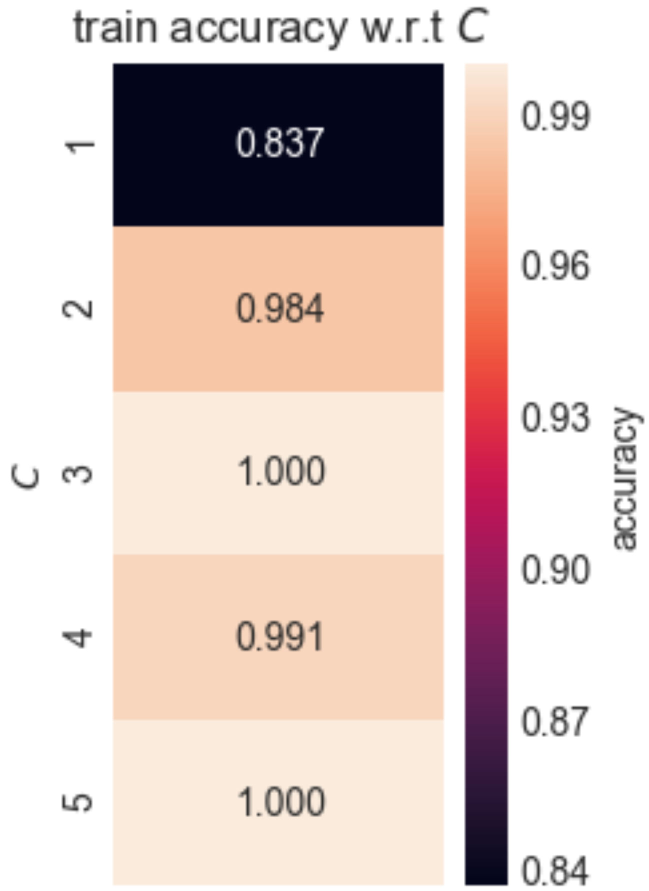
3rd Run)

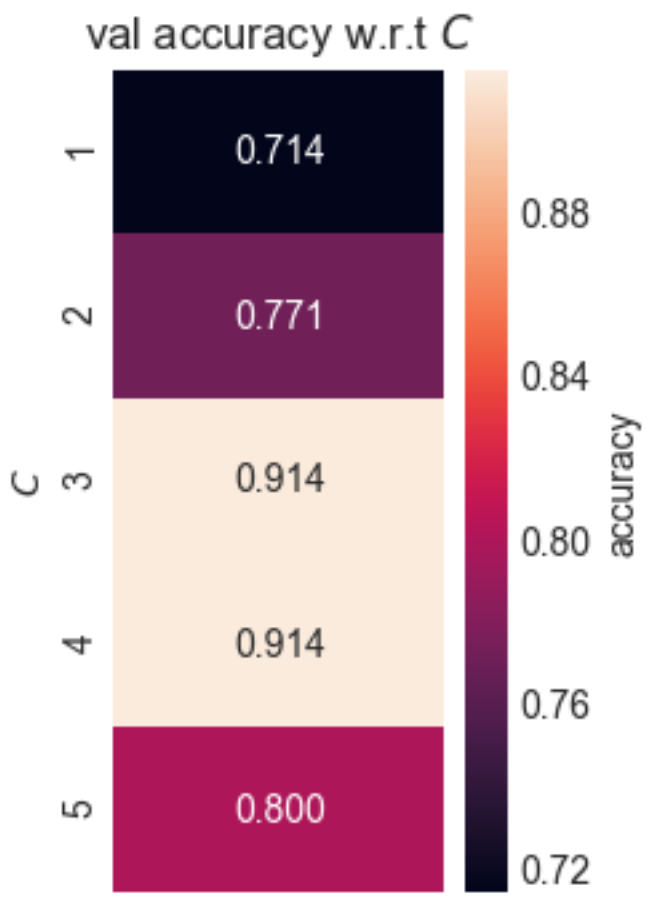
Third run uses the variables; X3_train_val, Y3_train_val, X3_test, Y3_test

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)
```

```
/Users/rod/anaconda3/lib/python3.6/site-packages/sklearn/model_selec
tion/_split.py:605: Warning: The least populated class in y has only
7 members, which is too few. The minimum number of members in any cl
ass cannot be less than n_splits=10.
  % (min_groups, self.n_splits)), Warning)

[ 0.83715481  0.98406647  1.          0.99071359  1.        ]
[ 0.71428571  0.77142857  0.91428571  0.91428571  0.8       ]
```


train accuracy w.r.t C
C
1
2
3
4
5
0.837
0.984
1.000
0.991
1.000
0.99
0.96
0.93
0.90
0.87
0.84
accuracy


val accuracy w.r.t C
C
1
2
3
4
5
0.714
0.771
0.914
0.914
0.800
0.88
0.84
0.80
0.76
0.72
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 0.914285714286 from index 2.
Best max_depth: 3
Test Accuracy Score: 0.887323943662
```

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:[0.89436619718309862, 0.88732394366197187, 0.887
32394366197187]
RT_accuracyTestList mean: 0.889671361502
```

Results of Random Forest

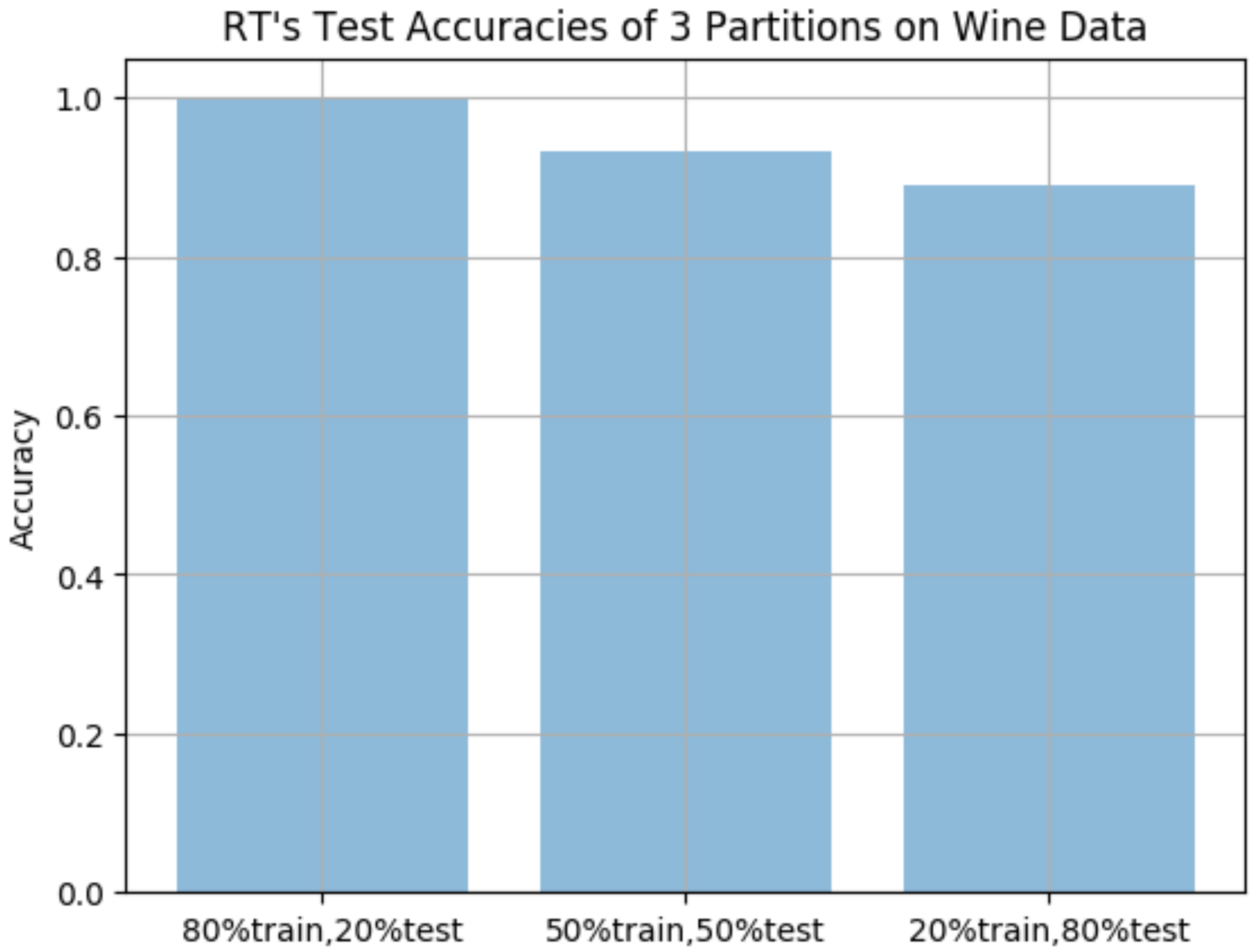
In [101]:

```
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: 1.0
RT_accuracyTestList (50% train, 50% test) partition mean: 0.93258426
9663
RT_accuracyTestList (20% train, 80% test) partition mean: 0.88967136
1502
```

In [109]:

```
displayAccuracies('RT', 'Wine Data', RF_accuracyAverage_80_20, RF_accuracyAverag
e_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 :[1.0,
1.0, 1.0]
Mean Accuracy of RT on (80% train, 20% test) partition: 1.0

Accuracy of RT's 3 trials on (50% train, 50% test) partition :[0.955
0561797752809, 0.8764044943820225, 0.9662921348314607]
Mean Accuracy of RT on (50% train, 50% test) partition: 0.9325842696
63

Accuracy of RT's 3 trials on (20% train, 80% test) partition:[0.8943
6619718309862, 0.8873239436619718 7, 0.8873239436619718 7]
Mean Accuracy of RT on (20% train, 80% test) partition: 0.8896713615
02
```

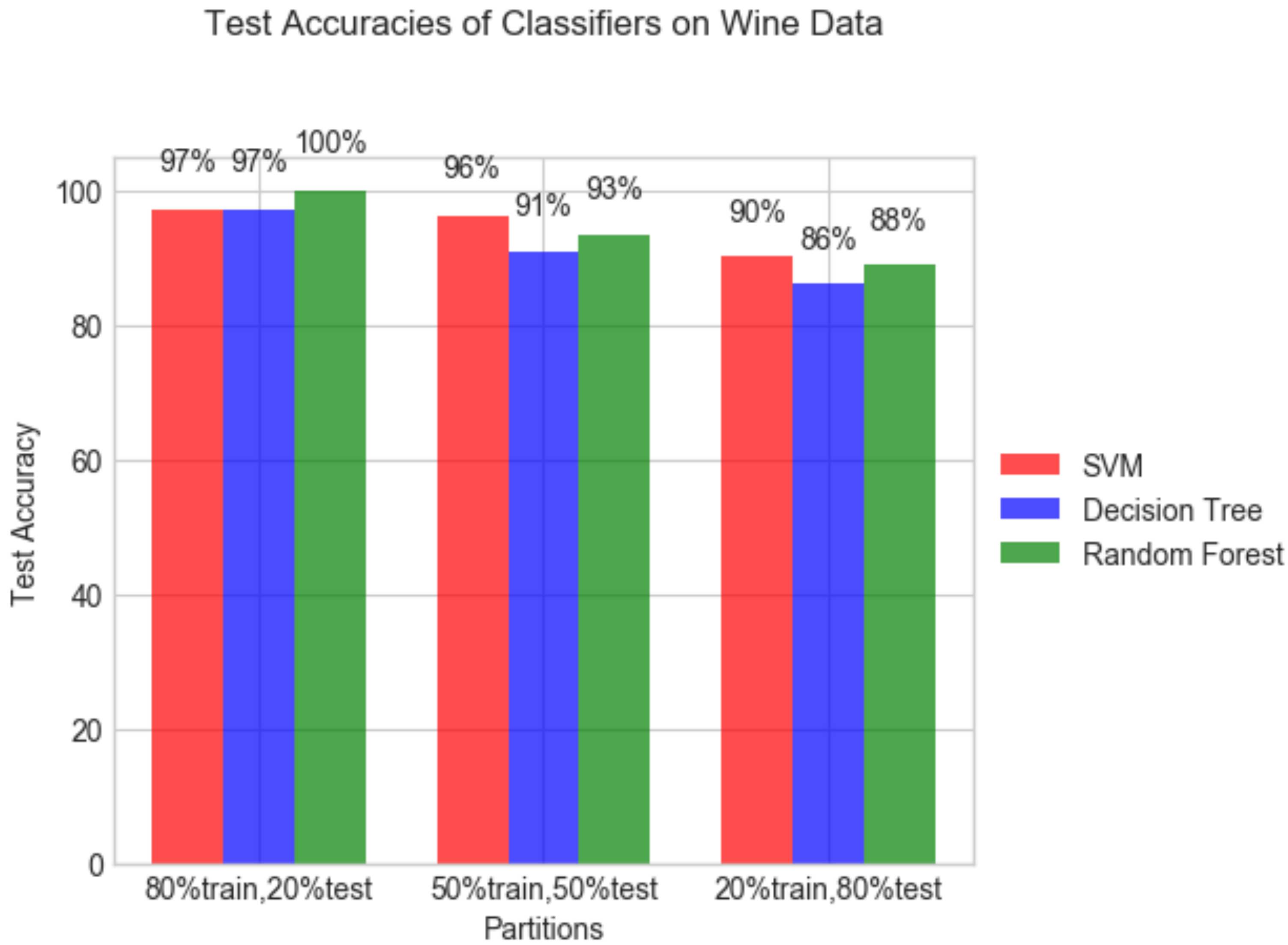
Results

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 [104]:

```
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 Wine 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 Wine Data
97%
97%
100%
96%
91%
93%
90%
86%
88%
Test Accuracy
0
20
40
60
80
100
80%train,20%test
50%train,50%test
20%train,80%test
Partitions
SVM
Decision Tree
Random Forest