In [10]:
```
import pandas as pd
import numpy as np
import urllib
import scipy.optimize
import random
import urllib.request
from sklearn import ensemble
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
#here we get the data
df = pd.read_csv('iris.csv',
                 sep=r'\s*,\s*',engine = 'python', na_values = '?')
df.dropna()
X_Y_chart = pd.get_dummies(df, drop_first=False)
X_Y_chart.drop(X_Y_chart.columns[len(X_Y_chart.columns)-1], axis=1, inpl
ace=True)
print(df.shape)
print(X_Y_chart.shape)
X_and_Y = X_Y_chart.values
#np.random.shuffle(X_and_Y)
X = X_and_Y[:, 0:-1]
Y = X_and_Y[:, -1]
print(X_Y_chart[:0])
import matplotlib.pyplot as plt
import seaborn as sns
def draw_heatmap(acc, acc_desc, k_list):
    plt.figure(figsize = (2,4))
    ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=k_list, xti
cklabels=[])
    ax.collections[0].colorbar.set_label("accuracy")
    ax.set(ylabel='$params$')
    plt.title(acc_desc)
    sns.set_style("whitegrid", {'axes.grid' : False})
    plt.show()
```

```
(150, 5)
(150, 5)
Empty DataFrame
Columns: [l1, w1, l2, w2, versicolor_Iris-versicolor]
Index: []
```

In [11]:
```
X_and_Y = X_Y_chart.values
#np.random.shuffle(X_and_Y)
X = X_and_Y[:, 0:-1]
Y = X_and_Y[:, -1]
#Y_front = X_and_Y[:,:7]
#Y_back = X_and_Y[:,8:]
#Y = np.concatenate((Y_front, Y_back), axis = 1)
#print(X.shape, Y_front.shape, Y_back.shape)
#print(X[0])
#print(Y[0])
X_train_val = X[:int(0.8*len(X))]
X_test = X[int(0.8*len(X)):]
Y_train_val = Y[:int(0.8*len(Y))]
Y_test = Y[int(0.8*len(Y)):]
print(X_train_val.shape, X_test.shape, Y_train_val.shape, Y_test.shape)
```

```
(120, 4) (30, 4) (120,) (30,)
```

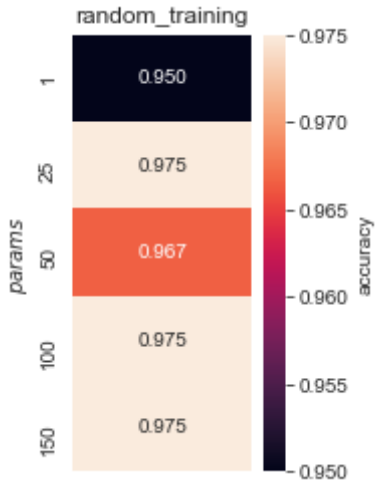
In [12]:
```
random_train_acc = []
adaboost_train_acc = []
gradient_train_acc = []
bagging_train_acc = []
decision_train_acc = []
random_test_acc = []
adaboost_test_acc = []
gradient_test_acc = []
bagging_test_acc = []
decision_test_acc = []
i = 1;
while i<4:
    #here we split the data again
    np.random.shuffle(X_and_Y)
    X = X_and_Y[:, 0:-1]
    Y = X_and_Y[:, -1]
    X_train_val = X[:int(0.8*len(X))]
    X_test = X[int(0.8*len(X)):]
    Y_train_val = Y[:int(0.8*len(Y))]
    Y_test = Y[int(0.8*len(Y)):]
    print(i)
    #then we run it on random forest
    random = RandomForestClassifier()
    print("\n RANDOM FOREST CLASSIFIER")
    #first grid search / cross validation for best parameter
    n_list = [1,25,50,100,150]
    parameters_random = {'n_estimators' : n_list}
    clf_random = GridSearchCV(random, parameters_random, cv=3)
    clf_random.fit(X_train_val, Y_train_val)
    #display result with heatmap
    print(clf_random.cv_results_['mean_test_score'])
    draw_heatmap(clf_random.cv_results_['mean_test_score'].reshape(len(n
_list),1), "random_training", n_list)
    #choose the best parameter and train again
    random = RandomForestClassifier(n_estimators = clf_random.best_param
s_['n_estimators'])
    random = random.fit(X_train_val, Y_train_val)
    #calculate the test error
    test_predict = random.predict(X_test)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    train_predict = random.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    print("random test accuracy with "+str(clf_random.best_params_['n_es
timators'])+" is: " + str(test_acc))
    random_test_acc.append(test_acc)
    random_train_acc.append(train_acc)

    #run it with Adaboost
    dt = DecisionTreeClassifier()
    adaboost = AdaBoostClassifier(base_estimator=dt)
    print("\n ADABOOST CLASSIFIER WITH DECISION TREE")
    #first grid search / cross validation for best parameter
    e_list = [10,60,120,200,300]
    parameters_adaboost = {'n_estimators' : e_list}
```

```
    clf_adaboost = GridSearchCV(adaboost, parameters_adaboost, cv=3)
    clf_adaboost.fit(X_train_val,Y_train_val)
    #display result with heatmap
    print(clf_adaboost.cv_results_['mean_test_score'])
    draw_heatmap(clf_adaboost.cv_results_['mean_test_score'].reshape(len
(e_list),1), "adaboost_training", e_list)
    #choose the best parameter and train again
    adaboost = AdaBoostClassifier(base_estimator=dt, n_estimators=clf_ad
aboost.best_params_['n_estimators'])
    adaboost = adaboost.fit(X_train_val, Y_train_val)
    #calculate the test error
    test_predict = adaboost.predict(X_test)
    train_predict = adaboost.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    print("Adaboost test accuracy with "+str(clf_adaboost.best_params_[
'n_estimators'])+" is: " + str(test_acc))
    adaboost_test_acc.append(test_acc)
    adaboost_train_acc.append(train_acc)

    #run it with gradient
    gradient = ensemble.GradientBoostingClassifier()
    print("\n GRADIENT BOOSTING CLASSIFIER")
    parameters_gradient = {'n_estimators': e_list}
    #first grid search / cross validation for best parameter
    clf_gradient = GridSearchCV(gradient, parameters_gradient, cv=3)
    clf_gradient.fit(X_train_val, Y_train_val)
    #display result with heatmap
    print(clf_gradient.cv_results_['mean_test_score'])
    draw_heatmap(clf_gradient.cv_results_['mean_test_score'].reshape(len
(e_list),1), "gradient_training", e_list)
    #choose the best parameter and train again
    gradient = ensemble.GradientBoostingClassifier(n_estimators=clf_grad
ient.best_params_['n_estimators'])
    gradient = gradient.fit(X_train_val,Y_train_val)
    #calculate the test error
    test_predict = gradient.predict(X_test)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    train_predict = gradient.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    print("Gradient test accuracy with "+str(clf_gradient.best_params_[
'n_estimators'])+" is: " + str(test_acc))
    gradient_test_acc.append(test_acc)
    gradient_train_acc.append(train_acc)

    #run it with Bagging Family with KNeighborClassifer
    bagging = BaggingClassifier(KNeighborsClassifier())
    print("\n BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER")
    s_list = [0.15,0.3,0.45,0.6,0.75]
    parameters_bagging = {'max_samples' : s_list}
    clf_bagging = GridSearchCV(bagging, parameters_bagging, cv=3)
    clf_bagging.fit(X_train_val, Y_train_val)
    print("max_samples: " + str(clf_bagging.cv_results_['mean_test_scor
```

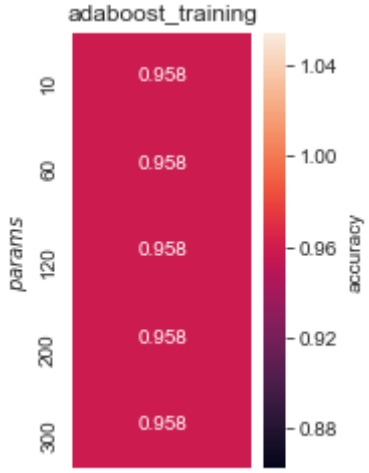
```
e']))
    draw_heatmap(clf_bagging.cv_results_['mean_test_score'].reshape(len(
s_list),1),"Kneighbor_s_list_training", s_list)
    #choose the best parameter and train again for max_features
    bagging = BaggingClassifier(KNeighborsClassifier(), max_samples=clf_
bagging.best_params_['max_samples'])
    best_max_samples = clf_bagging.best_params_['max_samples']
    f_list = [0.25,0.4,0.5,0.6,0.75]
    parameters_bagging = {'max_features': f_list}
    clf_bagging = GridSearchCV(bagging, parameters_bagging, cv=3)
    clf_bagging.fit(X_train_val, Y_train_val)
    print("max_features: " + str(clf_bagging.cv_results_['mean_test_scor
e']))
    draw_heatmap(clf_bagging.cv_results_['mean_test_score'].reshape(len(
f_list),1),"Kneighbor_f_list_training", f_list)
    #choose the best parameter and train again for max_features
    bagging = BaggingClassifier(KNeighborsClassifier(), max_samples=best
_max_samples,
                                max_features= clf_bagging.best_params_[
'max_features'])
    bagging = bagging.fit(X_train_val, Y_train_val)
    test_predict = bagging.predict(X_test)
    train_predict = bagging.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    print("Kneighbor Bagging test accuracy with "+str(best_max_samples)+

          " and "+ str(clf_bagging.best_params_['max_features'])+" is: "
 + str(test_acc))
    bagging_test_acc.append(test_acc)
    bagging_train_acc.append(train_acc)

    #run it with Bagging Family with Decision tree
    decision = BaggingClassifier()
    parameters_decision = {'max_samples' : s_list}
    print("\n BAGGING CLASSIFIER WITH DECISION TREE")
    clf_decision = GridSearchCV(decision, parameters_decision, cv=3)
    clf_decision.fit(X_train_val, Y_train_val)
    print("max_samples: " + str(clf_decision.cv_results_['mean_test_scor
e']))
    draw_heatmap(clf_decision.cv_results_['mean_test_score'].reshape(len
(s_list),1),"Decision_s_list_training", s_list)
    #choose the best parameter and train again for max_features
    decision = BaggingClassifier(max_samples=clf_decision.best_params_[
'max_samples'])
    best_max_samples = clf_decision.best_params_['max_samples']
    parameters_decision = {'max_features': f_list}
    clf_decision = GridSearchCV(decision, parameters_decision, cv=3)
    clf_decision.fit(X_train_val, Y_train_val)
    print("max_features: " + str(clf_decision.cv_results_['mean_test_sco
re']))
    draw_heatmap(clf_decision.cv_results_['mean_test_score'].reshape(len
(f_list),1),"Decision_f_list_training", f_list)
    #choose the best parameter and train again for max_features
    decision = BaggingClassifier(max_samples=best_max_samples,
```

```
                                 max_features= clf_decision.best_params_[
'max_features'])
    decision = decision.fit(X_train_val, Y_train_val)
    test_predict = decision.predict(X_test)
    train_predict = decision.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    print("Decision Bagging test accuracy with "+str(best_max_samples)+
          " and "+ str(clf_decision.best_params_['max_features'])+" is:
 " + str(test_acc))
    decision_test_acc.append(test_acc)
    decision_train_acc.append(train_acc)

    i = i+1;
    print("\n\n\n")
print("Random Forest Classifier 80/20 training accuracy:")
print(random_train_acc)
print("Adaboost Classifier 80/20 training accuracy:")
print(adaboost_train_acc)
print("Gradient Boost Classifier 80/20 training accuracy:")
print(gradient_train_acc)
print("Kneigher Bagging Classifier 80/20 training accuracy:")
print(bagging_train_acc)
print("Decision Tree Bagging Classifier 80/20 training accuracy:")
print(decision_train_acc)
print("Random Forest Classifier 80/20 test accuracy:         "+ str(sum(
random_test_acc)/len(random_test_acc)))
print("Adaboost Classifier 80/20 test accuracy:              "+ str(sum(
adaboost_test_acc)/len(adaboost_test_acc)))
print("Gradient Boost Classifier 80/20 test accuracy:        "+ str(sum(
gradient_test_acc)/len(gradient_test_acc)))
print("Kneighbor Bagging Classifier 80/20 test accuracy:     "+ str(sum(
bagging_test_acc)/len(bagging_test_acc)))
print("Decision Tree Bagging Classifier 80/20 test accuracy: "+ str(sum(
decision_test_acc)/len(decision_test_acc)))
print("Random Forest Classifier 80/20 test error:            "+ str(1-su
m(random_test_acc)/len(random_test_acc)))
print("Adaboost Classifier 80/20 test error:                 "+ str(1-su
m(adaboost_test_acc)/len(adaboost_test_acc)))
print("Gradient Boost Classifier 80/20 test error:           "+ str(1-su
m(gradient_test_acc)/len(gradient_test_acc)))
print("Kneighbor Bagging Classifier 80/20 test error:        "+ str(1-su
m(bagging_test_acc)/len(bagging_test_acc)))
print("Decision Tree Bagging Classifier 80/20 test error:    "+ str(1-su
m(decision_test_acc)/len(decision_test_acc)))
```

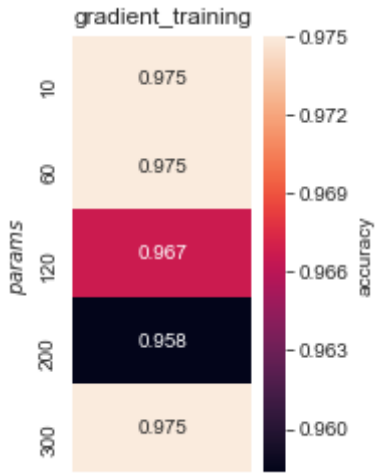
```
1

 RANDOM FOREST CLASSIFIER
[0.95       0.975      0.96666667 0.975      0.975     ]
```



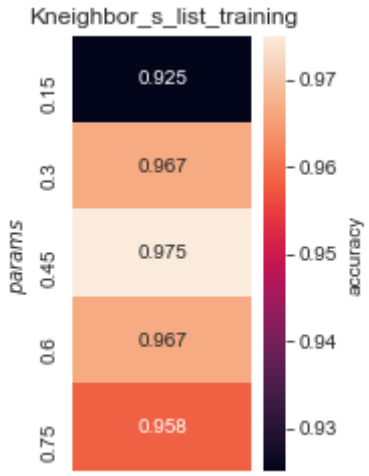

```
random test accuracy with 25 is: 0.9

 ADABOOST CLASSIFIER WITH DECISION TREE
[0.95833333 0.95833333 0.95833333 0.95833333 0.95833333]
```



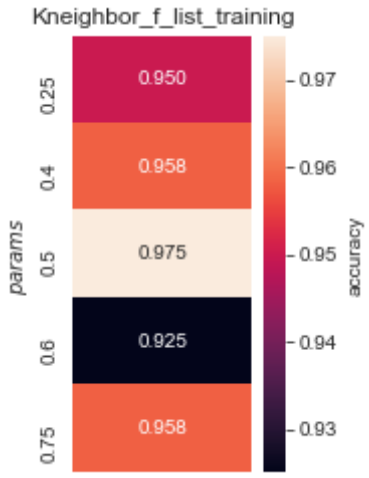

```
Adaboost test accuracy with 10 is: 0.9333333333333333

 GRADIENT BOOSTING CLASSIFIER
[0.975      0.975      0.96666667 0.95833333 0.975     ]
```



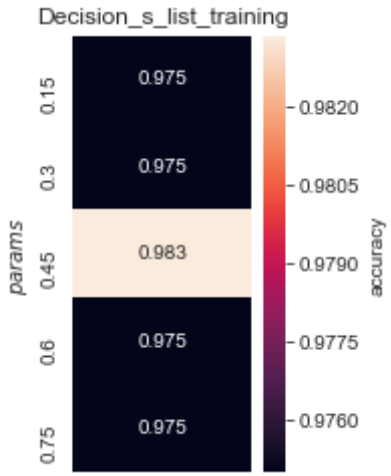

```
Gradient test accuracy with 10 is: 0.9333333333333333

 BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER
max_samples: [0.925      0.96666667 0.975      0.96666667 0.95833333]
```



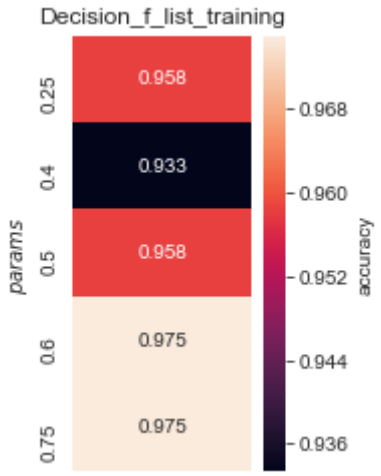

```
max_features: [0.95       0.95833333 0.975      0.925      0.95833333]
```




```
Kneighbor Bagging test accuracy with 0.45 and 0.5 is: 0.9

 BAGGING CLASSIFIER WITH DECISION TREE
max_samples: [0.975      0.975      0.98333333 0.975      0.975      ]
```



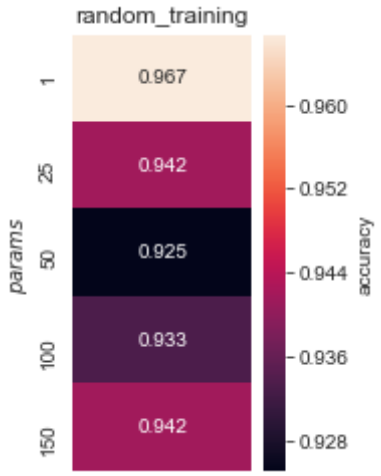

```
max_features: [0.95833333 0.93333333 0.95833333 0.975      0.975      ]
```



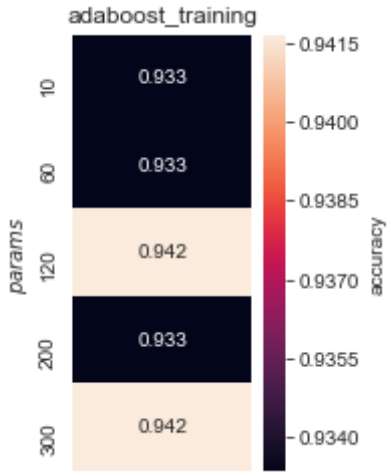

```
Decision Bagging test accuracy with 0.45 and 0.6 is: 0.9
```

```
2
```

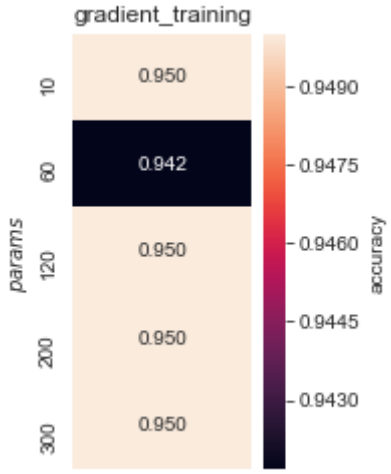
```
 RANDOM FOREST CLASSIFIER
[0.96666667 0.94166667 0.925      0.93333333 0.94166667]
```



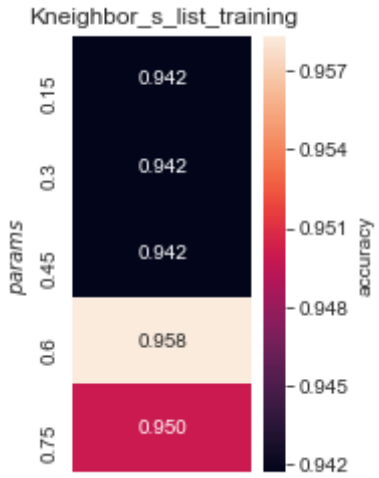
```
random test accuracy with 1 is: 0.9666666666666667

 ADABOOST CLASSIFIER WITH DECISION TREE
[0.93333333 0.93333333 0.94166667 0.93333333 0.94166667]
```



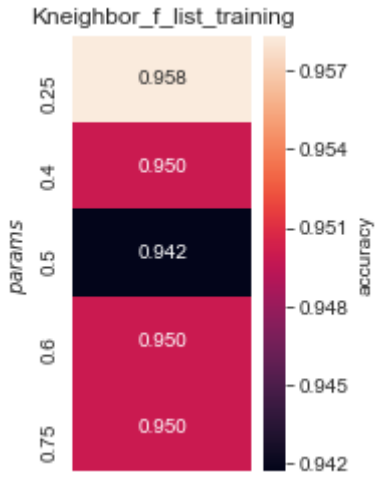

```
Adaboost test accuracy with 120 is: 0.9666666666666667

 GRADIENT BOOSTING CLASSIFIER
[0.95       0.94166667 0.95       0.95       0.95      ]
```



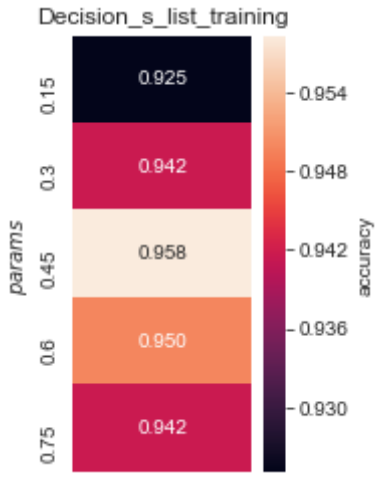

```
Gradient test accuracy with 10 is: 1.0

 BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER
max_samples: [0.94166667 0.94166667 0.94166667 0.95833333 0.95      ]
```



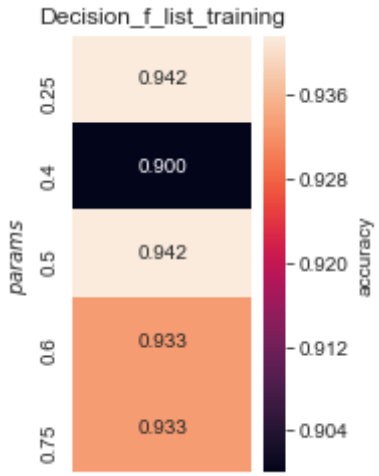

```
max_features: [0.95833333 0.95       0.94166667 0.95       0.95       ]
```




```
Kneighbor Bagging test accuracy with 0.6 and 0.25 is: 0.9666666666666667

 BAGGING CLASSIFIER WITH DECISION TREE
max_samples: [0.925      0.94166667 0.95833333 0.95       0.94166667]
```



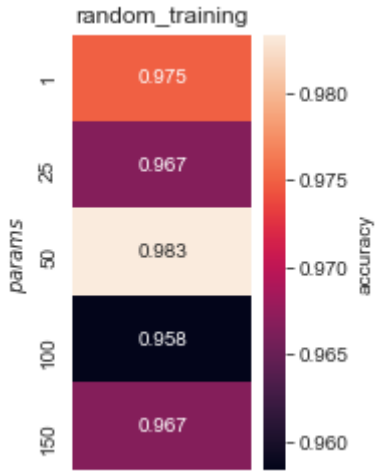

```
max_features: [0.94166667 0.9        0.94166667 0.93333333 0.93333333]
```




```
Decision Bagging test accuracy with 0.45 and 0.25 is: 1.0
```

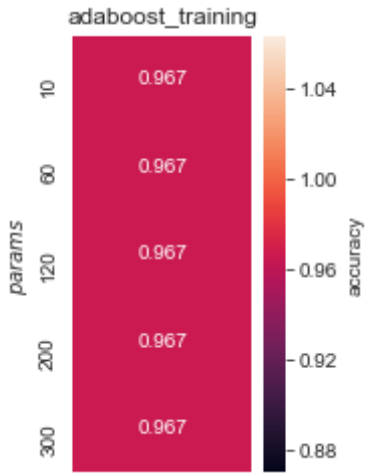
```
3
```

```
 RANDOM FOREST CLASSIFIER
[0.975      0.96666667 0.98333333 0.95833333 0.96666667]
```

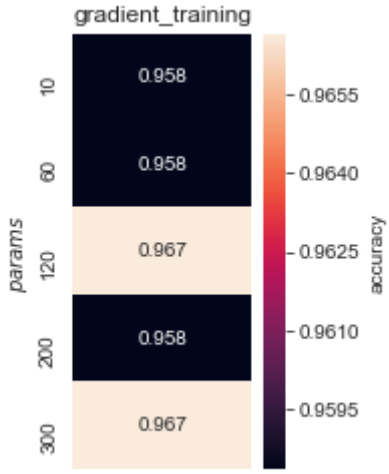



```
random test accuracy with 50 is: 0.9
```

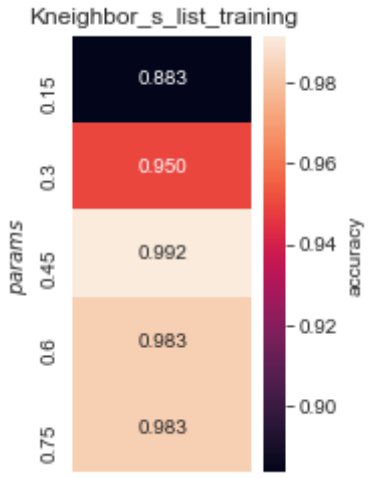
```
 ADABOOST CLASSIFIER WITH DECISION TREE
[0.96666667 0.96666667 0.96666667 0.96666667 0.96666667]
```



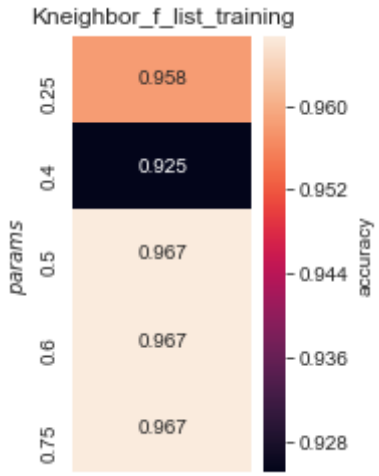

```
Adaboost test accuracy with 10 is: 0.9

 GRADIENT BOOSTING CLASSIFIER
[0.95833333 0.95833333 0.96666667 0.95833333 0.96666667]
```



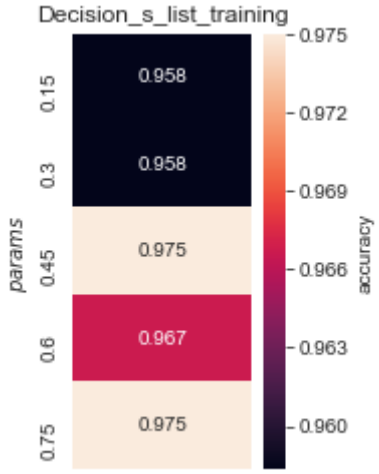

```
Gradient test accuracy with 120 is: 0.8666666666666667

 BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER
max_samples: [0.88333333 0.95       0.99166667 0.98333333 0.98333333]
```



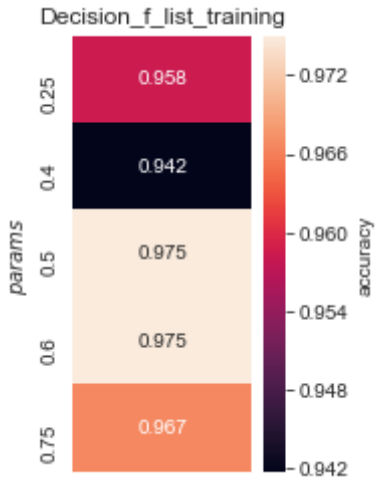

```
max_features: [0.95833333 0.925      0.96666667 0.96666667 0.96666667]
```




```
Kneighbor Bagging test accuracy with 0.45 and 0.5 is: 0.9

 BAGGING CLASSIFIER WITH DECISION TREE
max_samples: [0.95833333 0.95833333 0.975      0.96666667 0.975     ]
```




```
max_features: [0.95833333 0.94166667 0.975      0.975      0.96666667]
```



```
Decision Bagging test accuracy with 0.45 and 0.5 is: 0.9
```

```
Random Forest Classifier 80/20 training accuracy:
[1.0, 0.9666666666666667, 1.0]
Adaboost Classifier 80/20 training accuracy:
[1.0, 1.0, 1.0]
Gradient Boost Classifier 80/20 training accuracy:
[1.0, 0.9916666666666667, 1.0]
Kneigher Bagging Classifier 80/20 training accuracy:
[0.9833333333333333, 0.9416666666666667, 0.9833333333333333]
Decision Tree Bagging Classifier 80/20 training accuracy:
[0.9916666666666667, 0.9083333333333333, 0.9833333333333333]
Random Forest Classifier 80/20 test accuracy:         0.9222222222222
2
Adaboost Classifier 80/20 test accuracy:              0.9333333333333
2
Gradient Boost Classifier 80/20 test accuracy:        0.9333333333333
2
Kneighbor Bagging Classifier 80/20 test accuracy:     0.9222222222222
2
Decision Tree Bagging Classifier 80/20 test accuracy: 0.9333333333333
2
Random Forest Classifier 80/20 test error:            0.0777777777777
83
Adaboost Classifier 80/20 test error:                 0.0666666666666
76
Gradient Boost Classifier 80/20 test error:           0.0666666666666
76
Kneighbor Bagging Classifier 80/20 test error:        0.0777777777777
83
Decision Tree Bagging Classifier 80/20 test error:    0.0666666666666
76
```

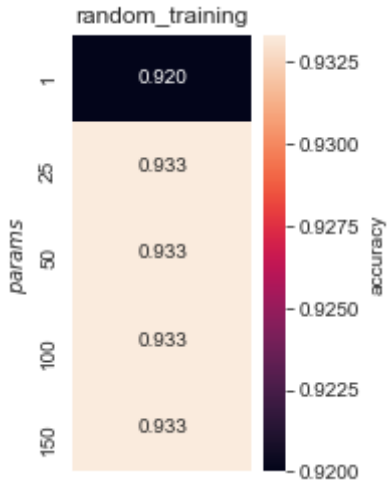
In [13]:
```
random_train_acc = []
adaboost_train_acc = []
gradient_train_acc = []
bagging_train_acc = []
decision_train_acc = []
random_test_acc = []
adaboost_test_acc = []
gradient_test_acc = []
bagging_test_acc = []
decision_test_acc = []
i = 1;
while i<4:
    #here we split the data again
    np.random.shuffle(X_and_Y)
    X = X_and_Y[:, 0:-1]
    Y = X_and_Y[:, -1]
    X_train_val = X[:int(0.5*len(X))]
    X_test = X[int(0.5*len(X)):]
    Y_train_val = Y[:int(0.5*len(Y))]
    Y_test = Y[int(0.5*len(Y)):]
    print(i)
    #then we run it on random forest
    random = RandomForestClassifier()
    print("\n RANDOM FOREST CLASSIFIER")
    #first grid search / cross validation for best parameter
    n_list = [1,25,50,100,150]
    parameters_random = {'n_estimators' : n_list}
    clf_random = GridSearchCV(random, parameters_random, cv=3)
    clf_random.fit(X_train_val, Y_train_val)
    #display result with heatmap
    print(clf_random.cv_results_['mean_test_score'])
    draw_heatmap(clf_random.cv_results_['mean_test_score'].reshape(len(n
_list),1), "random_training", n_list)
    #choose the best parameter and train again
    random = RandomForestClassifier(n_estimators = clf_random.best_param
s_['n_estimators'])
    random = random.fit(X_train_val, Y_train_val)
    #calculate the test error
    test_predict = random.predict(X_test)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    train_predict = random.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    print("random test accuracy with "+str(clf_random.best_params_['n_es
timators'])+" is: " + str(test_acc))
    random_test_acc.append(test_acc)
    random_train_acc.append(train_acc)

    #run it with Adaboost
    dt = DecisionTreeClassifier()
    adaboost = AdaBoostClassifier(base_estimator=dt)
    print("\n ADABOOST CLASSIFIER WITH DECISION TREE")
    #first grid search / cross validation for best parameter
    e_list = [10,60,120,200,300]
    parameters_adaboost = {'n_estimators' : e_list}
```

```
    clf_adaboost = GridSearchCV(adaboost, parameters_adaboost, cv=3)
    clf_adaboost.fit(X_train_val,Y_train_val)
    #display result with heatmap
    print(clf_adaboost.cv_results_['mean_test_score'])
    draw_heatmap(clf_adaboost.cv_results_['mean_test_score'].reshape(len
(e_list),1), "adaboost_training", e_list)
    #choose the best parameter and train again
    adaboost = AdaBoostClassifier(base_estimator=dt, n_estimators=clf_ad
aboost.best_params_['n_estimators'])
    adaboost = adaboost.fit(X_train_val, Y_train_val)
    #calculate the test error
    test_predict = adaboost.predict(X_test)
    train_predict = adaboost.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    print("Adaboost test accuracy with "+str(clf_adaboost.best_params_[
'n_estimators'])+" is: " + str(test_acc))
    adaboost_test_acc.append(test_acc)
    adaboost_train_acc.append(train_acc)

    #run it with gradient
    gradient = ensemble.GradientBoostingClassifier()
    print("\n GRADIENT BOOSTING CLASSIFIER")
    parameters_gradient = {'n_estimators': e_list}
    #first grid search / cross validation for best parameter
    clf_gradient = GridSearchCV(gradient, parameters_gradient, cv=3)
    clf_gradient.fit(X_train_val, Y_train_val)
    #display result with heatmap
    print(clf_gradient.cv_results_['mean_test_score'])
    draw_heatmap(clf_gradient.cv_results_['mean_test_score'].reshape(len
(e_list),1), "gradient_training", e_list)
    #choose the best parameter and train again
    gradient = ensemble.GradientBoostingClassifier(n_estimators=clf_grad
ient.best_params_['n_estimators'])
    gradient = gradient.fit(X_train_val,Y_train_val)
    #calculate the test error
    test_predict = gradient.predict(X_test)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    train_predict = gradient.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    print("Gradient test accuracy with "+str(clf_gradient.best_params_[
'n_estimators'])+" is: " + str(test_acc))
    gradient_test_acc.append(test_acc)
    gradient_train_acc.append(train_acc)

    #run it with Bagging Family with KNeighborClassifer
    bagging = BaggingClassifier(KNeighborsClassifier())
    print("\n BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER")
    s_list = [0.15,0.3,0.45,0.6,0.75]
    parameters_bagging = {'max_samples' : s_list}
    clf_bagging = GridSearchCV(bagging, parameters_bagging, cv=3)
    clf_bagging.fit(X_train_val, Y_train_val)
    print("max_samples: " + str(clf_bagging.cv_results_['mean_test_scor
```

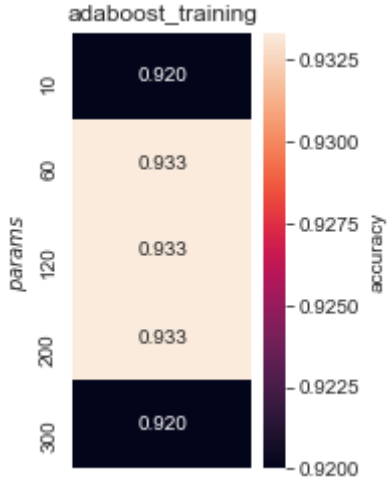
```
e']))
    draw_heatmap(clf_bagging.cv_results_['mean_test_score'].reshape(len(
s_list),1),"Kneighbor_s_list_training", s_list)
    #choose the best parameter and train again for max_features
    bagging = BaggingClassifier(KNeighborsClassifier(), max_samples=clf_
bagging.best_params_['max_samples'])
    best_max_samples = clf_bagging.best_params_['max_samples']
    f_list = [0.25,0.4,0.5,0.6,0.75]
    parameters_bagging = {'max_features': f_list}
    clf_bagging = GridSearchCV(bagging, parameters_bagging, cv=3)
    clf_bagging.fit(X_train_val, Y_train_val)
    print("max_features: " + str(clf_bagging.cv_results_['mean_test_scor
e']))
    draw_heatmap(clf_bagging.cv_results_['mean_test_score'].reshape(len(
f_list),1),"Kneighbor_f_list_training", f_list)
    #choose the best parameter and train again for max_features
    bagging = BaggingClassifier(KNeighborsClassifier(), max_samples=best
_max_samples,
                                max_features= clf_bagging.best_params_[
'max_features'])
    bagging = bagging.fit(X_train_val, Y_train_val)
    test_predict = bagging.predict(X_test)
    train_predict = bagging.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    print("Kneighbor Bagging test accuracy with "+str(best_max_samples)+

          " and "+ str(clf_bagging.best_params_['max_features'])+" is: "
 + str(test_acc))
    bagging_test_acc.append(test_acc)
    bagging_train_acc.append(train_acc)

    #run it with Bagging Family with Decision tree
    decision = BaggingClassifier()
    parameters_decision = {'max_samples' : s_list}
    print("\n BAGGING CLASSIFIER WITH DECISION TREE")
    clf_decision = GridSearchCV(decision, parameters_decision, cv=3)
    clf_decision.fit(X_train_val, Y_train_val)
    print("max_samples: " + str(clf_decision.cv_results_['mean_test_scor
e']))
    draw_heatmap(clf_decision.cv_results_['mean_test_score'].reshape(len
(s_list),1),"Decision_s_list_training", s_list)
    #choose the best parameter and train again for max_features
    decision = BaggingClassifier(max_samples=clf_decision.best_params_[
'max_samples'])
    best_max_samples = clf_decision.best_params_['max_samples']
    parameters_decision = {'max_features': f_list}
    clf_decision = GridSearchCV(decision, parameters_decision, cv=3)
    clf_decision.fit(X_train_val, Y_train_val)
    print("max_features: " + str(clf_decision.cv_results_['mean_test_sco
re']))
    draw_heatmap(clf_decision.cv_results_['mean_test_score'].reshape(len
(f_list),1),"Decision_f_list_training", f_list)
    #choose the best parameter and train again for max_features
    decision = BaggingClassifier(max_samples=best_max_samples,
```

```
                                 max_features= clf_decision.best_params_[
'max_features'])
    decision = decision.fit(X_train_val, Y_train_val)
    test_predict = decision.predict(X_test)
    train_predict = decision.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    print("Decision Bagging test accuracy with "+str(best_max_samples)+
          " and "+ str(clf_decision.best_params_['max_features'])+" is:
 " + str(test_acc))
    decision_test_acc.append(test_acc)
    decision_train_acc.append(train_acc)

    i = i+1;
    print("\n\n\n")
print("Random Forest Classifier 50/50 training accuracy:")
print(random_train_acc)
print("Adaboost Classifier 50/50 training accuracy:")
print(adaboost_train_acc)
print("Gradient Boost Classifier 50/50 training accuracy:")
print(gradient_train_acc)
print("Kneigher Bagging Classifier 50/50 training accuracy:")
print(bagging_train_acc)
print("Decision Tree Bagging Classifier 50/50 training accuracy:")
print(decision_train_acc)
print("Random Forest Classifier 50/50 test accuracy:         "+ str(sum(
random_test_acc)/len(random_test_acc)))
print("Adaboost Classifier 50/50 test accuracy:              "+ str(sum(
adaboost_test_acc)/len(adaboost_test_acc)))
print("Gradient Boost Classifier 50/50 test accuracy:        "+ str(sum(
gradient_test_acc)/len(gradient_test_acc)))
print("Kneighbor Bagging Classifier 50/50 test accuracy:     "+ str(sum(
bagging_test_acc)/len(bagging_test_acc)))
print("Decision Tree Bagging Classifier 50/50 test accuracy: "+ str(sum(
decision_test_acc)/len(decision_test_acc)))
print("Random Forest Classifier 50/50 test error:            "+ str(1-su
m(random_test_acc)/len(random_test_acc)))
print("Adaboost Classifier 50/50 test error:                 "+ str(1-su
m(adaboost_test_acc)/len(adaboost_test_acc)))
print("Gradient Boost Classifier 50/50 test error:           "+ str(1-su
m(gradient_test_acc)/len(gradient_test_acc)))
print("Kneighbor Bagging Classifier 50/50 test error:        "+ str(1-su
m(bagging_test_acc)/len(bagging_test_acc)))
print("Decision Tree Bagging Classifier 50/50 test error:    "+ str(1-su
m(decision_test_acc)/len(decision_test_acc)))
```

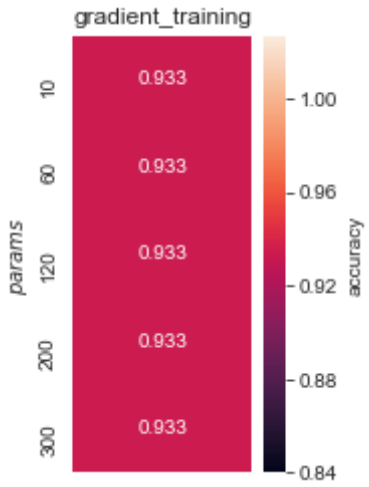
```
1

 RANDOM FOREST CLASSIFIER
[0.92       0.93333333 0.93333333 0.93333333 0.93333333]
```




```
random test accuracy with 25 is: 0.96

 ADABOOST CLASSIFIER WITH DECISION TREE
[0.92       0.93333333 0.93333333 0.93333333 0.92      ]
```

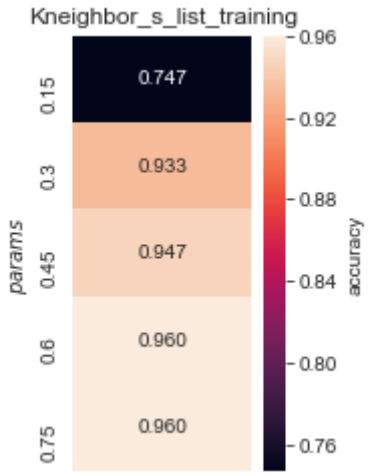



```
Adaboost test accuracy with 60 is: 0.92

 GRADIENT BOOSTING CLASSIFIER
[0.93333333 0.93333333 0.93333333 0.93333333 0.93333333]
```

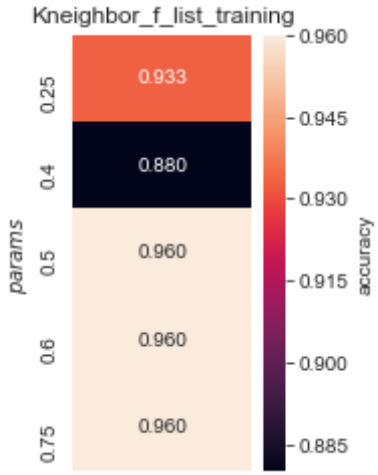



```
Gradient test accuracy with 10 is: 0.92
```

```
 BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER
max_samples: [0.74666667 0.93333333 0.94666667 0.96       0.96       ]
```

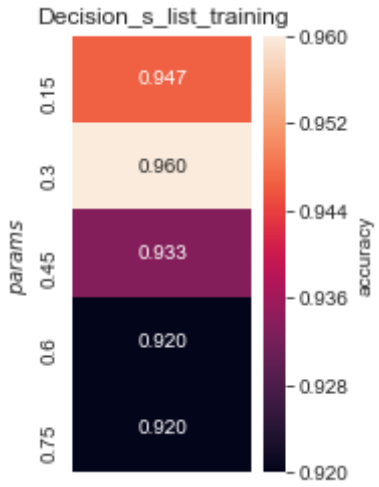



```
max_features: [0.93333333 0.88       0.96       0.96       0.96       ]
```

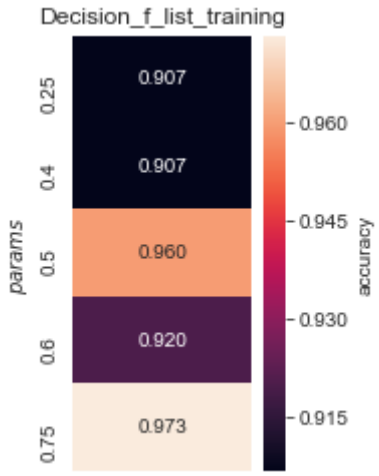



```
Kneighbor Bagging test accuracy with 0.6 and 0.5 is: 0.9466666666666667
```

```
 BAGGING CLASSIFIER WITH DECISION TREE
max_samples: [0.94666667 0.96       0.93333333 0.92       0.92       ]
```



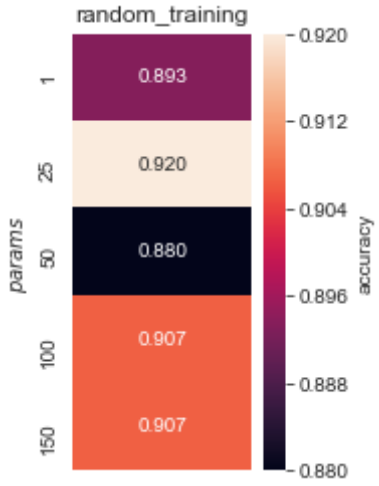

```
max_features: [0.90666667 0.90666667 0.96       0.92       0.97333333]
```



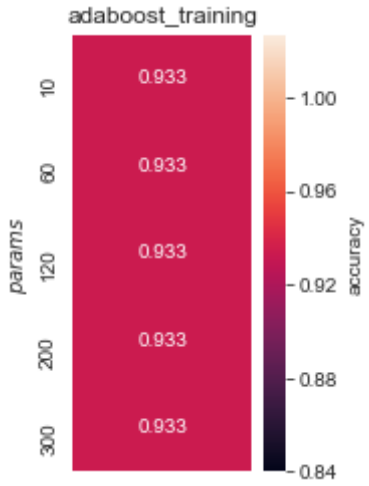

```
Decision Bagging test accuracy with 0.3 and 0.75 is: 0.9466666666666667
```

```
2
```

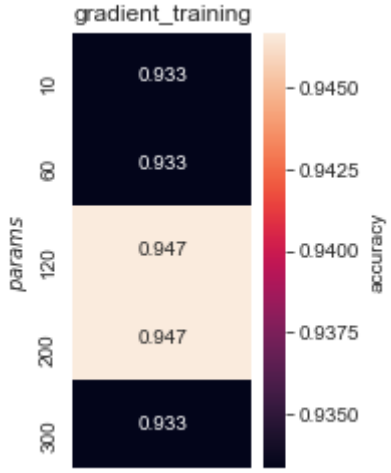
```
 RANDOM FOREST CLASSIFIER
[0.89333333 0.92       0.88       0.90666667 0.90666667]
```



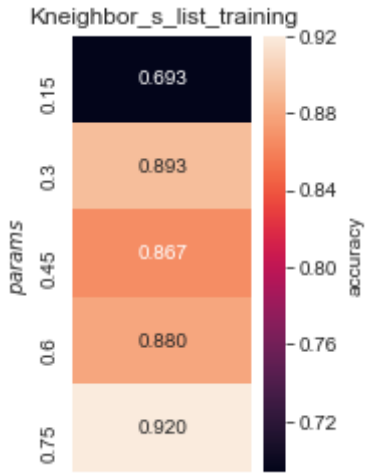
```
random test accuracy with 25 is: 0.9733333333333334

 ADABOOST CLASSIFIER WITH DECISION TREE
[0.93333333 0.93333333 0.93333333 0.93333333 0.93333333]
```



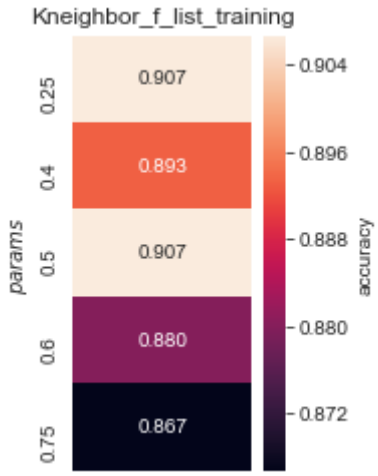

```
Adaboost test accuracy with 10 is: 0.96

 GRADIENT BOOSTING CLASSIFIER
[0.93333333 0.93333333 0.94666667 0.94666667 0.93333333]
```



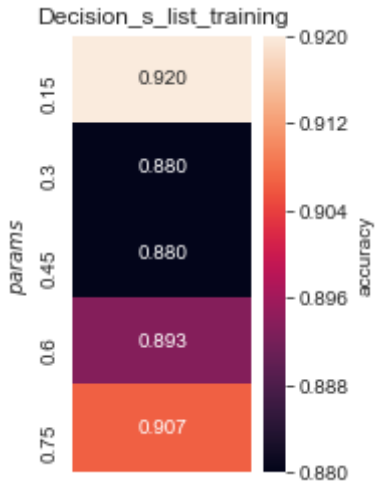

```
Gradient test accuracy with 120 is: 0.96

 BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER
max_samples: [0.69333333 0.89333333 0.86666667 0.88       0.92       ]
```



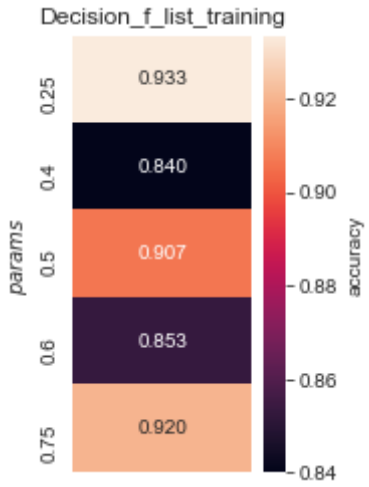

```
max_features: [0.90666667 0.89333333 0.90666667 0.88       0.86666667]
```




```
Kneighbor Bagging test accuracy with 0.75 and 0.25 is: 0.9733333333333
34

 BAGGING CLASSIFIER WITH DECISION TREE
max_samples: [0.92       0.88       0.88       0.89333333 0.90666667]
```



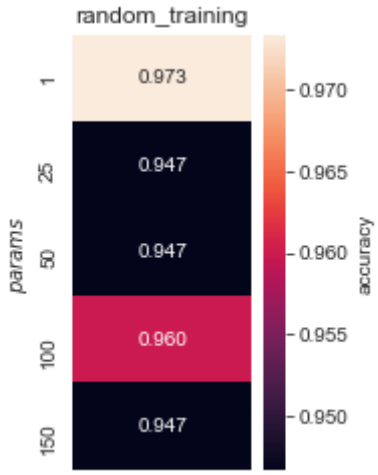

```
max_features: [0.93333333 0.84       0.90666667 0.85333333 0.92       ]
```




```
Decision Bagging test accuracy with 0.15 and 0.25 is: 0.893333333333333
3
```

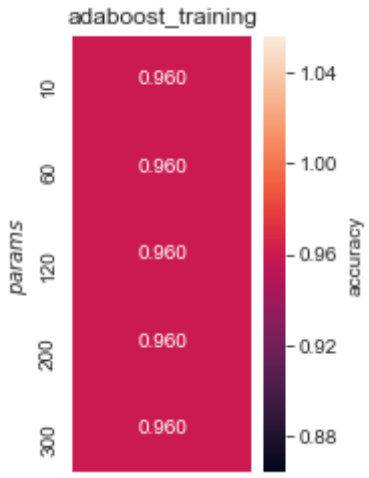
```
3
```

```
 RANDOM FOREST CLASSIFIER
[0.97333333 0.94666667 0.94666667 0.96       0.94666667]
```

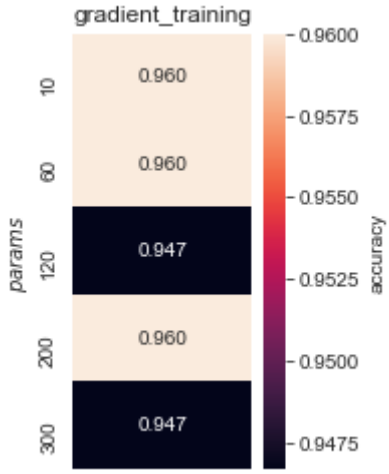



```
random test accuracy with 1 is: 0.96
```

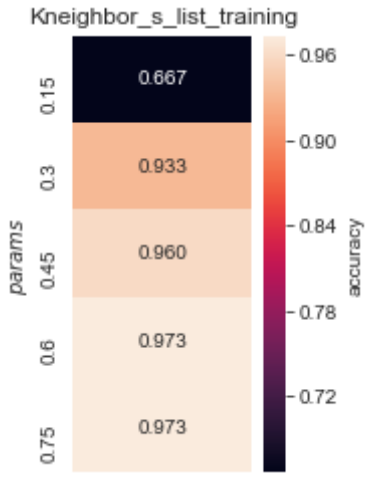
```
 ADABOOST CLASSIFIER WITH DECISION TREE
[0.96 0.96 0.96 0.96 0.96]
```



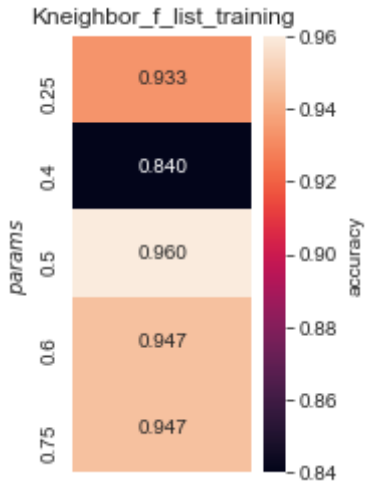

```
Adaboost test accuracy with 10 is: 0.96

 GRADIENT BOOSTING CLASSIFIER
[0.96       0.96       0.94666667 0.96       0.94666667]
```



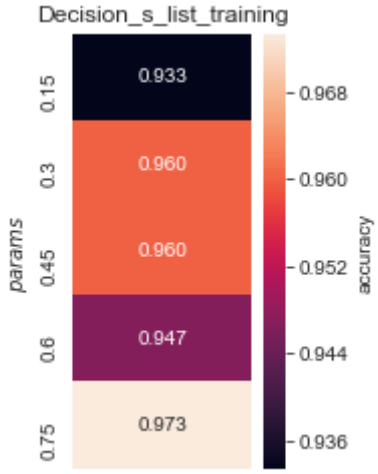

```
Gradient test accuracy with 10 is: 0.9466666666666667

 BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER
max_samples: [0.66666667 0.93333333 0.96       0.97333333 0.97333333]
```



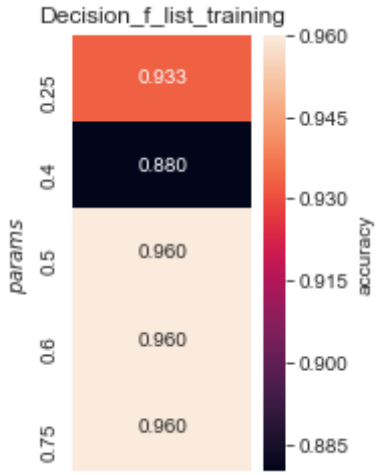

```
max_features: [0.93333333 0.84       0.96       0.94666667 0.94666667]
```




```
Kneighbor Bagging test accuracy with 0.6 and 0.5 is: 0.9466666666666667

 BAGGING CLASSIFIER WITH DECISION TREE
max_samples: [0.93333333 0.96       0.96       0.94666667 0.97333333]
```




```
max_features: [0.93333333 0.88       0.96       0.96       0.96       ]
```



```
Decision Bagging test accuracy with 0.75 and 0.5 is: 0.9466666666666667
```

```
Random Forest Classifier 50/50 training accuracy:
[1.0, 1.0, 0.9466666666666667]
Adaboost Classifier 50/50 training accuracy:
[1.0, 1.0, 1.0]
Gradient Boost Classifier 50/50 training accuracy:
[1.0, 1.0, 1.0]
Kneigher Bagging Classifier 50/50 training accuracy:
[0.9733333333333334, 0.9466666666666667, 1.0]
Decision Tree Bagging Classifier 50/50 training accuracy:
[0.9866666666666667, 0.92, 1.0]
Random Forest Classifier 50/50 test accuracy:         0.964444444444444
5
Adaboost Classifier 50/50 test accuracy:              0.946666666666666
7
Gradient Boost Classifier 50/50 test accuracy:        0.942222222222222
2
Kneighbor Bagging Classifier 50/50 test accuracy:     0.955555555555555
6
Decision Tree Bagging Classifier 50/50 test accuracy: 0.928888888888888
9
Random Forest Classifier 50/50 test error:            0.035555555555555
45
Adaboost Classifier 50/50 test error:                 0.053333333333333
344
Gradient Boost Classifier 50/50 test error:           0.057777777777777
82
Kneighbor Bagging Classifier 50/50 test error:        0.044444444444444
4
Decision Tree Bagging Classifier 50/50 test error:    0.071111111111111
12
```

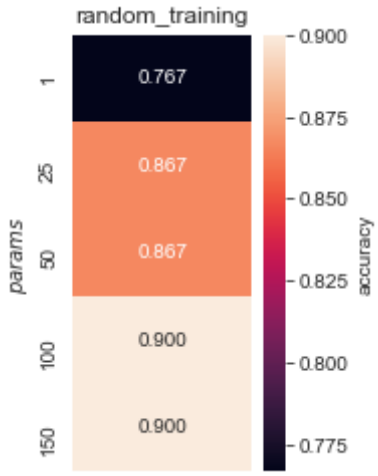
In [16]:
```
random_train_acc = []
adaboost_train_acc = []
gradient_train_acc = []
bagging_train_acc = []
decision_train_acc = []
random_test_acc = []
adaboost_test_acc = []
gradient_test_acc = []
bagging_test_acc = []
decision_test_acc = []
i = 1;
while i<4:
    #here we split the data again
    np.random.shuffle(X_and_Y)
    X = X_and_Y[:, 0:-1]
    Y = X_and_Y[:, -1]
    X_train_val = X[:int(0.2*len(X))]
    X_test = X[int(0.2*len(X)):]
    Y_train_val = Y[:int(0.2*len(Y))]
    Y_test = Y[int(0.2*len(Y)):]
    print(i)
    #then we run it on random forest
    random = RandomForestClassifier()
    print("\n RANDOM FOREST CLASSIFIER")
    #first grid search / cross validation for best parameter
    n_list = [1,25,50,100,150]
    parameters_random = {'n_estimators' : n_list}
    clf_random = GridSearchCV(random, parameters_random, cv=3)
    clf_random.fit(X_train_val, Y_train_val)
    #display result with heatmap
    print(clf_random.cv_results_['mean_test_score'])
    draw_heatmap(clf_random.cv_results_['mean_test_score'].reshape(len(n
_list),1), "random_training", n_list)
    #choose the best parameter and train again
    random = RandomForestClassifier(n_estimators = clf_random.best_param
s_['n_estimators'])
    random = random.fit(X_train_val, Y_train_val)
    #calculate the test error
    test_predict = random.predict(X_test)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    train_predict = random.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    print("random test accuracy with "+str(clf_random.best_params_['n_es
timators'])+" is: " + str(test_acc))
    random_test_acc.append(test_acc)
    random_train_acc.append(train_acc)

    #run it with Adaboost
    dt = DecisionTreeClassifier()
    adaboost = AdaBoostClassifier(base_estimator=dt)
    print("\n ADABOOST CLASSIFIER WITH DECISION TREE")
    #first grid search / cross validation for best parameter
    e_list = [10,60,120,200,300]
    parameters_adaboost = {'n_estimators' : e_list}
```

```
    clf_adaboost = GridSearchCV(adaboost, parameters_adaboost, cv=3)
    clf_adaboost.fit(X_train_val,Y_train_val)
    #display result with heatmap
    print(clf_adaboost.cv_results_['mean_test_score'])
    draw_heatmap(clf_adaboost.cv_results_['mean_test_score'].reshape(len
(e_list),1), "adaboost_training", e_list)
    #choose the best parameter and train again
    adaboost = AdaBoostClassifier(base_estimator=dt, n_estimators=clf_ad
aboost.best_params_['n_estimators'])
    adaboost = adaboost.fit(X_train_val, Y_train_val)
    #calculate the test error
    test_predict = adaboost.predict(X_test)
    train_predict = adaboost.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    print("Adaboost test accuracy with "+str(clf_adaboost.best_params_[
'n_estimators'])+" is: " + str(test_acc))
    adaboost_test_acc.append(test_acc)
    adaboost_train_acc.append(train_acc)

    #run it with gradient
    gradient = ensemble.GradientBoostingClassifier()
    print("\n GRADIENT BOOSTING CLASSIFIER")
    parameters_gradient = {'n_estimators': e_list}
    #first grid search / cross validation for best parameter
    clf_gradient = GridSearchCV(gradient, parameters_gradient, cv=3)
    clf_gradient.fit(X_train_val, Y_train_val)
    #display result with heatmap
    print(clf_gradient.cv_results_['mean_test_score'])
    draw_heatmap(clf_gradient.cv_results_['mean_test_score'].reshape(len
(e_list),1), "gradient_training", e_list)
    #choose the best parameter and train again
    gradient = ensemble.GradientBoostingClassifier(n_estimators=clf_grad
ient.best_params_['n_estimators'])
    gradient = gradient.fit(X_train_val,Y_train_val)
    #calculate the test error
    test_predict = gradient.predict(X_test)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    train_predict = gradient.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    print("Gradient test accuracy with "+str(clf_gradient.best_params_[
'n_estimators'])+" is: " + str(test_acc))
    gradient_test_acc.append(test_acc)
    gradient_train_acc.append(train_acc)

    #run it with Bagging Family with KNeighborClassifer
    bagging = BaggingClassifier(KNeighborsClassifier())
    print("\n BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER")
    s_list = [0.3,0.4,0.5,0.6,0.7]
    parameters_bagging = {'max_samples' : s_list}
    clf_bagging = GridSearchCV(bagging, parameters_bagging, cv=3)
    clf_bagging.fit(X_train_val, Y_train_val)
    print("max_samples: " + str(clf_bagging.cv_results_['mean_test_scor
```

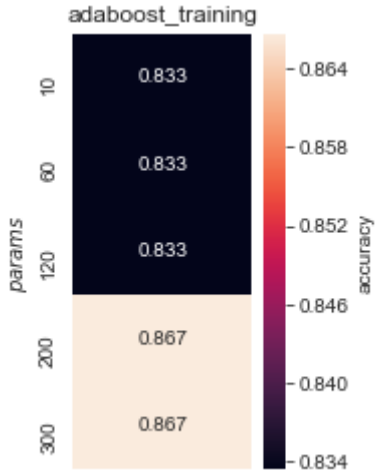
```
e']))
    draw_heatmap(clf_bagging.cv_results_['mean_test_score'].reshape(len(
s_list),1),"Kneighbor_s_list_training", s_list)
    #choose the best parameter and train again for max_features
    bagging = BaggingClassifier(KNeighborsClassifier(), max_samples=clf_
bagging.best_params_['max_samples'])
    best_max_samples = clf_bagging.best_params_['max_samples']
    f_list = [0.3,0.4,0.5,0.6,0.7]
    parameters_bagging = {'max_features': f_list}
    clf_bagging = GridSearchCV(bagging, parameters_bagging, cv=3)
    clf_bagging.fit(X_train_val, Y_train_val)
    print("max_features: " + str(clf_bagging.cv_results_['mean_test_scor
e']))
    draw_heatmap(clf_bagging.cv_results_['mean_test_score'].reshape(len(
f_list),1),"Kneighbor_f_list_training", f_list)
    #choose the best parameter and train again for max_features
    bagging = BaggingClassifier(KNeighborsClassifier(), max_samples=best
_max_samples,
                                max_features= clf_bagging.best_params_[
'max_features'])
    bagging = bagging.fit(X_train_val, Y_train_val)
    test_predict = bagging.predict(X_test)
    train_predict = bagging.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    print("Kneighbor Bagging test accuracy with "+str(best_max_samples)+

          " and "+ str(clf_bagging.best_params_['max_features'])+" is: "
 + str(test_acc))
    bagging_test_acc.append(test_acc)
    bagging_train_acc.append(train_acc)

    #run it with Bagging Family with Decision tree
    decision = BaggingClassifier()
    parameters_decision = {'max_samples' : s_list}
    print("\n BAGGING CLASSIFIER WITH DECISION TREE")
    clf_decision = GridSearchCV(decision, parameters_decision, cv=3)
    clf_decision.fit(X_train_val, Y_train_val)
    print("max_samples: " + str(clf_decision.cv_results_['mean_test_scor
e']))
    draw_heatmap(clf_decision.cv_results_['mean_test_score'].reshape(len
(s_list),1),"Decision_s_list_training", s_list)
    #choose the best parameter and train again for max_features
    decision = BaggingClassifier(max_samples=clf_decision.best_params_[
'max_samples'])
    best_max_samples = clf_decision.best_params_['max_samples']
    parameters_decision = {'max_features': f_list}
    clf_decision = GridSearchCV(decision, parameters_decision, cv=3)
    clf_decision.fit(X_train_val, Y_train_val)
    print("max_features: " + str(clf_decision.cv_results_['mean_test_sco
re']))
    draw_heatmap(clf_decision.cv_results_['mean_test_score'].reshape(len
(f_list),1),"Decision_f_list_training", f_list)
    #choose the best parameter and train again for max_features
    decision = BaggingClassifier(max_samples=best_max_samples,
```

```
                                 max_features= clf_decision.best_params_[
'max_features'])
    decision = decision.fit(X_train_val, Y_train_val)
    test_predict = decision.predict(X_test)
    train_predict = decision.predict(X_train_val)
    train_acc = sum([train_predict[i] == Y_train_val[i] for i in range(l
en(Y_train_val))])/len(Y_train_val)
    test_acc = sum([test_predict[i] == Y_test[i] for i in range(len(Y_te
st))])/len(Y_test)
    print("Decision Bagging test accuracy with "+str(best_max_samples)+
          " and "+ str(clf_decision.best_params_['max_features'])+" is:
 " + str(test_acc))
    decision_test_acc.append(test_acc)
    decision_train_acc.append(train_acc)

    i = i+1;
    print("\n\n\n")
print("Random Forest Classifier 20/80 training accuracy:")
print(random_train_acc)
print("Adaboost Classifier 20/80 training accuracy:")
print(adaboost_train_acc)
print("Gradient Boost Classifier 20/80 training accuracy:")
print(gradient_train_acc)
print("Kneigher Bagging Classifier 20/80 training accuracy:")
print(bagging_train_acc)
print("Decision Tree Bagging Classifier 20/80 training accuracy:")
print(decision_train_acc)
print("Random Forest Classifier 20/80 test accuracy:         "+ str(sum(
random_test_acc)/len(random_test_acc)))
print("Adaboost Classifier 20/80 test accuracy:              "+ str(sum(
adaboost_test_acc)/len(adaboost_test_acc)))
print("Gradient Boost Classifier 20/80 test accuracy:        "+ str(sum(
gradient_test_acc)/len(gradient_test_acc)))
print("Kneighbor Bagging Classifier 20/80 test accuracy:     "+ str(sum(
bagging_test_acc)/len(bagging_test_acc)))
print("Decision Tree Bagging Classifier 20/80 test accuracy: "+ str(sum(
decision_test_acc)/len(decision_test_acc)))
print("Random Forest Classifier 20/80 test error:            "+ str(1-su
m(random_test_acc)/len(random_test_acc)))
print("Adaboost Classifier 20/80 test error:                 "+ str(1-su
m(adaboost_test_acc)/len(adaboost_test_acc)))
print("Gradient Boost Classifier 20/80 test error:           "+ str(1-su
m(gradient_test_acc)/len(gradient_test_acc)))
print("Kneighbor Bagging Classifier 20/80 test error:        "+ str(1-su
m(bagging_test_acc)/len(bagging_test_acc)))
print("Decision Tree Bagging Classifier 20/80 test error:    "+ str(1-su
m(decision_test_acc)/len(decision_test_acc)))
```

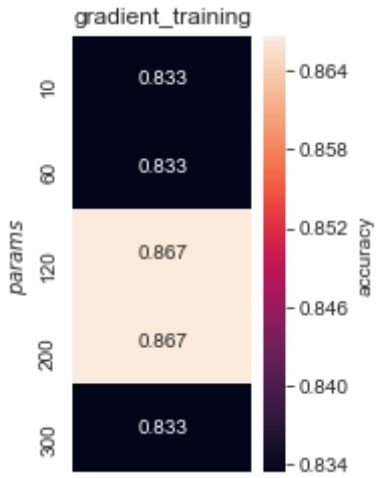
```
1

 RANDOM FOREST CLASSIFIER
[0.76666667 0.86666667 0.86666667 0.9        0.9        ]
```




```
random test accuracy with 100 is: 0.925

 ADABOOST CLASSIFIER WITH DECISION TREE
[0.83333333 0.83333333 0.83333333 0.86666667 0.86666667]
```

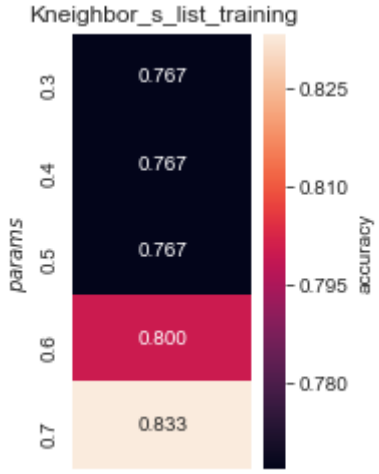



```
Adaboost test accuracy with 200 is: 0.9333333333333333

 GRADIENT BOOSTING CLASSIFIER
[0.83333333 0.83333333 0.86666667 0.86666667 0.83333333]
```

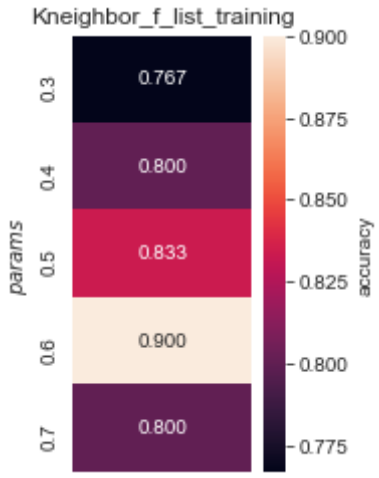



```
Gradient test accuracy with 120 is: 0.925
```

```
 BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER
max_samples: [0.76666667 0.76666667 0.76666667 0.8        0.83333333]
```

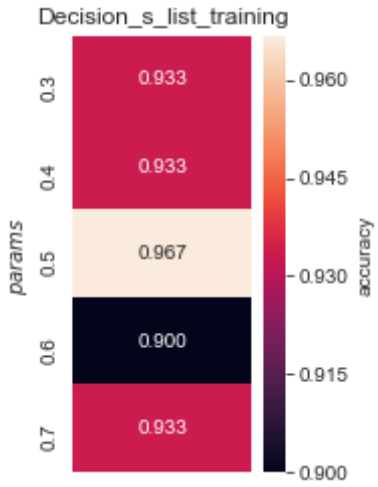



```
max_features: [0.76666667 0.8        0.83333333 0.9        0.8        ]
```

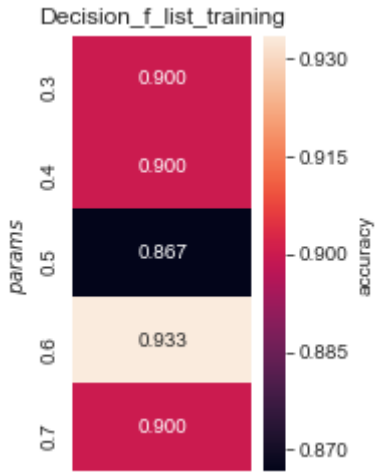



```
Kneighbor Bagging test accuracy with 0.7 and 0.6 is: 0.9083333333333333
```

```
 BAGGING CLASSIFIER WITH DECISION TREE
max_samples: [0.93333333 0.93333333 0.96666667 0.9        0.93333333]
```



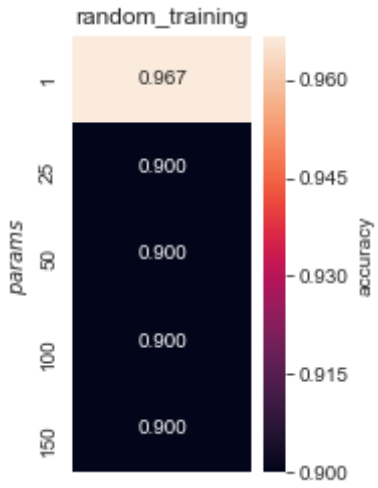

```
max_features: [0.9         0.9         0.86666667 0.93333333 0.9         ]
```



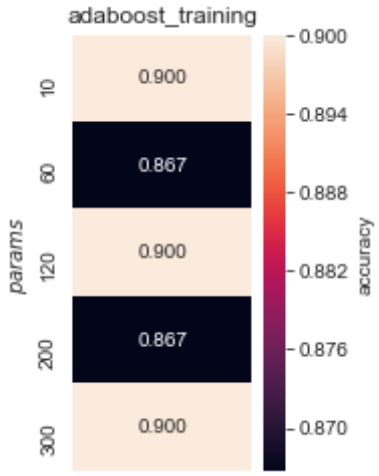

```
Decision Bagging test accuracy with 0.5 and 0.6 is: 0.9333333333333333
```

```
2
```

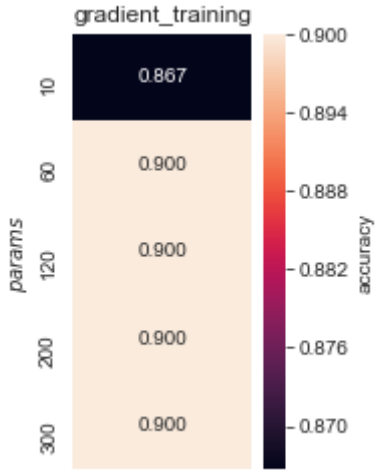
```
 RANDOM FOREST CLASSIFIER
[0.96666667 0.9         0.9         0.9         0.9         ]
```



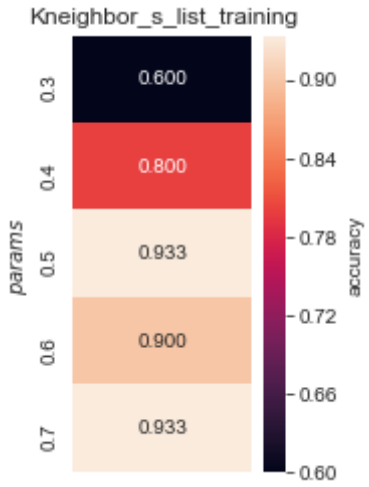
```
random test accuracy with 1 is: 0.8916666666666667

 ADABOOST CLASSIFIER WITH DECISION TREE
[0.9         0.86666667 0.9         0.86666667 0.9         ]
```



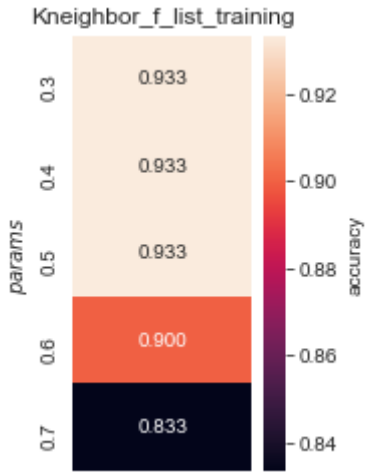

```
Adaboost test accuracy with 10 is: 0.875

 GRADIENT BOOSTING CLASSIFIER
[0.86666667 0.9         0.9         0.9         0.9         ]
```



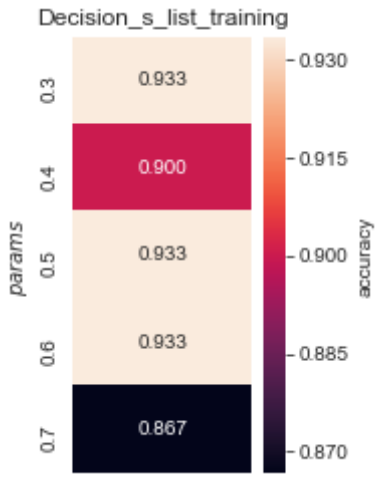

```
Gradient test accuracy with 60 is: 0.95

 BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER
max_samples: [0.6         0.8         0.93333333 0.9         0.93333333]
```



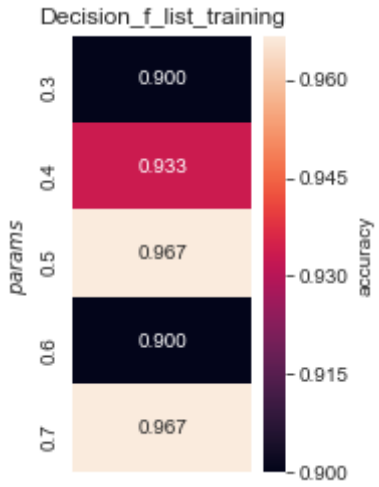

```
max_features: [0.93333333 0.93333333 0.93333333 0.9        0.83333333]
```




```
Kneighbor Bagging test accuracy with 0.5 and 0.3 is: 0.8916666666666667

 BAGGING CLASSIFIER WITH DECISION TREE
max_samples: [0.93333333 0.9        0.93333333 0.93333333 0.86666667]
```



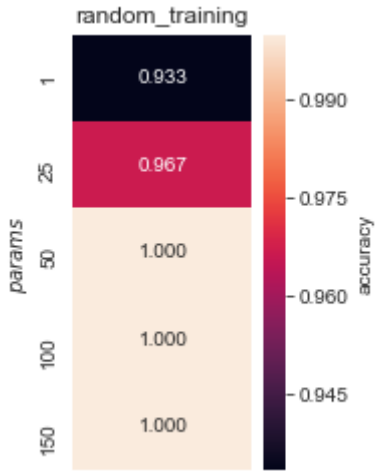

```
max_features: [0.9        0.93333333 0.96666667 0.9        0.96666667]
```




```
Decision Bagging test accuracy with 0.3 and 0.5 is: 0.875
```

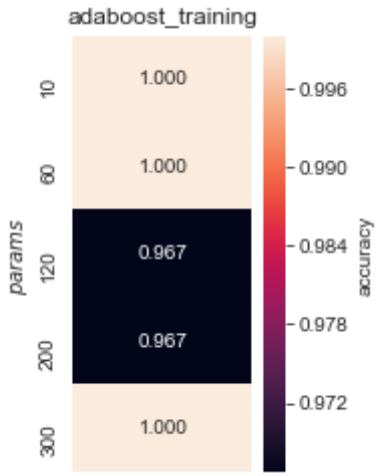
```
3
```

```
 RANDOM FOREST CLASSIFIER
[0.93333333 0.96666667 1.         1.         1.        ]
```

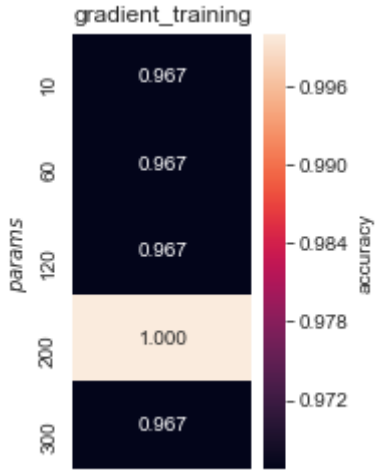



```
random test accuracy with 50 is: 0.925
```

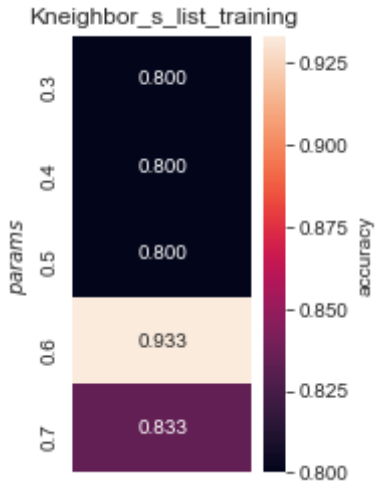
```
 ADABOOST CLASSIFIER WITH DECISION TREE
[1.         1.         0.96666667 0.96666667 1.        ]
```



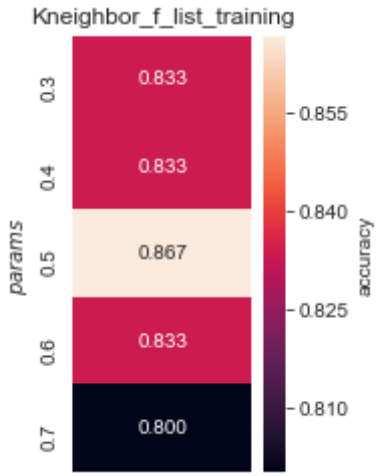

```
Adaboost test accuracy with 10 is: 0.9416666666666667

 GRADIENT BOOSTING CLASSIFIER
[0.96666667 0.96666667 0.96666667 1.         0.96666667]
```



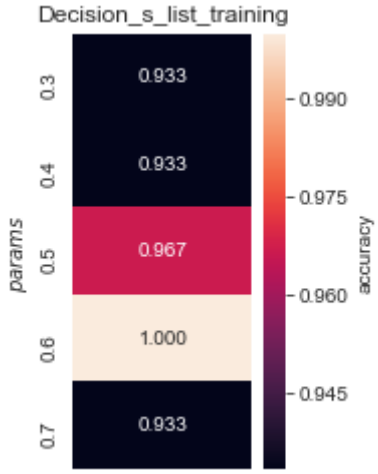

```
Gradient test accuracy with 200 is: 0.9416666666666667

 BAGGING CLASSIFIER WITH KNEIGHBORS CLASSIFIER
max_samples: [0.8        0.8        0.8        0.93333333 0.83333333]
```



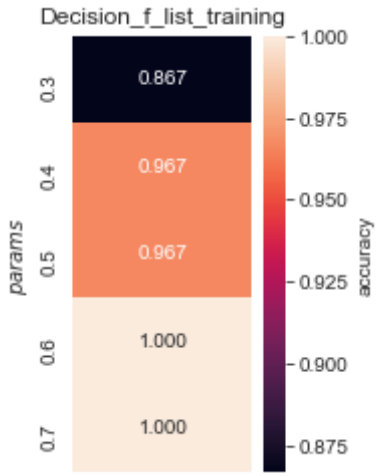

```
max_features: [0.83333333 0.83333333 0.86666667 0.83333333 0.8        ]
```




```
Kneighbor Bagging test accuracy with 0.6 and 0.5 is: 0.825

 BAGGING CLASSIFIER WITH DECISION TREE
max_samples: [0.93333333 0.93333333 0.96666667 1.         0.93333333]
```




```
max_features: [0.86666667 0.96666667 0.96666667 1.         1.        ]
```



```
Decision Bagging test accuracy with 0.6 and 0.6 is: 0.9333333333333333
```

```
Random Forest Classifier 20/80 training accuracy:
[1.0, 1.0, 1.0]
Adaboost Classifier 20/80 training accuracy:
[1.0, 1.0, 1.0]
Gradient Boost Classifier 20/80 training accuracy:
[1.0, 1.0, 1.0]
Kneigher Bagging Classifier 20/80 training accuracy:
[0.9666666666666667, 0.9333333333333333, 0.9333333333333333]
Decision Tree Bagging Classifier 20/80 training accuracy:
[0.9333333333333333, 0.9666666666666667, 1.0]
Random Forest Classifier 20/80 test accuracy:         0.913888888888889
1
Adaboost Classifier 20/80 test accuracy:              0.916666666666666
6
Gradient Boost Classifier 20/80 test accuracy:        0.938888888888888
8
Kneighbor Bagging Classifier 20/80 test accuracy:     0.875
Decision Tree Bagging Classifier 20/80 test accuracy: 0.913888888888888
9
Random Forest Classifier 20/80 test error:            0.086111111111110
92
Adaboost Classifier 20/80 test error:                 0.083333333333333
37
Gradient Boost Classifier 20/80 test error:           0.061111111111111
23
Kneighbor Bagging Classifier 20/80 test error:        0.125
Decision Tree Bagging Classifier 20/80 test error:    0.086111111111111
14
```

In [ ]:

In [ ]: