In [137]:
```
import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
```

Load Data

Mushroom

In [325]:
```
mushroom_data = pd.read_csv('agaricus-lepiota.data', header=None, sep=',')
mushroom_data = mushroom_data.iloc[:5000,:]

mushroom_data = pd.get_dummies(mushroom_data, columns=[1,2,3,4,5,6,7,8,9,10,11
,12,13,14,15,16,17,18,19,20,21,22])

eddible_nums = {0: {'e': 1, 'p': 0}}
mushroom_data.replace(eddible_nums, inplace=True)


print(mushroom_data.shape)
```

```
(5000, 97)
```

In [326]:
```
mushroom_X_train_80 = [None] * 3
mushroom_X_test_20 = [None] * 3
mushroom_Y_train_80 = [None] * 3
mushroom_Y_test_20 = [None] * 3

for i in range(3):
    mushroom_data = mushroom_data.sample(frac=1).reset_index(drop=True)
    mushroom_X = mushroom_data.iloc[:,1:]
    mushroom_Y = mushroom_data.iloc[:,0]

    mushroom_X_train_80[i], mushroom_X_test_20[i], mushroom_Y_train_80[i], mus
hroom_Y_test_20[i] = train_test_split(mushroom_X, mushroom_Y, test_size=0.2, r
andom_state=0)

print(mushroom_X_train_80[0].shape, mushroom_X_test_20[0].shape, mushroom_Y_tr
ain_80[0].shape, mushroom_Y_test_20[0].shape)
```

```
(4000, 96) (1000, 96) (4000,) (1000,)
```

In [327]:
```
mushroom_X_train_50 = [None] * 3
mushroom_X_test_50 = [None] * 3
mushroom_Y_train_50 = [None] * 3
mushroom_Y_test_50 = [None] * 3

for i in range(3):
    mushroom_data = mushroom_data.sample(frac=1).reset_index(drop=True)
    mushroom_X = mushroom_data.iloc[:,1:]
    mushroom_Y = mushroom_data.iloc[:,0]

    mushroom_X_train_50[i], mushroom_X_test_50[i], mushroom_Y_train_50[i], mus
hroom_Y_test_50[i] = train_test_split(mushroom_X, mushroom_Y, test_size=0.5, r
andom_state=0)

print(mushroom_X_train_50[0].shape, mushroom_X_test_50[0].shape, mushroom_Y_tr
ain_50[0].shape, mushroom_Y_test_50[0].shape)
```

```
(2500, 96) (2500, 96) (2500,) (2500,)
```

In [328]:
```
mushroom_X_train_20 = [None] * 3
mushroom_X_test_80 = [None] * 3
mushroom_Y_train_20 = [None] * 3
mushroom_Y_test_80 = [None] * 3

for i in range(3):
    mushroom_data = mushroom_data.sample(frac=1).reset_index(drop=True)
    mushroom_X = mushroom_data.iloc[:,1:]
    mushroom_Y = mushroom_data.iloc[:,0]

    mushroom_X_train_20[i], mushroom_X_test_80[i], mushroom_Y_train_20[i], mus
hroom_Y_test_80[i] = train_test_split(mushroom_X, mushroom_Y, test_size=0.8, r
andom_state=0)

print(mushroom_X_train_20[0].shape, mushroom_X_test_80[0].shape, mushroom_Y_tr
ain_20[0].shape, mushroom_Y_test_80[0].shape)
```

```
(1000, 96) (4000, 96) (1000,) (4000,)
```

Wine

In [280]:
```
def convert_quality(quality):
    if quality > 5:
        return 1.0
    return 0.0
```

In [299]:
```
wine_data = pd.read_csv('winequality-white.csv', sep=';')
wine_data = wine_data.iloc[:5000,:]

wine_data['quality'] = wine_data.apply(lambda row: convert_quality(row['qualit
y']), axis = 1)

print(wine_data.shape)
```

```
(4898, 12)
```

In [300]:
```
wine_X_train_80 = [None] * 3
wine_X_test_20 = [None] * 3
wine_Y_train_80 = [None] * 3
wine_Y_test_20 = [None] * 3

for i in range(3):
    wine_data = wine_data.sample(frac=1).reset_index(drop=True)
    wine_X = wine_data.iloc[:,:-1]
    wine_Y = wine_data.iloc[:,-1]

    wine_X_train_80[i], wine_X_test_20[i], wine_Y_train_80[i], wine_Y_test_20[
i] = train_test_split(wine_X, wine_Y, test_size=0.2, random_state=0)

print(wine_X_train_80[0].shape, wine_X_test_20[0].shape, wine_Y_train_80[0].sh
ape, wine_Y_test_20[0].shape)
```

```
(3918, 11) (980, 11) (3918,) (980,)
```

In [301]:
```
wine_X_train_50 = [None] * 3
wine_X_test_50 = [None] * 3
wine_Y_train_50 = [None] * 3
wine_Y_test_50 = [None] * 3

for i in range(3):
    wine_data = wine_data.sample(frac=1).reset_index(drop=True)
    wine_X = wine_data.iloc[:,:-1]
    wine_Y = wine_data.iloc[:,-1]

    wine_X_train_50[i], wine_X_test_50[i], wine_Y_train_50[i], wine_Y_test_50[
i] = train_test_split(wine_X, wine_Y, test_size=0.5, random_state=0)

print(wine_X_train_50[0].shape, wine_X_test_50[0].shape, wine_Y_train_50[0].sh
ape, wine_Y_test_50[0].shape)
```

```
(2449, 11) (2449, 11) (2449,) (2449,)
```

In [302]:
```
wine_X_train_20 = [None] * 3
wine_X_test_80 = [None] * 3
wine_Y_train_20 = [None] * 3
wine_Y_test_80 = [None] * 3

for i in range(3):
    wine_data = wine_data.sample(frac=1).reset_index(drop=True)
    wine_X = wine_data.iloc[:,:-1]
    wine_Y = wine_data.iloc[:,-1]

    wine_X_train_20[i], wine_X_test_80[i], wine_Y_train_20[i], wine_Y_test_80[
i] = train_test_split(wine_X, wine_Y, test_size=0.8, random_state=0)

print(wine_X_train_20[0].shape, wine_X_test_80[0].shape, wine_Y_train_20[0].sh
ape, wine_Y_test_80[0].shape)
```

```
(979, 11) (3919, 11) (979,) (3919,)
```

Dota

In [11]:
```
dota_data = pd.read_csv('dota2Train.csv', header=None, sep=',')
dota_data = dota_data.iloc[:5000,:]

print(dota_data.shape)
```

```
(5000, 117)
```

In [54]:
```
dota_X_train_80 = [None] * 3
dota_X_test_20 = [None] * 3
dota_Y_train_80 = [None] * 3
dota_Y_test_20 = [None] * 3

for i in range(3):
    dota_data = dota_data.sample(frac=1).reset_index(drop=True)
    dota_X = dota_data.iloc[:,1:]
    dota_Y = dota_data.iloc[:,0]

    dota_X_train_80[i], dota_X_test_20[i], dota_Y_train_80[i], dota_Y_test_20[
i] = train_test_split(dota_X, dota_Y, test_size=0.2, random_state=0)

print(dota_X_train_80[0].shape, dota_X_test_20[0].shape, dota_Y_train_80[0].sh
ape, dota_Y_test_20[0].shape)
```

```
(4000, 116) (1000, 116) (4000,) (1000,)
```

In [53]:
```
dota_X_train_50 = [None] * 3
dota_X_test_50 = [None] * 3
dota_Y_train_50 = [None] * 3
dota_Y_test_50 = [None] * 3

for i in range(3):
    dota_data = dota_data.sample(frac=1).reset_index(drop=True)
    dota_X = dota_data.iloc[:,1:]
    dota_Y = dota_data.iloc[:,0]

    dota_X_train_50[i], dota_X_test_50[i], dota_Y_train_50[i], dota_Y_test_50[
i] = train_test_split(dota_X, dota_Y, test_size=0.5, random_state=0)

print(dota_X_train_50[0].shape, dota_X_test_50[0].shape, dota_Y_train_50[0].sh
ape, dota_Y_test_50[0].shape)
```

```
(2500, 116) (2500, 116) (2500,) (2500,)
```

In [52]:
```
dota_X_train_20 = [None] * 3
dota_X_test_80 = [None] * 3
dota_Y_train_20 = [None] * 3
dota_Y_test_80 = [None] * 3

for i in range(3):
    dota_data = dota_data.sample(frac=1).reset_index(drop=True)
    dota_X = dota_data.iloc[:,1:]
    dota_Y = dota_data.iloc[:,0]

    dota_X_train_20[i], dota_X_test_80[i], dota_Y_train_20[i], dota_Y_test_80[
i] = train_test_split(dota_X, dota_Y, test_size=0.8, random_state=0)

print(dota_X_train_20[0].shape, dota_X_test_80[0].shape, dota_Y_train_20[0].sh
ape, dota_Y_test_80[0].shape)
```

```
(1000, 116) (4000, 116) (1000,) (4000,)
```

Classifiers

In [249]:
```
C_list = [10**-3, 10**-2, 10**-1, 10**0]
G_list = [0.001, 0.005,0.01, 0.05]
```

Linear SVM

Mushroom

80 / 20

In [250]:
```
clfs = []

for i in range(3):
    clf = GridSearchCV(cv=5, estimator=svm.SVC(), param_grid=[{'C': C_list, 'k
ernel': ['linear'], 'gamma': G_list}], n_jobs=-1)
    clf.fit(mushroom_X_train_80[i].values, mushroom_Y_train_80[i].values)
    clfs.append(clf)
    print(i)
```

```
0
1
2
```

In [253]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16,
1))))
    val_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16, 1
))))
    test_accs.append(svm.SVC(C=clfs[i].best_estimator_.C, kernel='linear', gam
ma=clfs[i].best_estimator_.gamma).fit(mushroom_X_train_80[i],mushroom_Y_train_
80[i]).score(mushroom_X_test_20[i], mushroom_Y_test_20[i]))
    print(i)
```

```
0

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

1

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

2
```

In [254]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: [1.]
```

50 / 50

In [19]:
```
clfs = []

for i in range(3):
    clf = GridSearchCV(cv=5, estimator=svm.SVC(), param_grid=[{'C': C_list, 'k
ernel': ['linear'], 'gamma': G_list}], n_jobs=-1)
    clf.fit(mushroom_X_train_50[i].values, mushroom_Y_train_50[i].values)
    clfs.append(clf)
    print(i)
```

```
0
1
2
```

In [20]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16,
1))))
    val_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16, 1
))))
    test_accs.append(svm.SVC(C=clfs[i].best_estimator_.C, kernel='linear', gam
ma=clfs[i].best_estimator_.gamma).fit(mushroom_X_train_50[i],mushroom_Y_train_
50[i]).score(mushroom_X_test_50[i], mushroom_Y_test_50[i]))
    print(i)
```

```
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

0
1

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

2
```

In [21]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: [1.]
```

20 / 80

In [22]:
```
clfs = []

for i in range(3):
    clf = GridSearchCV(cv=5, estimator=svm.SVC(), param_grid=[{'C': C_list, 'k
ernel': ['linear'], 'gamma': G_list}], n_jobs=-1)
    clf.fit(mushroom_X_train_20[i].values, mushroom_Y_train_20[i].values)
    clfs.append(clf)
    print(i)
```

```
0
1
2
```

In [23]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16,
1))))
    val_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16, 1
))))
    test_accs.append(svm.SVC(C=clfs[i].best_estimator_.C, kernel='linear', gam
ma=clfs[i].best_estimator_.gamma).fit(mushroom_X_train_20[i],mushroom_Y_train_
20[i]).score(mushroom_X_test_80[i], mushroom_Y_test_80[i]))
    print(i)
```

```
0

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

1
2

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
```

In [24]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: [1.]
```

Wine

80 / 20

In [296]:
```
clfs = []

for i in range(3):
    clf = GridSearchCV(cv=5, estimator=svm.SVC(), param_grid=[{'C': C_list, 'k
ernel': ['linear'], 'gamma': G_list}], n_jobs=-1)
    clf.fit(wine_X_train_80[i].values, wine_Y_train_80[i].values)
    clfs.append(clf)
    print(i)
```

```
0
1
2
```

In [297]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16,
1))))
    val_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16, 1
))))
    test_accs.append(svm.SVC(C=clfs[i].best_estimator_.C, kernel='linear', gam
ma=clfs[i].best_estimator_.gamma).fit(wine_X_train_80[i],wine_Y_train_80[i]).s
core(wine_X_test_20[i], wine_Y_test_20[i]))
    print(i)
```

```
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

0

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

1

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

2
```

In [298]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: [0.72604427]
```

50 / 50

In [259]:
```
clfs = []

for i in range(3):
    clf = GridSearchCV(cv=5, estimator=svm.SVC(), param_grid=[{'C': C_list, 'k
ernel': ['linear'], 'gamma': G_list}], n_jobs=-1)
    clf.fit(wine_X_train_50[i].values, wine_Y_train_50[i].values)
    clfs.append(clf)
    print(i)
```

```
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:
652: Warning: The least populated class in y has only 3 members, which is too
few. The minimum number of members in any class cannot be less than n_splits=
5.
  % (min_groups, self.n_splits)), Warning)

0

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:
652: Warning: The least populated class in y has only 2 members, which is too
few. The minimum number of members in any class cannot be less than n_splits=
5.
  % (min_groups, self.n_splits)), Warning)

1

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:
652: Warning: The least populated class in y has only 3 members, which is too
few. The minimum number of members in any class cannot be less than n_splits=
5.
  % (min_groups, self.n_splits)), Warning)

2
```

In [260]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16,
1))))
    val_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16, 1
))))
    test_accs.append(svm.SVC(C=clfs[i].best_estimator_.C, kernel='linear', gam
ma=clfs[i].best_estimator_.gamma).fit(wine_X_train_50[i],wine_Y_train_50[i]).s
core(wine_X_test_50[i], wine_Y_test_50[i]))
    print(i)
```

```
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

0

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

1

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

2
```

In [261]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: [0.49249977]
```

20 / 80

In [262]:
```
clfs = []

for i in range(3):
    clf = GridSearchCV(cv=5, estimator=svm.SVC(), param_grid=[{'C': C_list, 'k
ernel': ['linear'], 'gamma': G_list}], n_jobs=-1)
    clf.fit(wine_X_train_20[i].values, wine_Y_train_20[i].values)
    clfs.append(clf)
    print(i)
```

```
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:
652: Warning: The least populated class in y has only 1 members, which is too
few. The minimum number of members in any class cannot be less than n_splits=
5.
  % (min_groups, self.n_splits)), Warning)
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:841: DeprecationWarning: The default of the `iid` parameter will change fro
m True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:
652: Warning: The least populated class in y has only 2 members, which is too
few. The minimum number of members in any class cannot be less than n_splits=
5.
  % (min_groups, self.n_splits)), Warning)

0

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:841: DeprecationWarning: The default of the `iid` parameter will change fro
m True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)

1

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:
652: Warning: The least populated class in y has only 2 members, which is too
few. The minimum number of members in any class cannot be less than n_splits=
5.
  % (min_groups, self.n_splits)), Warning)
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:841: DeprecationWarning: The default of the `iid` parameter will change fro
m True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)

2
```

In [263]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16,
1))))
    val_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16, 1
))))
    test_accs.append(svm.SVC(C=clfs[i].best_estimator_.C, kernel='linear', gam
ma=clfs[i].best_estimator_.gamma).fit(wine_X_train_20[i],wine_Y_train_20[i]).s
core(wine_X_test_80[i], wine_Y_test_80[i]))
    print(i)
```

```
0

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

1

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

2
```

In [264]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: [0.48319063]
```

Dota

80 / 20

In [265]:
```
clfs = []

for i in range(3):
    clf = GridSearchCV(cv=5, estimator=svm.SVC(), param_grid=[{'C': C_list, 'k
ernel': ['linear'], 'gamma': G_list}], n_jobs=-1)
    clf.fit(dota_X_train_80[i].values, dota_Y_train_80[i].values)
    clfs.append(clf)
    print(i)
```

```
0
1
2
```

In [266]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16,
1))))
    val_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16, 1
))))
    test_accs.append(svm.SVC(C=clfs[i].best_estimator_.C, kernel='linear', gam
ma=clfs[i].best_estimator_.gamma).fit(dota_X_train_80[i],dota_Y_train_80[i]).s
core(dota_X_test_20[i], dota_Y_test_20[i]))
    print(i)
```

```
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

0

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

1

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

2
```

In [267]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: [0.62337498]
```

50 / 50

In [268]:
```
clfs = []

for i in range(3):
    clf = GridSearchCV(cv=5, estimator=svm.SVC(), param_grid=[{'C': C_list, 'k
ernel': ['linear'], 'gamma': G_list}], n_jobs=-1)
    clf.fit(dota_X_train_50[i].values, dota_Y_train_50[i].values)
    clfs.append(clf)
    print(i)
```

```
0
1
2
```

In [269]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16,
1))))
    val_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16, 1
))))
    test_accs.append(svm.SVC(C=clfs[i].best_estimator_.C, kernel='linear', gam
ma=clfs[i].best_estimator_.gamma).fit(dota_X_train_50[i],dota_Y_train_50[i]).s
core(dota_X_test_50[i], dota_Y_test_50[i]))
    print(i)
```

```
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

0

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

1

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

2
```

In [270]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: [0.62846661]
```

20 / 80

In [271]:
```
clfs = []

for i in range(3):
    clf = GridSearchCV(cv=5, estimator=svm.SVC(), param_grid=[{'C': C_list, 'k
ernel': ['linear'], 'gamma': G_list}], n_jobs=-1)
    clf.fit(dota_X_train_20[i].values, dota_Y_train_20[i].values)
    clfs.append(clf)
    print(i)
```

```
0
1
2
```

In [272]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16,
1))))
    val_accs.append(max(clfs[i].cv_results_['mean_train_score'].reshape((16, 1
))))
    test_accs.append(svm.SVC(C=clfs[i].best_estimator_.C, kernel='linear', gam
ma=clfs[i].best_estimator_.gamma).fit(dota_X_train_20[i],dota_Y_train_20[i]).s
core(dota_X_test_80[i], dota_Y_test_80[i]))
    print(i)
```

```
C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

0

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

1

C:\Users\ejozy\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125:
FutureWarning: You are accessing a training score ('mean_train_score'), which
will not be available by default any more in 0.21. If you need training score
s, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)

2
```

In [273]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: [0.65866134]
```

Logistic Regression

Mushroom

80 / 20

In [329]:
```
logregs = []

for i in range(3):
    logreg = LogisticRegressionCV(Cs= C_list, solver='liblinear', multi_class=
'ovr', n_jobs=-1, cv=5)
    logreg.fit(mushroom_X_train_80[i].values, mushroom_Y_train_80[i].values)
    logregs.append(logreg)
    print(i)
```

```
0
1
2
```

In [330]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(logregs[i].score(mushroom_X_train_80[i], mushroom_Y_trai
n_80[i]))
    val_accs.append(logregs[i].score(mushroom_X_train_80[i], mushroom_Y_train_
80[i]))
    test_accs.append(logregs[i].score(mushroom_X_test_20[i], mushroom_Y_test_2
0[i]))
    print(i)
```

```
0
1
2
```

In [331]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.9998333333333335
```

50 / 50

In [332]:
```
logregs = []

for i in range(3):
    logreg = LogisticRegressionCV(Cs= C_list, solver='liblinear', multi_class=
'ovr', n_jobs=-1, cv=5)
    logreg.fit(mushroom_X_train_50[i].values, mushroom_Y_train_50[i].values)
    logregs.append(logreg)
    print(i)
```

```
0
1
2
```

In [333]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(logregs[i].score(mushroom_X_train_50[i], mushroom_Y_trai
n_50[i]))
    val_accs.append(logregs[i].score(mushroom_X_train_50[i], mushroom_Y_train_
50[i]))
    test_accs.append(logregs[i].score(mushroom_X_test_50[i], mushroom_Y_test_5
0[i]))
    print(i)
```

```
0
1
2
```

In [334]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.9994666666666667
```

20 / 80

In [335]:
```
logregs = []

for i in range(3):
    logreg = LogisticRegressionCV(Cs= C_list, solver='liblinear', multi_class=
'ovr', n_jobs=-1, cv=5)
    logreg.fit(mushroom_X_train_20[i].values, mushroom_Y_train_20[i].values)
    logregs.append(logreg)
    print(i)
```

```
0
1
2
```

In [336]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(logregs[i].score(mushroom_X_train_20[i], mushroom_Y_trai
n_20[i]))
    val_accs.append(logregs[i].score(mushroom_X_train_20[i], mushroom_Y_train_
20[i]))
    test_accs.append(logregs[i].score(mushroom_X_test_80[i], mushroom_Y_test_8
0[i]))
    print(i)
```

```
0
1
2
```

In [337]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.999
```

Wine

80 / 20

In [290]:
```
logregs = []

for i in range(3):
    logreg = LogisticRegressionCV(Cs= C_list, solver='liblinear', multi_class=
'ovr', n_jobs=-1, cv=5)
    logreg.fit(wine_X_train_80[i].values, wine_Y_train_80[i].values)
    logregs.append(logreg)
    print(i)
```

```
0
1
2
```

In [291]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(logregs[i].score(wine_X_train_80[i], wine_Y_train_80[i
]))
    val_accs.append(logregs[i].score(wine_X_train_80[i], wine_Y_train_80[i]))
    test_accs.append(logregs[i].score(wine_X_test_20[i], wine_Y_test_20[i]))
    print(i)
```

```
0
1
2
```

In [292]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.72375
```

50 / 50

In [293]:
```
logregs = []

for i in range(3):
    logreg = LogisticRegressionCV(Cs= C_list, solver='liblinear', multi_class=
'ovr', n_jobs=-1, cv=5)
    logreg.fit(wine_X_train_50[i].values, wine_Y_train_50[i].values)
    logregs.append(logreg)
    print(i)
```

```
0
1
2
```

In [294]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(logregs[i].score(wine_X_train_50[i], wine_Y_train_50[i
]))
    val_accs.append(logregs[i].score(wine_X_train_50[i], wine_Y_train_50[i]))
    test_accs.append(logregs[i].score(wine_X_test_50[i], wine_Y_test_50[i]))
    print(i)
```

```
0
1
2
```

In [295]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.731
```

20 / 80

In [287]:
```
logregs = []

for i in range(3):
    logreg = LogisticRegressionCV(Cs= C_list, solver='liblinear', multi_class=
'ovr', n_jobs=-1, cv=5)
    logreg.fit(wine_X_train_20[i].values, wine_Y_train_20[i].values)
    logregs.append(logreg)
    print(i)
```

```
0
1
2
```

In [288]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(logregs[i].score(wine_X_train_20[i], wine_Y_train_20[i
]))
    val_accs.append(logregs[i].score(wine_X_train_20[i], wine_Y_train_20[i]))
    test_accs.append(logregs[i].score(wine_X_test_80[i], wine_Y_test_80[i]))
    print(i)
```

```
0
1
2
```

In [289]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.7170833333333334
```

Dota

80 / 20

In [165]:
```
logregs = []

for i in range(3):
    logreg = LogisticRegressionCV(Cs= C_list, solver='liblinear', multi_class=
'ovr', n_jobs=-1, cv=5)
    logreg.fit(dota_X_train_80[i].values, dota_Y_train_80[i].values)
    logregs.append(logreg)
    print(i)
```

```
0
1
2
```

In [166]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(logregs[i].score(dota_X_train_80[i], dota_Y_train_80[i
]))
    val_accs.append(logregs[i].score(dota_X_train_80[i], dota_Y_train_80[i]))
    test_accs.append(logregs[i].score(dota_X_test_20[i], dota_Y_test_20[i]))
    print(i)
```

```
0
1
2
```

In [167]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.6168333333333333
```

50 / 50

In [168]:
```
logregs = []

for i in range(3):
    logreg = LogisticRegressionCV(Cs= C_list, solver='liblinear', multi_class=
'ovr', n_jobs=-1, cv=5)
    logreg.fit(dota_X_train_50[i].values, dota_Y_train_50[i].values)
    logregs.append(logreg)
    print(i)
```

```
0
1
2
```

In [169]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(logregs[i].score(dota_X_train_50[i], dota_Y_train_50[i
]))
    val_accs.append(logregs[i].score(dota_X_train_50[i], dota_Y_train_50[i]))
    test_accs.append(logregs[i].score(dota_X_test_50[i], dota_Y_test_50[i]))
    print(i)
```

```
0
1
2
```

In [170]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.6244
```

20 / 80

In [171]:
```
logregs = []

for i in range(3):
    logreg = LogisticRegressionCV(Cs= C_list, solver='liblinear', multi_class=
'ovr', n_jobs=-1, cv=5)
    logreg.fit(dota_X_train_20[i].values, dota_Y_train_20[i].values)
    logregs.append(logreg)
    print(i)
```

```
0
1
2
```

In [172]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(logregs[i].score(dota_X_train_20[i], dota_Y_train_20[i
]))
    val_accs.append(logregs[i].score(dota_X_train_20[i], dota_Y_train_20[i]))
    test_accs.append(logregs[i].score(dota_X_test_80[i], dota_Y_test_80[i]))
    print(i)
```

```
0
1
2
```

In [173]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.6466666666666666
```

Random Forest

Mushroom

80 / 20

In [174]:
```
rfs = []

for i in range(3):
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(mushroom_X_train_80[i].values, mushroom_Y_train_80[i].values)
    rfs.append(rf)
    print(i)
```

```
0
1
2
```

In [189]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(rfs[i].score(mushroom_X_train_80[i], mushroom_Y_train_80
[i]))
    val_accs.append(rfs[i].score(mushroom_X_train_80[i], mushroom_Y_train_80[i
]))
    test_accs.append(rfs[i].score(mushroom_X_test_20[i], mushroom_Y_test_20[i
]))
    print(i)
```

```
0
1
2
```

In [191]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 1.0
```

50 / 50

In [192]:
```
rfs = []

for i in range(3):
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(mushroom_X_train_50[i].values, mushroom_Y_train_50[i].values)
    rfs.append(rf)
    print(i)
```

```
0
1
2
```

In [193]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(rfs[i].score(mushroom_X_train_50[i], mushroom_Y_train_50
[i]))
    val_accs.append(rfs[i].score(mushroom_X_train_50[i], mushroom_Y_train_50[i
]))
    test_accs.append(rfs[i].score(mushroom_X_test_50[i], mushroom_Y_test_50[i
]))
    print(i)
```

```
0
1
2
```

In [194]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 1.0
```

20 / 80

In [195]:
```
rfs = []

for i in range(3):
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(mushroom_X_train_20[i].values, mushroom_Y_train_20[i].values)
    rfs.append(rf)
    print(i)
```

```
0
1
2
```

In [196]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(rfs[i].score(mushroom_X_train_20[i], mushroom_Y_train_20
[i]))
    val_accs.append(rfs[i].score(mushroom_X_train_20[i], mushroom_Y_train_20[i
]))
    test_accs.append(rfs[i].score(mushroom_X_test_80[i], mushroom_Y_test_80[i
]))
    print(i)
```

```
0
1
2
```

In [197]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 1.0
```

Wine

80 / 20

In [198]:
```
rfs = []

for i in range(3):
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(wine_X_train_80[i].values, wine_Y_train_80[i].values)
    rfs.append(rf)
    print(i)
```

```
0
1
2
```

In [200]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(rfs[i].score(wine_X_train_80[i], wine_Y_train_80[i]))
    val_accs.append(rfs[i].score(wine_X_train_80[i], wine_Y_train_80[i]))
    test_accs.append(rfs[i].score(wine_X_test_20[i], wine_Y_test_20[i]))
    print(i)
```

```
0
1
2
```

In [201]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.8975
```

50 / 50

In [202]:
```
rfs = []

for i in range(3):
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(wine_X_train_50[i].values, wine_Y_train_50[i].values)
    rfs.append(rf)
    print(i)
```

```
0
1
2
```

In [203]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(rfs[i].score(wine_X_train_50[i], wine_Y_train_50[i]))
    val_accs.append(rfs[i].score(wine_X_train_50[i], wine_Y_train_50[i]))
    test_accs.append(rfs[i].score(wine_X_test_50[i], wine_Y_test_50[i]))
    print(i)
```

```
0
1
2
```

In [204]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.868
```

20 / 80

In [205]:
```
rfs = []

for i in range(3):
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(wine_X_train_20[i].values, wine_Y_train_20[i].values)
    rfs.append(rf)
    print(i)
```

```
0
1
2
```

In [208]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(rfs[i].score(wine_X_train_20[i], wine_Y_train_20[i]))
    val_accs.append(rfs[i].score(wine_X_train_20[i], wine_Y_train_20[i]))
    test_accs.append(rfs[i].score(wine_X_test_80[i], wine_Y_test_80[i]))
    print(i)
```

```
0
1
2
```

In [209]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.8470833333333333
```

Dota

80 / 20

In [228]:
```
rfs = []

for i in range(3):
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(dota_X_train_80[i].values, dota_Y_train_80[i].values)
    rfs.append(rf)
    print(i)
```

```
0
1
2
```

In [229]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(rfs[i].score(dota_X_train_80[i], dota_Y_train_80[i]))
    val_accs.append(rfs[i].score(dota_X_train_80[i], dota_Y_train_80[i]))
    test_accs.append(rfs[i].score(dota_X_test_20[i], dota_Y_test_20[i]))
    print(i)
```

```
0
1
2
```

In [230]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.8546666666666667
```

50 / 50

In [213]:
```
rfs = []

for i in range(3):
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(dota_X_train_50[i].values, dota_Y_train_50[i].values)
    rfs.append(rf)
    print(i)
```

```
0
1
2
```

In [214]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(rfs[i].score(dota_X_train_50[i], dota_Y_train_50[i]))
    val_accs.append(rfs[i].score(dota_X_train_50[i], dota_Y_train_50[i]))
    test_accs.append(rfs[i].score(dota_X_test_50[i], dota_Y_test_50[i]))
    print(i)
```

```
0
1
2
```

In [215]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.8525333333333333
```

20 / 80

In [216]:
```
rfs = []

for i in range(3):
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(dota_X_train_20[i].values, dota_Y_train_20[i].values)
    rfs.append(rf)
    print(i)
```

```
0
1
2
```

In [217]:
```
train_accs = []
val_accs = []
test_accs = []

for i in range(3):
    train_accs.append(rfs[i].score(dota_X_train_20[i], dota_Y_train_20[i]))
    val_accs.append(rfs[i].score(dota_X_train_20[i], dota_Y_train_20[i]))
    test_accs.append(rfs[i].score(dota_X_test_80[i], dota_Y_test_80[i]))
    print(i)
```

```
0
1
2
```

In [218]:
```
accs = []

for i in range(3):
    accs.append((train_accs[i] + val_accs[i] + test_accs[i])/3)

print('Best Accuracy: ' + str(max(accs)))
```

```
Best Accuracy: 0.8496666666666667
```

In [ ]:
```

```