import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
import mlrose\_hiive as mlrose  
from sklearn.metrics import accuracy\_score, f1\_score  
import seaborn as sns  
import time  
  
from mlrose\_hiive.algorithms.decay import GeomDecay  
  
  
#Random State  
rs = 614

class Data():  
 def dataAllocation(self, path):  
 df = pd.read\_csv(path)  
 x\_data = df.iloc[:, :-1]  
 y\_data = df.iloc[:, -1 ]  
 return x\_data,y\_data  
  
 def trainSets(self,x\_data,y\_data):  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size = 0.3, random\_state = rs, shuffle = True)  
 return x\_train, x\_test, y\_train, y\_test

dataset = Data()  
x1\_data,y1\_data = dataset.dataAllocation('data/pima-indians-diabetes.csv')  
x1\_train, x1\_test, y1\_train, y1\_test = dataset.trainSets(x1\_data,y1\_data)  
scaler = StandardScaler()  
scaled\_x1\_train = scaler.fit\_transform(x1\_train)  
scaled\_x1\_test = scaler.transform(x1\_test)

algorithms = ['random\_hill\_climb', 'simulated\_annealing', 'genetic\_alg']  
algorithm = algorithms[0]

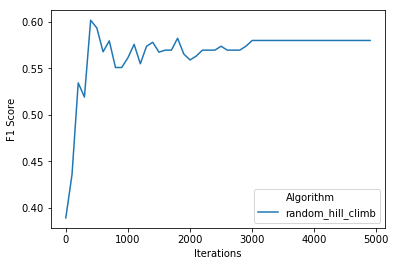
results = []  
for i in range(1, 5000, 100):  
 model = mlrose.NeuralNetwork(hidden\_nodes=[4], activation='relu',  
 algorithm=algorithm, max\_iters=i,  
 bias=True, is\_classifier=True, learning\_rate=0.1,  
 early\_stopping=True, clip\_max=5, max\_attempts=100,  
 random\_state=rs)  
 model.fit(scaled\_x1\_train, y1\_train)  
 y\_train\_pred = model.predict(scaled\_x1\_train)  
 y\_train\_accuracy = accuracy\_score(y1\_train, y\_train\_pred)  
  
 y\_test\_pred = model.predict(scaled\_x1\_test)  
 y\_test\_accuracy = accuracy\_score(y1\_test, y\_test\_pred)  
  
 f1score = f1\_score(y1\_test, y\_test\_pred)  
  
 results.append([i, algorithm, y\_train\_accuracy, y\_test\_accuracy, f1score])  
 print([i, algorithm, y\_train\_accuracy, y\_test\_accuracy, f1score])

[1, 'random\_hill\_climb', 0.521415270018622, 0.49783549783549785, 0.38947368421052625]  
[101, 'random\_hill\_climb', 0.6089385474860335, 0.6190476190476191, 0.4358974358974359]  
[201, 'random\_hill\_climb', 0.7411545623836127, 0.7056277056277056, 0.5342465753424657]  
[301, 'random\_hill\_climb', 0.7653631284916201, 0.7272727272727273, 0.5190839694656489]  
[401, 'random\_hill\_climb', 0.7839851024208566, 0.7532467532467533, 0.6013986013986014]  
[501, 'random\_hill\_climb', 0.7914338919925512, 0.7445887445887446, 0.593103448275862]  
[601, 'random\_hill\_climb', 0.7914338919925512, 0.7229437229437229, 0.5675675675675677]  
[701, 'random\_hill\_climb', 0.7951582867783985, 0.7359307359307359, 0.5793103448275863]  
[801, 'random\_hill\_climb', 0.7932960893854749, 0.7316017316017316, 0.5507246376811594]  
[901, 'random\_hill\_climb', 0.8044692737430168, 0.7316017316017316, 0.5507246376811594]  
[1001, 'random\_hill\_climb', 0.8026070763500931, 0.7359307359307359, 0.5611510791366906]  
[1101, 'random\_hill\_climb', 0.8081936685288641, 0.7445887445887446, 0.5755395683453237]  
[1201, 'random\_hill\_climb', 0.813780260707635, 0.7359307359307359, 0.5547445255474452]  
[1301, 'random\_hill\_climb', 0.819366852886406, 0.7489177489177489, 0.573529411764706]  
[1401, 'random\_hill\_climb', 0.8100558659217877, 0.7532467532467533, 0.5777777777777778]  
[1501, 'random\_hill\_climb', 0.813780260707635, 0.7489177489177489, 0.5671641791044777]  
[1601, 'random\_hill\_climb', 0.8119180633147114, 0.7445887445887446, 0.5693430656934306]  
[1701, 'random\_hill\_climb', 0.813780260707635, 0.7445887445887446, 0.5693430656934306]  
[1801, 'random\_hill\_climb', 0.8119180633147114, 0.7575757575757576, 0.582089552238806]  
[1901, 'random\_hill\_climb', 0.8175046554934823, 0.7402597402597403, 0.5652173913043478]  
[2001, 'random\_hill\_climb', 0.8175046554934823, 0.7402597402597403, 0.5588235294117648]  
[2101, 'random\_hill\_climb', 0.8175046554934823, 0.7445887445887446, 0.562962962962963]  
[2201, 'random\_hill\_climb', 0.8175046554934823, 0.7445887445887446, 0.5693430656934306]  
[2301, 'random\_hill\_climb', 0.8175046554934823, 0.7445887445887446, 0.5693430656934306]  
[2401, 'random\_hill\_climb', 0.813780260707635, 0.7445887445887446, 0.5693430656934306]  
[2501, 'random\_hill\_climb', 0.8156424581005587, 0.7489177489177489, 0.573529411764706]  
[2601, 'random\_hill\_climb', 0.813780260707635, 0.7445887445887446, 0.5693430656934306]  
[2701, 'random\_hill\_climb', 0.8119180633147114, 0.7445887445887446, 0.5693430656934306]  
[2801, 'random\_hill\_climb', 0.8156424581005587, 0.7445887445887446, 0.5693430656934306]  
[2901, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.573529411764706]  
[3001, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[3101, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[3201, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[3301, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[3401, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[3501, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[3601, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[3701, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[3801, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[3901, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[4001, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[4101, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[4201, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[4301, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[4401, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[4501, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[4601, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[4701, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[4801, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]  
[4901, 'random\_hill\_climb', 0.8119180633147114, 0.7489177489177489, 0.5797101449275361]

df = pd.DataFrame(results, columns=["Iterations", "Algorithm", "Train Accuracy", "Test Accuracy", "F1 Score"])

sns.lineplot(data=df, x="Iterations", y="F1 Score", hue="Algorithm")

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fac58cff390>



df.to\_csv("nn\_opts\_rhc.csv", index=False)

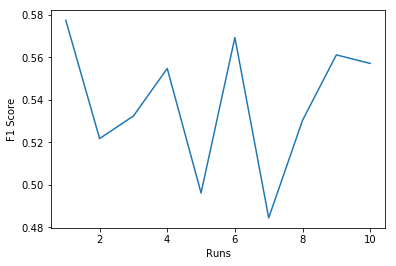
# Randomized Hill Climbing with 10 restarts

results = []  
for i in range(1, 11):  
 model = mlrose.NeuralNetwork(hidden\_nodes=[4], activation='relu',  
 algorithm=algorithm, max\_iters=3000,  
 bias=True, is\_classifier=True, learning\_rate=0.1,  
 early\_stopping=True, clip\_max=5, max\_attempts=100,  
 restarts=10)  
 model.fit(scaled\_x1\_train, y1\_train)  
 y\_train\_pred = model.predict(scaled\_x1\_train)  
 y\_train\_accuracy = accuracy\_score(y1\_train, y\_train\_pred)  
  
 y\_test\_pred = model.predict(scaled\_x1\_test)  
 y\_test\_accuracy = accuracy\_score(y1\_test, y\_test\_pred)  
  
 f1score = f1\_score(y1\_test, y\_test\_pred)  
 results.append([i, algorithm, y\_train\_accuracy, y\_test\_accuracy, f1score])  
 print([i, algorithm, y\_train\_accuracy, y\_test\_accuracy, f1score])

[1, 'random\_hill\_climb', 0.8100558659217877, 0.7402597402597403, 0.5774647887323944]  
[2, 'random\_hill\_climb', 0.8100558659217877, 0.7142857142857143, 0.5217391304347826]  
[3, 'random\_hill\_climb', 0.8175046554934823, 0.7186147186147186, 0.5323741007194244]  
[4, 'random\_hill\_climb', 0.819366852886406, 0.7359307359307359, 0.5547445255474452]  
[5, 'random\_hill\_climb', 0.8026070763500931, 0.7186147186147186, 0.49612403100775193]  
[6, 'random\_hill\_climb', 0.813780260707635, 0.7445887445887446, 0.5693430656934306]  
[7, 'random\_hill\_climb', 0.7951582867783985, 0.7142857142857143, 0.484375]  
[8, 'random\_hill\_climb', 0.7951582867783985, 0.7316017316017316, 0.5303030303030304]  
[9, 'random\_hill\_climb', 0.8063314711359404, 0.7359307359307359, 0.5611510791366906]  
[10, 'random\_hill\_climb', 0.8119180633147114, 0.7316017316017316, 0.557142857142857]

runs = pd.DataFrame(results, columns=["Runs", "Algorithm", "Train Accuracy", "Test Accuracy", "F1 Score"])  
sns.lineplot(data=runs, x="Runs", y="F1 Score")

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fac5b1a8390>



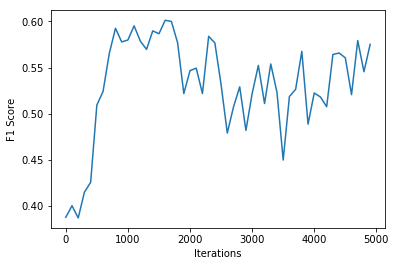
runs.to\_csv('nn\_opts\_rhc\_runs.csv', index=False)

# Analyze Decay Rate for simulated\_annealing

results = []  
algorithm = algorithms[1]  
for i in range(1, 5000, 100):  
 model = mlrose.NeuralNetwork(hidden\_nodes=[4], activation='relu',  
 algorithm=algorithm, max\_iters=i,  
 bias=True, is\_classifier=True, learning\_rate=0.1,  
 early\_stopping=True, clip\_max=5, max\_attempts=100,  
 random\_state=rs)  
 model.fit(scaled\_x1\_train, y1\_train)  
 y\_train\_pred = model.predict(scaled\_x1\_train)  
 y\_train\_accuracy = accuracy\_score(y1\_train, y\_train\_pred)  
  
 y\_test\_pred = model.predict(scaled\_x1\_test)  
 y\_test\_accuracy = accuracy\_score(y1\_test, y\_test\_pred)  
  
 f1score = f1\_score(y1\_test, y\_test\_pred)  
  
 results.append([i, algorithm, y\_train\_accuracy, y\_test\_accuracy, f1score])  
 print([i, algorithm, y\_train\_accuracy, y\_test\_accuracy, f1score])

[1, 'simulated\_annealing', 0.521415270018622, 0.4935064935064935, 0.38743455497382195]  
[101, 'simulated\_annealing', 0.5195530726256983, 0.5064935064935064, 0.4]  
[201, 'simulated\_annealing', 0.4748603351955307, 0.43722943722943725, 0.38679245283018865]  
[301, 'simulated\_annealing', 0.4934823091247672, 0.45021645021645024, 0.4147465437788019]  
[401, 'simulated\_annealing', 0.4823091247672253, 0.45021645021645024, 0.42533936651583715]  
[501, 'simulated\_annealing', 0.5605214152700186, 0.5411255411255411, 0.5092592592592593]  
[601, 'simulated\_annealing', 0.6443202979515829, 0.6147186147186147, 0.5240641711229946]  
[701, 'simulated\_annealing', 0.6871508379888268, 0.6536796536796536, 0.5652173913043479]  
[801, 'simulated\_annealing', 0.7262569832402235, 0.7142857142857143, 0.5925925925925926]  
[901, 'simulated\_annealing', 0.7374301675977654, 0.670995670995671, 0.5777777777777777]  
[1001, 'simulated\_annealing', 0.7541899441340782, 0.6926406926406926, 0.5798816568047337]  
[1101, 'simulated\_annealing', 0.7597765363128491, 0.7056277056277056, 0.5952380952380953]  
[1201, 'simulated\_annealing', 0.7579143389199255, 0.70995670995671, 0.5786163522012578]  
[1301, 'simulated\_annealing', 0.7374301675977654, 0.6926406926406926, 0.5696969696969698]  
[1401, 'simulated\_annealing', 0.7523277467411545, 0.7229437229437229, 0.5897435897435898]  
[1501, 'simulated\_annealing', 0.7635009310986964, 0.7316017316017316, 0.5866666666666667]  
[1601, 'simulated\_annealing', 0.7635009310986964, 0.7359307359307359, 0.6013071895424836]  
[1701, 'simulated\_annealing', 0.7877094972067039, 0.7402597402597403, 0.6]  
[1801, 'simulated\_annealing', 0.776536312849162, 0.7142857142857143, 0.576923076923077]  
[1901, 'simulated\_annealing', 0.7932960893854749, 0.7142857142857143, 0.5217391304347826]  
[2001, 'simulated\_annealing', 0.7914338919925512, 0.7056277056277056, 0.5466666666666666]  
[2101, 'simulated\_annealing', 0.7839851024208566, 0.7229437229437229, 0.5492957746478873]  
[2201, 'simulated\_annealing', 0.7802607076350093, 0.7142857142857143, 0.5217391304347826]  
[2301, 'simulated\_annealing', 0.7672253258845437, 0.70995670995671, 0.5838509316770185]  
[2401, 'simulated\_annealing', 0.7839851024208566, 0.7012987012987013, 0.5766871165644172]  
[2501, 'simulated\_annealing', 0.7877094972067039, 0.70995670995671, 0.5314685314685313]  
[2601, 'simulated\_annealing', 0.7858472998137802, 0.6796536796536796, 0.47887323943661975]  
[2701, 'simulated\_annealing', 0.7988826815642458, 0.7056277056277056, 0.5072463768115942]  
[2801, 'simulated\_annealing', 0.7895716945996276, 0.683982683982684, 0.5290322580645161]  
[2901, 'simulated\_annealing', 0.8100558659217877, 0.6926406926406926, 0.48175182481751827]  
[3001, 'simulated\_annealing', 0.7951582867783985, 0.7142857142857143, 0.5217391304347826]  
[3101, 'simulated\_annealing', 0.8026070763500931, 0.7402597402597403, 0.5522388059701493]  
[3201, 'simulated\_annealing', 0.7932960893854749, 0.70995670995671, 0.510948905109489]  
[3301, 'simulated\_annealing', 0.7951582867783985, 0.7489177489177489, 0.5538461538461539]  
[3401, 'simulated\_annealing', 0.7951582867783985, 0.7316017316017316, 0.523076923076923]  
[3501, 'simulated\_annealing', 0.7932960893854749, 0.6926406926406926, 0.4496124031007752]  
[3601, 'simulated\_annealing', 0.7970204841713222, 0.7186147186147186, 0.5185185185185186]  
[3701, 'simulated\_annealing', 0.7932960893854749, 0.7272727272727273, 0.5263157894736842]  
[3801, 'simulated\_annealing', 0.7858472998137802, 0.7229437229437229, 0.5675675675675677]  
[3901, 'simulated\_annealing', 0.7932960893854749, 0.70995670995671, 0.48854961832061067]  
[4001, 'simulated\_annealing', 0.7895716945996276, 0.7229437229437229, 0.5223880597014926]  
[4101, 'simulated\_annealing', 0.7895716945996276, 0.70995670995671, 0.5179856115107914]  
[4201, 'simulated\_annealing', 0.7914338919925512, 0.7142857142857143, 0.5074626865671641]  
[4301, 'simulated\_annealing', 0.7858472998137802, 0.7056277056277056, 0.5641025641025642]  
[4401, 'simulated\_annealing', 0.7914338919925512, 0.7142857142857143, 0.5657894736842105]  
[4501, 'simulated\_annealing', 0.7839851024208566, 0.7012987012987013, 0.5605095541401275]  
[4601, 'simulated\_annealing', 0.7895716945996276, 0.696969696969697, 0.5205479452054794]  
[4701, 'simulated\_annealing', 0.7839851024208566, 0.7359307359307359, 0.5793103448275863]  
[4801, 'simulated\_annealing', 0.7877094972067039, 0.7186147186147186, 0.5454545454545454]  
[4901, 'simulated\_annealing', 0.7858472998137802, 0.7186147186147186, 0.5751633986928105]

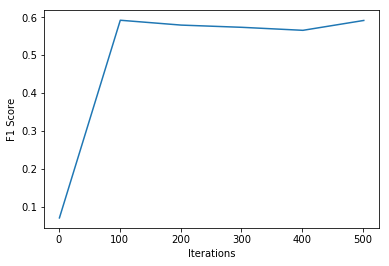
df = pd.DataFrame(results, columns=["Iterations", "Algorithm", "Train Accuracy", "Test Accuracy", "F1 Score"])  
sns.lineplot(data=df, x="Iterations", y="F1 Score")  
df.to\_csv("nn\_opts\_sa.csv", index=False)



results = []  
algorithm = algorithms[2]  
for i in range(1, 600, 100):  
 model = mlrose.NeuralNetwork(hidden\_nodes=[4], activation='relu',  
 algorithm=algorithm, max\_iters=i,  
 bias=True, is\_classifier=True, learning\_rate=0.1,  
 early\_stopping=True, clip\_max=5, max\_attempts=100,  
 random\_state=rs, pop\_size=200, mutation\_prob = 0.1)  
  
   
 model.fit(scaled\_x1\_train, y1\_train)  
 y\_train\_pred = model.predict(scaled\_x1\_train)  
 y\_train\_accuracy = accuracy\_score(y1\_train, y\_train\_pred)  
  
 y\_test\_pred = model.predict(scaled\_x1\_test)  
 y\_test\_accuracy = accuracy\_score(y1\_test, y\_test\_pred)  
  
 f1score = f1\_score(y1\_test, y\_test\_pred)  
  
 results.append([i, algorithm, y\_train\_accuracy, y\_test\_accuracy, f1score])  
 print([i, algorithm, y\_train\_accuracy, y\_test\_accuracy, f1score])

[1, 'genetic\_alg', 0.6424581005586593, 0.658008658008658, 0.07058823529411765]  
[101, 'genetic\_alg', 0.7728119180633147, 0.7316017316017316, 0.5921052631578947]  
[201, 'genetic\_alg', 0.7914338919925512, 0.7359307359307359, 0.5793103448275863]  
[301, 'genetic\_alg', 0.7895716945996276, 0.7359307359307359, 0.5734265734265734]  
[401, 'genetic\_alg', 0.7970204841713222, 0.7272727272727273, 0.5655172413793104]  
[501, 'genetic\_alg', 0.7914338919925512, 0.7489177489177489, 0.591549295774648]

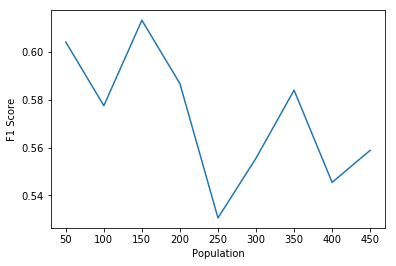
df = pd.DataFrame(results, columns=["Iterations", "Algorithm", "Train Accuracy", "Test Accuracy", "F1 Score"])  
sns.lineplot(data=df, x="Iterations", y="F1 Score")  
df.to\_csv("nn\_opts\_ga.csv", index=False)



results = []  
algorithm = algorithms[2]  
for i in range(50, 500, 50):  
 model = mlrose.NeuralNetwork(hidden\_nodes=[4], activation='relu',  
 algorithm=algorithm, max\_iters=150,  
 bias=True, is\_classifier=True, learning\_rate=0.1,  
 early\_stopping=True, clip\_max=5, max\_attempts=100,  
 random\_state=rs, pop\_size=i, mutation\_prob = 0.1)  
 model.fit(scaled\_x1\_train, y1\_train)  
 y\_train\_pred = model.predict(scaled\_x1\_train)  
 y\_train\_accuracy = accuracy\_score(y1\_train, y\_train\_pred)  
  
 y\_test\_pred = model.predict(scaled\_x1\_test)  
 y\_test\_accuracy = accuracy\_score(y1\_test, y\_test\_pred)  
  
 f1score = f1\_score(y1\_test, y\_test\_pred)  
 results.append([i, algorithm, y\_train\_accuracy, y\_test\_accuracy, f1score])  
 print([i, algorithm, y\_train\_accuracy, y\_test\_accuracy, f1score])

[50, 'genetic\_alg', 0.7895716945996276, 0.7445887445887446, 0.6040268456375839]  
[100, 'genetic\_alg', 0.7858472998137802, 0.7402597402597403, 0.5774647887323944]  
[150, 'genetic\_alg', 0.7802607076350093, 0.7705627705627706, 0.6131386861313869]  
[200, 'genetic\_alg', 0.7783985102420856, 0.7316017316017316, 0.5866666666666667]  
[250, 'genetic\_alg', 0.7783985102420856, 0.7012987012987013, 0.5306122448979592]  
[300, 'genetic\_alg', 0.7783985102420856, 0.7229437229437229, 0.5555555555555556]  
[350, 'genetic\_alg', 0.7914338919925512, 0.7532467532467533, 0.583941605839416]  
[400, 'genetic\_alg', 0.7839851024208566, 0.7402597402597403, 0.5454545454545454]  
[450, 'genetic\_alg', 0.7802607076350093, 0.7402597402597403, 0.5588235294117648]

df = pd.DataFrame(results, columns=["Population", "Algorithm", "Train Accuracy", "Test Accuracy", "F1 Score"])  
sns.lineplot(data=df, x="Population", y="F1 Score")  
df.to\_csv("nn\_opts\_ga\_population.csv", index=False)



# Flip Flop Analysis

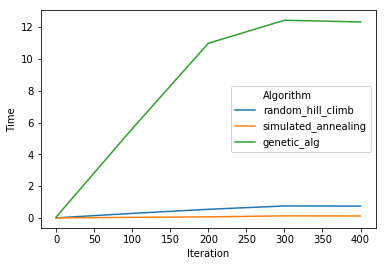
def rhc(max\_iters= 100):  
 start = time.time()  
 fitness\_score = mlrose.random\_hill\_climb(problem\_fit, max\_attempts=100, max\_iters= max\_iters, restarts=10, random\_state = rs)[1]  
 return [max\_iters, "random\_hill\_climb", fitness\_score, time.time()-start]  
  
def sa(max\_iters= 100):  
 start = time.time()  
 fitness\_score = mlrose.simulated\_annealing(problem\_fit, max\_attempts=100, max\_iters= max\_iters, random\_state = rs)[1]  
 return [max\_iters, "simulated\_annealing", fitness\_score, time.time()-start]  
  
def ga(max\_iters= 100):  
 start = time.time()  
 fitness\_score = mlrose.genetic\_alg(problem\_fit, max\_attempts=100, max\_iters= max\_iters, pop\_size= 200, mutation\_prob=0.1, random\_state = rs)[1]  
 return [max\_iters, "genetic\_alg", fitness\_score, time.time()-start]  
  
def mimic(max\_iters= 100):  
 start = time.time()  
 fitness\_score = mlrose.mimic(problem\_fit, max\_attempts=10, max\_iters= max\_iters, pop\_size= 200, keep\_pct=0.2, random\_state = rs)[1]  
 return [max\_iters, "mimic", fitness\_score, time.time()-start]

fitness = mlrose.FlipFlop()  
results = []  
problem\_fit = mlrose.DiscreteOpt(length = 100, fitness\_fn = fitness, maximize=True, max\_val = 2)  
for i in range(0, 100, 10):  
 results.append(rhc(max\_iters= i))  
 results.append(sa(max\_iters= i))  
 results.append(ga(max\_iters= i))  
 print(i, end=" ")  
  
 # start = time.time()  
 # fitness\_score = mlrose.simulated\_annealing(problem\_fit, max\_attempts=100, max\_iters= i, schedule= mlrose.GeomDecay(), random\_state = rs)[1]  
 # results.append([i, "simulated\_annealing", fitness\_score, time.time()-start])  
 # print([i, "simulated\_annealing", fitness\_score])  
 # start = time.time()  
 # fitness\_score = mlrose.genetic\_alg(problem\_fit, max\_attempts=100, max\_iters= i, pop\_size= 200, mutation\_prob=0.1, random\_state = rs)[1]  
 # results.append([i, "genetic\_alg", fitness\_score, time.time()-start])  
 # print([i, "genetic\_alg", fitness\_score])  
 # start = time.time()  
 # fitness\_score = mlrose.mimic(problem\_fit, max\_attempts=10, max\_iters= i, pop\_size= 200, keep\_pct=0.2, random\_state = rs)[1]  
 # results.append([i, "mimic", fitness\_score, time.time()-start])  
 # print([i, "mimic", fitness\_score])

0 100 200 300 400

df = pd.DataFrame(results, columns=["Iteration", "Algorithm", "Fitness", "Time"])  
sns.lineplot(data=df, x="Iteration", y="Time", hue="Algorithm")

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa38bfeb080>



df.to\_csv("flipflop.csv", index=False)