Cloud dataset

February 20, 2023

[59]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
import warnings
#suppress warnings
warnings.filterwarnings('ignore')

# Load the data into a DataFrame
url = "D:\ECE-GY 7143 Advanced Machine Learning\cloud.data"
df = pd.read_csv(url, header=None, delim_whitespace=True, usecols=[0, 1, 2, 3,␣
↪4,5,6,7,8,9])

# Split the data into features and target variable
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[60]:
```
class FixedShareAlgorithm:
    def __init__(self, experts, alpha):
        self.experts = experts
        self.alpha = alpha
        self.weights = np.ones(len(experts)) / len(experts)

    def predict(self, X):
        predictions = [expert.predict(X) for expert in self.experts]
        return np.average(predictions, axis=0, weights=self.weights)

    def train(self, X, y, num_iterations=1):
```

```
        learner_losses = []
        expert_losses = []

        for iteration in range(num_iterations):
            predictions = [expert.predict(X) for expert in self.experts]
            learner_prediction = self.predict(X)
            learner_loss = np.mean((learner_prediction - y) ** 2)
            learner_losses.append(learner_loss)

            expert_loss = np.zeros(len(self.experts))
            for i, expert in enumerate(self.experts):
                expert_prediction = expert.predict(X)
                expert_loss[i] = np.mean((expert_prediction - y) ** 2)

            expert_losses.append(expert_loss)

            #self.weights = self.weights * (1 - self.alpha) ** expert_loss
            self.weights = self.alpha/5 + (1-6*self.alpha/5) * self.weights * np.
↪exp(-expert_loss)
            self.weights = self.weights / np.sum(self.weights)

        return learner_losses, expert_losses, self.weights
```

[61]:
```
class StaticExpertAlgorithm:
    def __init__(self, experts, weights):
        self.experts = experts
        self.weights = weights

    def predict(self, X):
        predictions = [expert.predict(X) for expert in self.experts]
        return np.average(predictions, axis=0, weights=self.weights)

    def train(self, X, y, num_iterations=1):
        learner_losses = []
        expert_losses = []

        for iteration in range(num_iterations):
            predictions = [expert.predict(X) for expert in self.experts]
            learner_prediction = self.predict(X)
            learner_loss = np.mean((learner_prediction - y) ** 2)
            learner_losses.append(learner_loss)

            expert_loss = np.zeros(len(self.experts))
            for i, expert in enumerate(self.experts):
                expert_prediction = expert.predict(X)
                expert_loss[i] = np.mean((expert_prediction - y) ** 2)
```

```
            expert_losses.append(expert_loss)

        return learner_losses, expert_losses, self.weights
```

[62]:
```
tree_expert1 = DecisionTreeRegressor(max_depth=5)
tree_expert2 = DecisionTreeRegressor(max_depth=10)
knn_expert = KNeighborsRegressor()
RFC=RandomForestRegressor()
LR=LinearRegression()
MLPR=MLPRegressor()

# Define the fixed-share algorithm with different values of alpha
alphas = [0.1, 0.5, 0.9]
fixed_share_algorithms = [FixedShareAlgorithm(experts=[tree_expert1,␣
 ↪tree_expert2, knn_expert, LR, RFC, MLPR], alpha=alpha) for alpha in alphas]

# Train the experts and the fixed-share algorithms
num_iterations = np.arange(10, 501, 5)
fixed_expert_losses_track = np.zeros((len(num_iterations),␣
 ↪len(fixed_share_algorithms), 6))
fixed_expert_weights_track = np.zeros((len(num_iterations),␣
 ↪len(fixed_share_algorithms), 6))
fixed_learner_losses_track = np.zeros((len(num_iterations),␣
 ↪len(fixed_share_algorithms)))
for i, n in enumerate(num_iterations):
    for expert in [tree_expert1, tree_expert2, knn_expert, LR, RFC, MLPR]:
        expert.fit(X_train[:n], y_train[:n])
    for j, fixed_share_algorithm in enumerate(fixed_share_algorithms):
        learner_loss, expert_losses, expert_weights = fixed_share_algorithm.
 ↪train(X_train[:n], y_train[:n], num_iterations=1)
        fixed_expert_losses_track[i, j, :] = np.array(expert_losses)
        fixed_expert_weights_track[i, j, :] = np.array(expert_weights)
        fixed_learner_losses_track[i, j] = np.reshape(learner_loss, (1,))
```

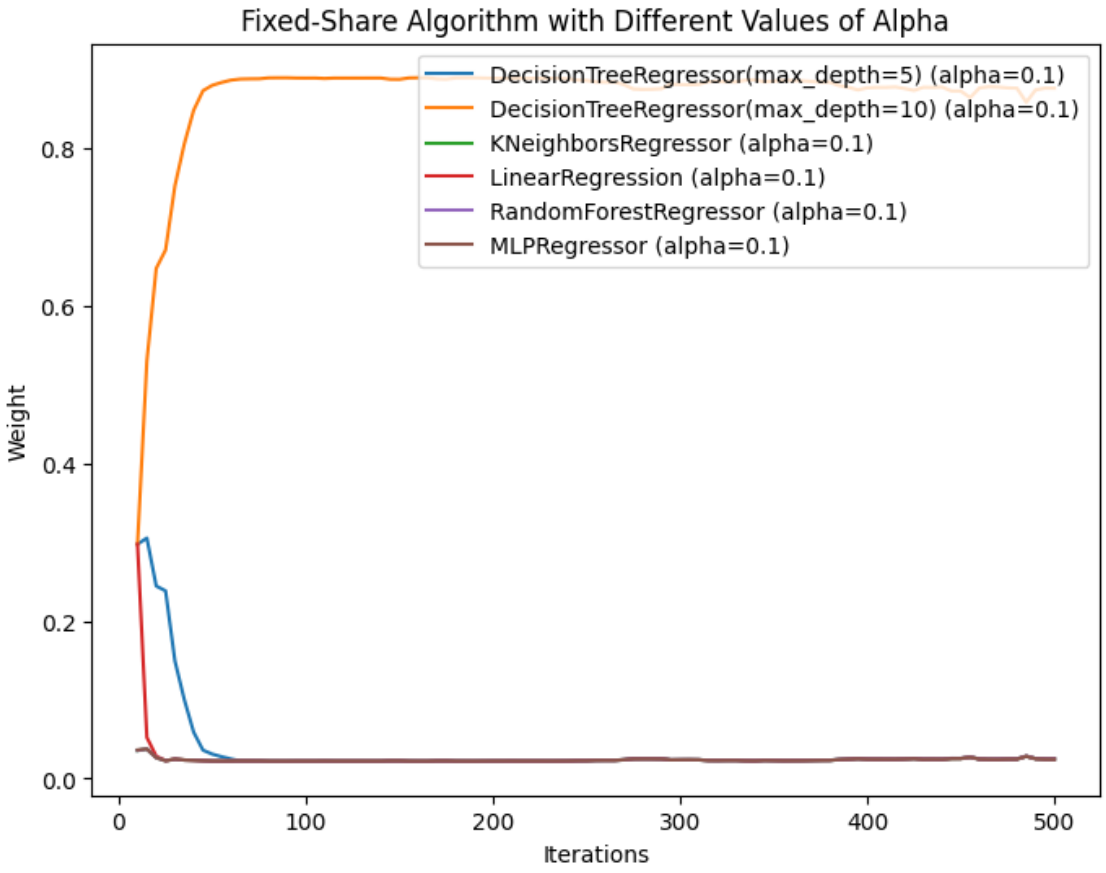
[63]:
```
# Print the cumulative loss of learner vs iteration in Fixed-Share (alpha)␣
 ↪algorithm
for j, alpha in enumerate(alphas):
    print(f"learner loss with (alpha={alpha})", fixed_learner_losses_track[:, j])
```

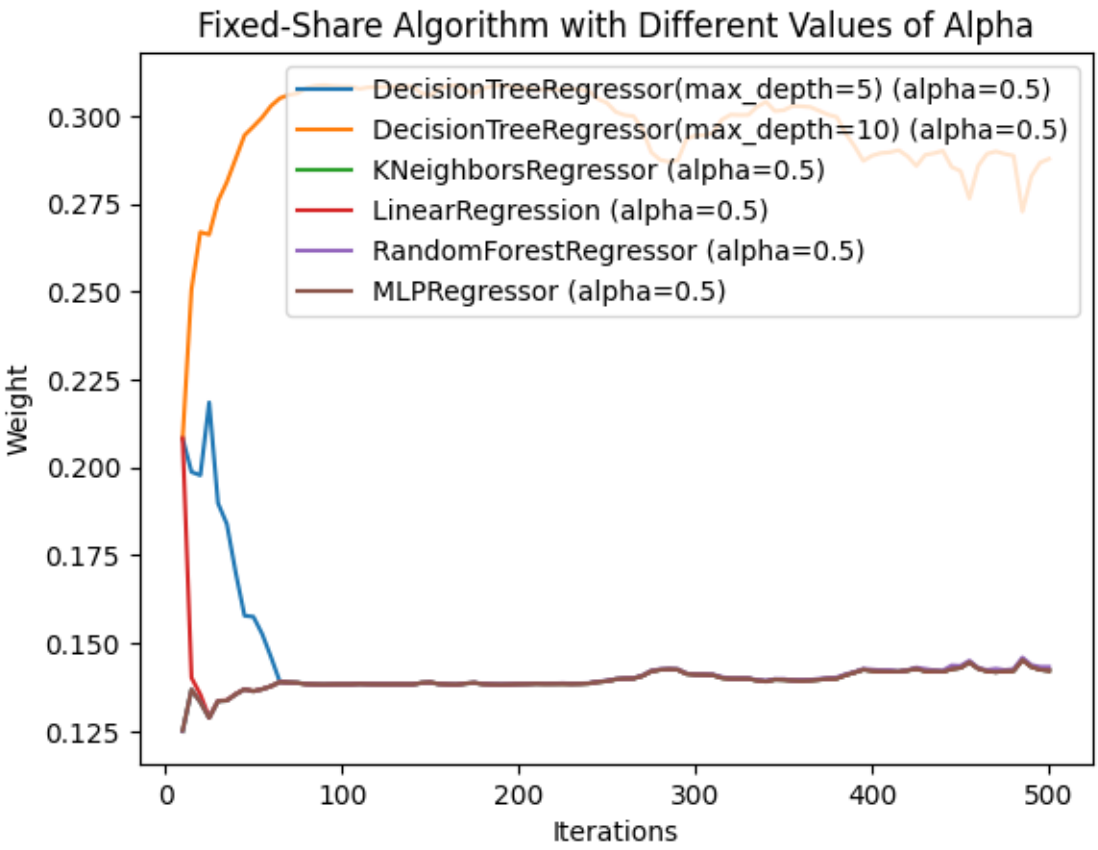
```
learner loss with (alpha=0.1) [2.03300168e+02 3.46465226e+00 1.76997160e+00
6.03253807e-01
 1.80861594e-01 1.63327616e-01 1.38980264e-01 1.69602296e-01
 1.62704823e-01 1.32246696e-01 1.61368836e-01 2.34683149e-01
 2.20567507e-01 1.87096607e-01 1.95760967e-01 2.18079497e-01
 1.81169459e-01 1.81793285e-01 1.95269406e-01 1.36980670e-01
 1.90689428e-01 1.70940143e-01 1.64897963e-01 1.39483051e-01
```

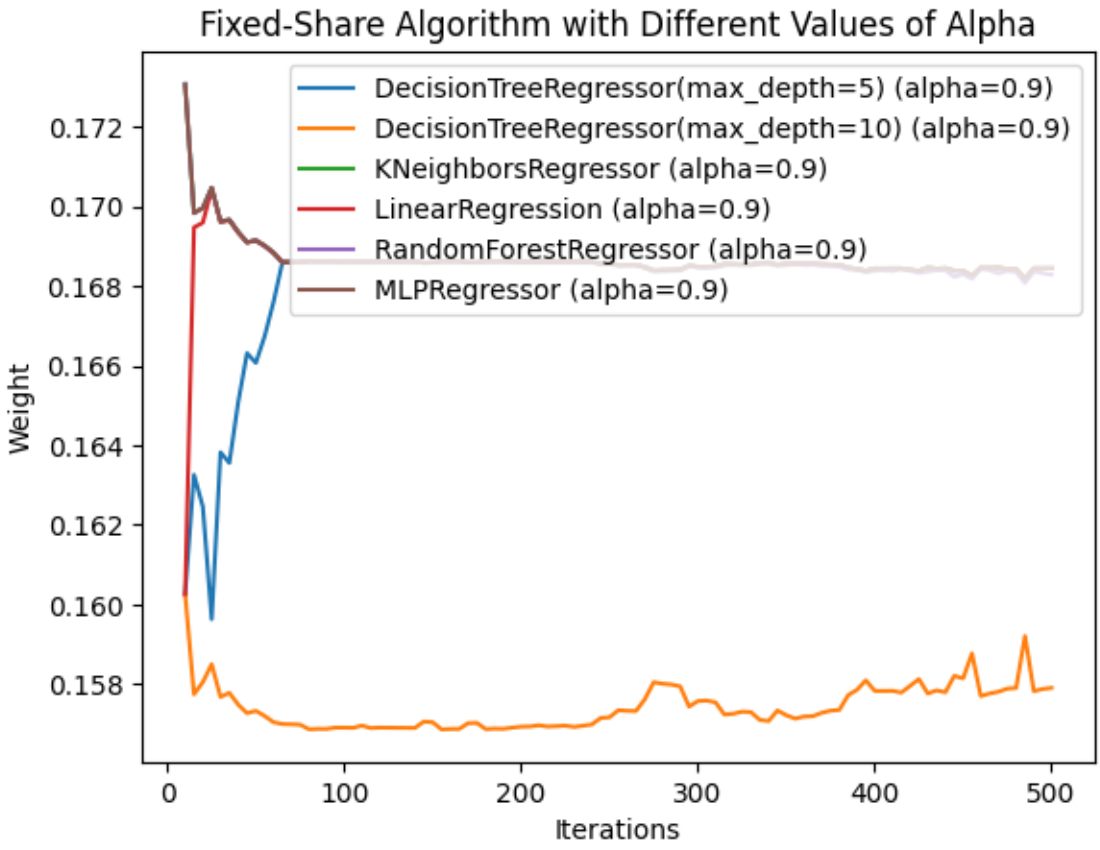
```
 1.47844009e-01 1.48179764e-01 1.77201471e-01 1.98788820e-01
 1.99631448e-01 2.00909638e-01 1.60684942e-01 1.77248780e-01
 1.81891048e-01 2.11892652e-01 1.73241445e-01 1.67679117e-01
 1.89202529e-01 1.93855953e-01 1.97501884e-01 1.76066002e-01
 1.77412077e-01 1.61165792e-01 1.55109409e-01 1.64473019e-01
 1.54345272e-01 1.63307575e-01 1.62491608e-01 1.80268860e-01
 2.21204702e-01 2.19810849e-01 2.17671651e-01 2.05332404e-01
 2.68887431e-01 2.91626743e-01 3.04393439e-01 3.22000628e-01
 2.91907423e-01 2.63416867e-01 2.56311522e-01 2.59775711e-01
 2.39467422e-01 2.12136172e-01 2.28219339e-01 2.26719669e-01
 2.58674125e-01 1.82434932e-01 1.79189657e-01 1.98808005e-01
 2.00553717e-01 2.08351630e-01 1.96378833e-01 1.86237951e-01
 1.92147601e-01 2.06924436e-01 2.11974136e-01 2.45555330e-01
 2.63872066e-01 2.93347105e-01 2.81743568e-01 2.65334834e-01
 2.59431819e-01 2.61582063e-01 2.69409513e-01 3.04947183e-01
 2.66353181e-01 2.52738036e-01 2.63278658e-01 3.12408069e-01
 3.20149718e-01 3.80590919e-01 3.01563094e-01 2.77255625e-01
 2.59415458e-01 2.68120267e-01 2.76102809e-01 4.08582612e-01
 3.20907163e-01 2.86950236e-01 2.70685427e-01]
learner loss with (alpha=0.5) [203.30016834  36.70196213  22.16680475
14.70176609   4.24944304
   4.15179562   4.37444718   5.78877603   5.89184627   4.92964212
   6.1171283    8.77770112   8.30471486   7.06435394   7.48971444
   8.52240261   7.07460527   7.00737344   7.57052293   5.22315064
   6.9923428    6.46777928   6.16509368   5.17481126   5.53347749
   5.52076881   6.63302212   6.96878709   6.85506548   7.59808063
   6.23238913   6.8678778    6.37464442   7.33817323   6.54669517
   6.50087246   7.34830805   7.41231877   7.43735353   6.56471414
   6.46145387   5.92998686   5.71748924   6.0491519    5.77029544
   6.11871923   5.93298226   5.90799646   7.13066798   6.48843529
   6.33971981   5.90242089   6.86623467   6.64291975   6.77729141
   7.32128342   6.36026398   6.8031864    6.57359446   6.54277974
   5.92798759   6.18317223   7.06546476   6.84203278   7.91316662
   6.0005001    6.25088972   6.04719232   6.24725925   6.93301205
   6.31168985   5.85579601   5.76138618   5.91407267   5.84068739
   5.67249652   5.67392859   5.87341095   5.94720275   5.76859988
   5.5563564    5.78876079   5.59784188   5.91742247   5.73155717
   5.45774372   5.74852293   5.90082102   6.04038274   5.94764793
   6.14188991   6.27309371   5.77968197   5.72633215   5.8131618
   5.84328644   5.88704129   6.13801445   5.63203302]
learner loss with (alpha=0.9) [203.30016834  69.95379155  34.09352397
23.86784869   7.36573374
   6.64093529   7.06720885   9.07888524   9.00456635   7.5852656
   9.32472221  13.12163808  12.23235869  10.43575406  11.06581939
  12.65805781  10.52150462  10.4283355   11.26013002   7.76237266
  10.38363296   9.59126841   9.14973199   7.6808578    8.21640574
   8.19639416   9.8494678   10.34700695  10.1186635   11.20601003
   9.24589117  10.21006221   9.471017    10.85005845   9.66949284
```

```
 9.64956366  10.92422412  11.02719278  11.05409295   9.74459952
 9.58192067   8.78278359   8.47510764   8.96584171   8.55106471
 9.07802673   8.7944827    8.73426378  10.47097877   9.49297404
 9.20704683   8.54924378   9.91351845   9.51086616   9.51915743
10.2203663    8.85695643   9.48799772   9.32465659   9.30205262
 8.42821003   8.82501442  10.20580135   9.91779852  11.46582141
 8.70804268   9.14119192   8.86871133   9.08991498  10.10296875
 9.22111975   8.55355994   8.4061879    8.59469316   8.45243916
 8.17818232   8.07013007   8.26910092   8.27289814   8.06768059
 7.78274265   8.11866264   7.85840782   8.25457673   7.94630428
 7.64385255   8.05028919   8.26034553   8.36011624   8.18329607
 8.28936907   8.70057428   8.09661367   8.04093618   8.13480959
 8.13948688   7.85562411   8.4442317    7.83120453]
```

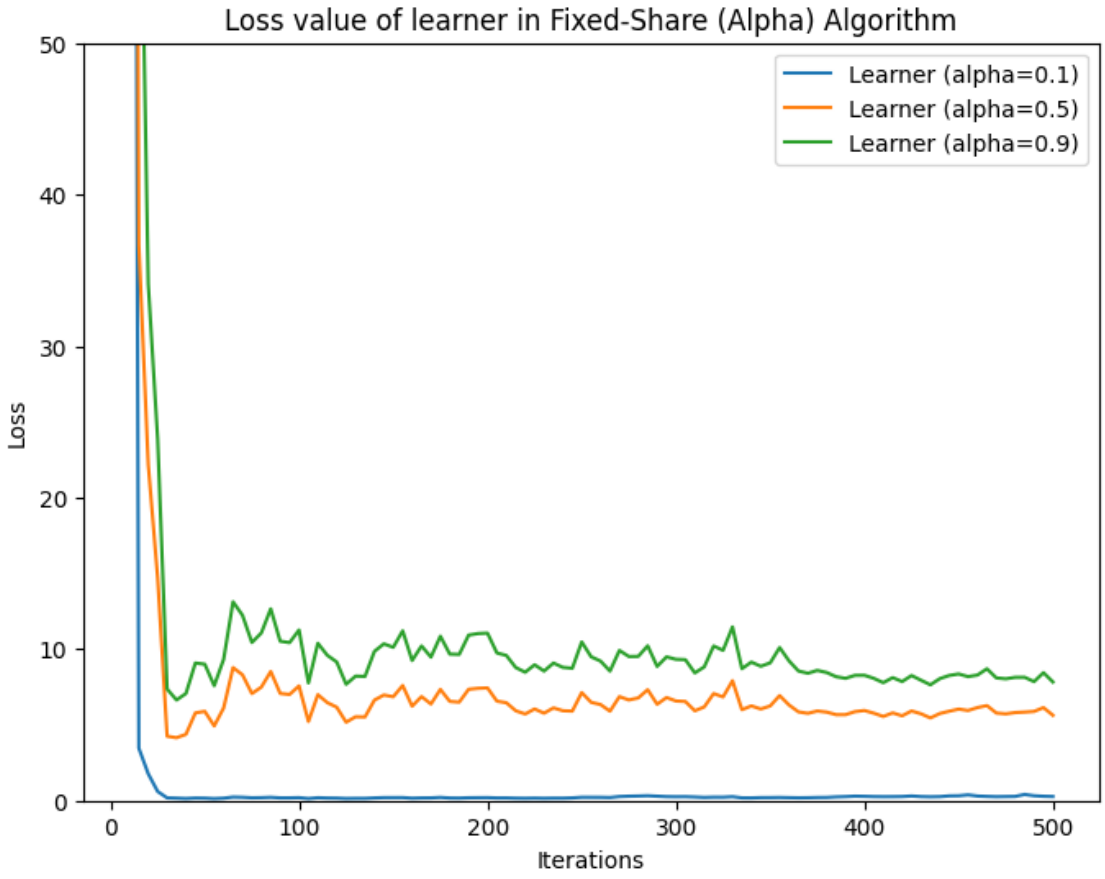
[80]:
```
# Plot the experts weight of Fixed-Share Algorithm with Different Values of Alpha
plt.figure(figsize=(8, 6))
for j, alpha in enumerate(alphas):
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 0],␣
↪label=f"DecisionTreeRegressor(max_depth=5) (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 1],␣
↪label=f"DecisionTreeRegressor(max_depth=10) (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 2],␣
↪label=f"KNeighborsRegressor (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 3],␣
↪label=f"LinearRegression (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 4],␣
↪label=f"RandomForestRegressor (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 5],␣
↪label=f"MLPRegressor (alpha={alpha})")
    plt.xlabel("Iterations")
    plt.ylabel("Weight")
    plt.title("Fixed-Share Algorithm with Different Values of Alpha")
    plt.legend(loc="upper right")
    plt.show()
```


Fixed-Share Algorithm with Different Values of Alpha
DecisionTreeRegressor(max_depth=5) (alpha=0.1)
DecisionTreeRegressor(max_depth=10) (alpha=0.1)
KNeighborsRegressor (alpha=0.1)
LinearRegression (alpha=0.1)
RandomForestRegressor (alpha=0.1)
MLPRegressor (alpha=0.1)
Weight
0.0
0.2
0.4
0.6
0.8
0
100
200
300
400
500
Iterations


Fixed-Share Algorithm with Different Values of Alpha
DecisionTreeRegressor(max_depth=5) (alpha=0.5)
DecisionTreeRegressor(max_depth=10) (alpha=0.5)
KNeighborsRegressor (alpha=0.5)
LinearRegression (alpha=0.5)
RandomForestRegressor (alpha=0.5)
MLPRegressor (alpha=0.5)
Weight
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0
100
200
300
400
500
Iterations

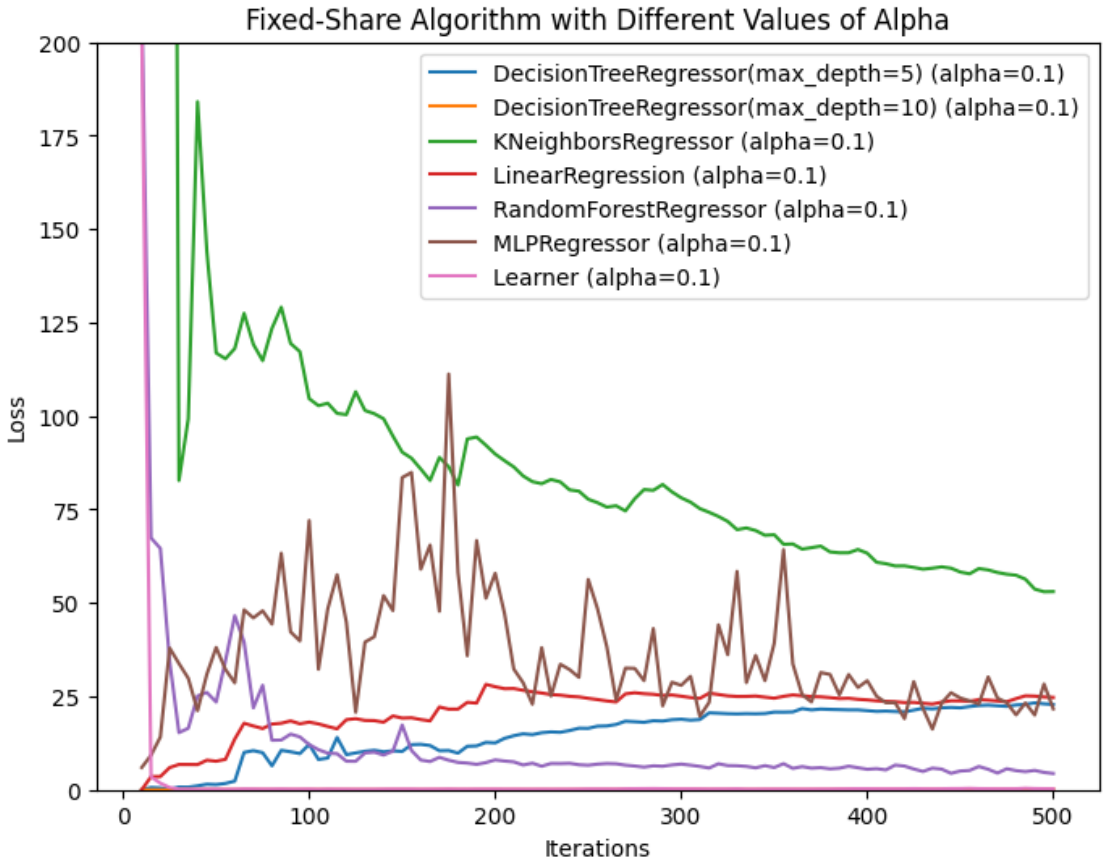


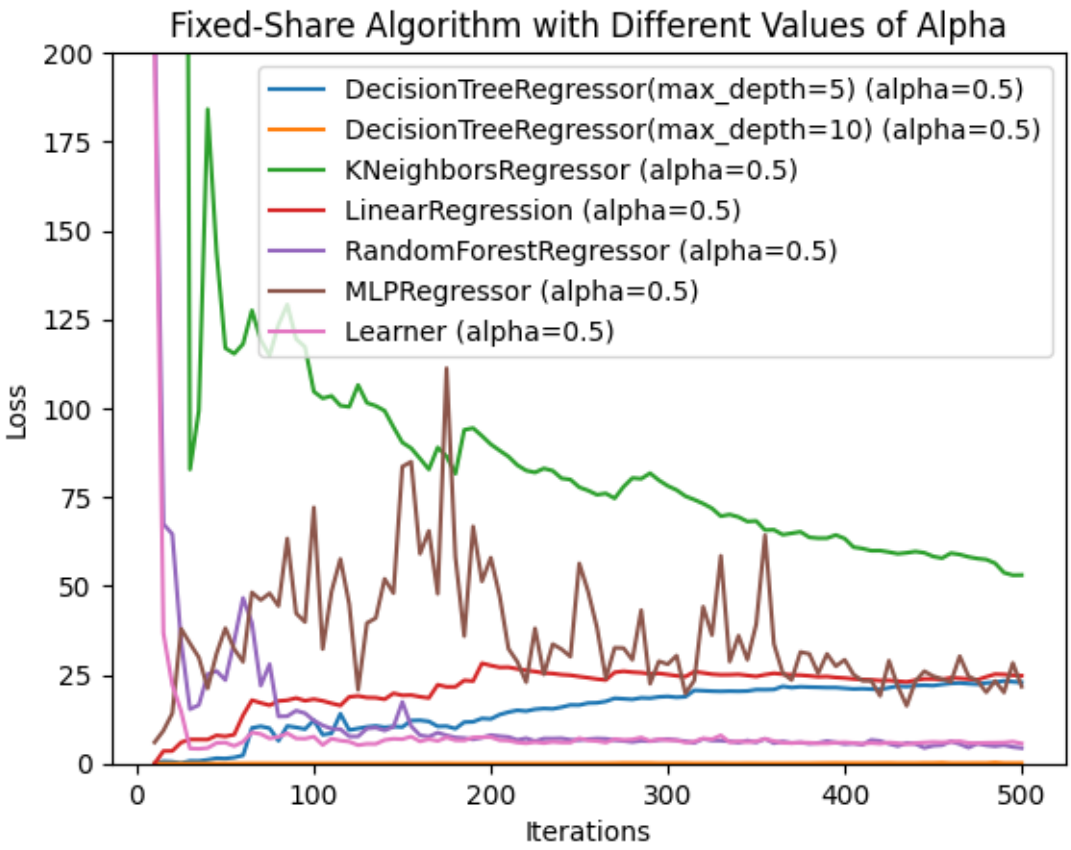
[81]:
```
# Plot the loss of learners in Fixed-Share Algorithm with Different Values of␣
 ↪Alpha
plt.figure(figsize=(8, 6))
for j, alpha in enumerate(alphas):
    plt.plot(num_iterations, fixed_learner_losses_track[:, j], label=f"Learner␣
 ↪(alpha={alpha})")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.ylim(0,50)
plt.title("Loss value of learner in Fixed-Share (Alpha) Algorithm")
plt.legend(loc="upper right")
plt.show()
```

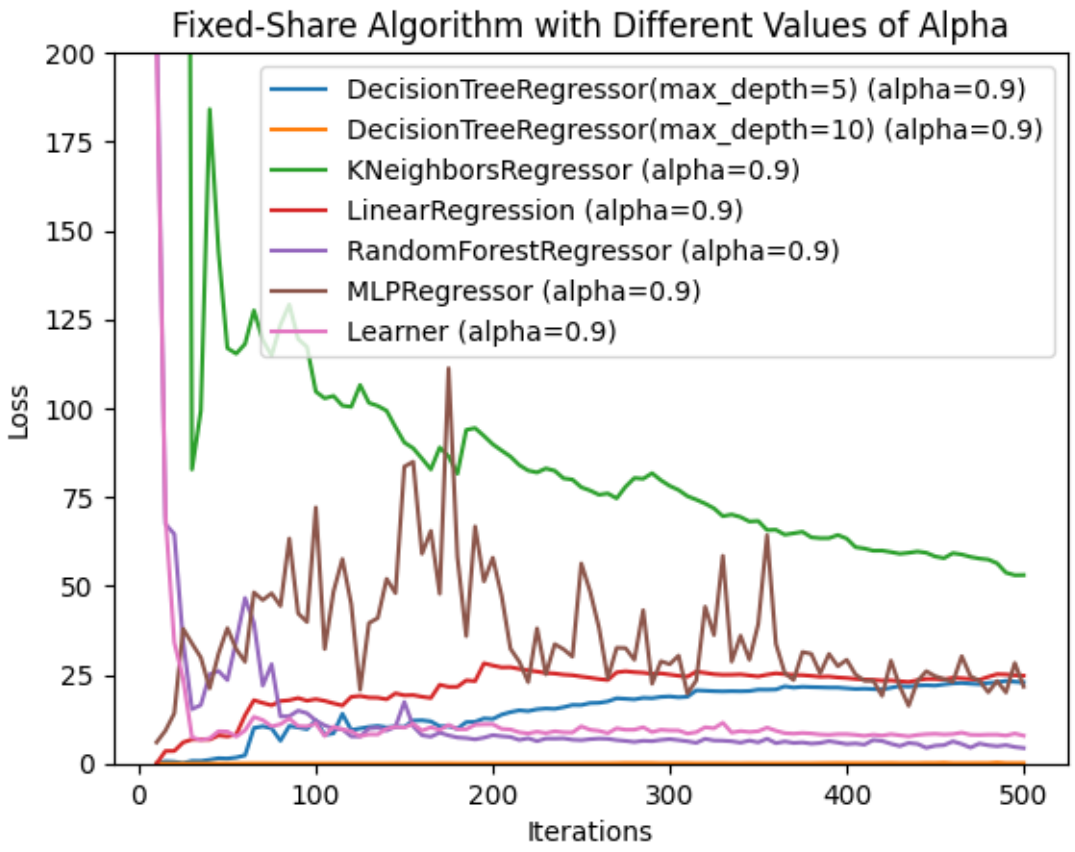


[82]:
```
# Plot the loss values of Fixed-Share Algorithm with Different Values of Alpha
plt.figure(figsize=(8, 6))
for j, alpha in enumerate(alphas):
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 0],␣
↪label=f"DecisionTreeRegressor(max_depth=5) (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 1],␣
↪label=f"DecisionTreeRegressor(max_depth=10) (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 2],␣
↪label=f"KNeighborsRegressor (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 3],␣
↪label=f"LinearRegression (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 4],␣
↪label=f"RandomForestRegressor (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 5],␣
↪label=f"MLPRegressor (alpha={alpha})")
    plt.plot(num_iterations, fixed_learner_losses_track[:, j], label=f"Learner␣
↪(alpha={alpha})")
    plt.xlabel("Iterations")
```

```
    plt.ylabel("Loss")
    plt.ylim(0,200)
    plt.title("Fixed-Share Algorithm with Different Values of Alpha")
    plt.legend(loc="upper right")
    plt.show()
```




Fixed-Share Algorithm with Different Values of Alpha
DecisionTreeRegressor(max_depth=5) (alpha=0.5)
DecisionTreeRegressor(max_depth=10) (alpha=0.5)
KNeighborsRegressor (alpha=0.5)
LinearRegression (alpha=0.5)
RandomForestRegressor (alpha=0.5)
MLPRegressor (alpha=0.5)
Learner (alpha=0.5)
Loss
Iterations
0
25
50
75
100
125
150
175
200
0
100
200
300
400
500



[75]:
```
# Define the experts
tree_expert1 = DecisionTreeRegressor(max_depth=5)
tree_expert2 = DecisionTreeRegressor(max_depth=10)
knn_expert = KNeighborsRegressor()
RFC=RandomForestRegressor()
LR=LinearRegression()
MLPR=MLPRegressor()

# Define the expert weights
weights = [0.2, 0.2, 0.2, 0.2, 0.1, 0.1]

# Define the static expert algorithm
static_expert_algorithm = StaticExpertAlgorithm(experts=[tree_expert1,␣
 ↪tree_expert2, knn_expert, LR, RFC, MLPR], weights=weights)

# Train the experts and the static expert algorithm
num_iterations = np.arange(10, 501, 5)
static_expert_losses_track = np.zeros((len(num_iterations), 6))
static_learner_losses_track = np.zeros((len(num_iterations)))
```

```
for i, n in enumerate(num_iterations):
    for expert in [tree_expert1, tree_expert2, knn_expert, LR, RFC, MLPR]:
        expert.fit(X_train[:n], y_train[:n])
    learner_loss, expert_losses, expert_weights = static_expert_algorithm.
 ↪train(X_train[:n], y_train[:n], num_iterations=1)
    static_expert_losses_track[i, :] = np.array(expert_losses)
    static_learner_losses_track[i] = np.reshape(learner_loss, (1,))
```

[76]:
```
# Print the cumulative loss of learner vs iteration in Static Expert algorithm

print("cumulative loss of learner vs iteration in Static Expert algorithm:",␣
 ↪static_learner_losses_track)
```

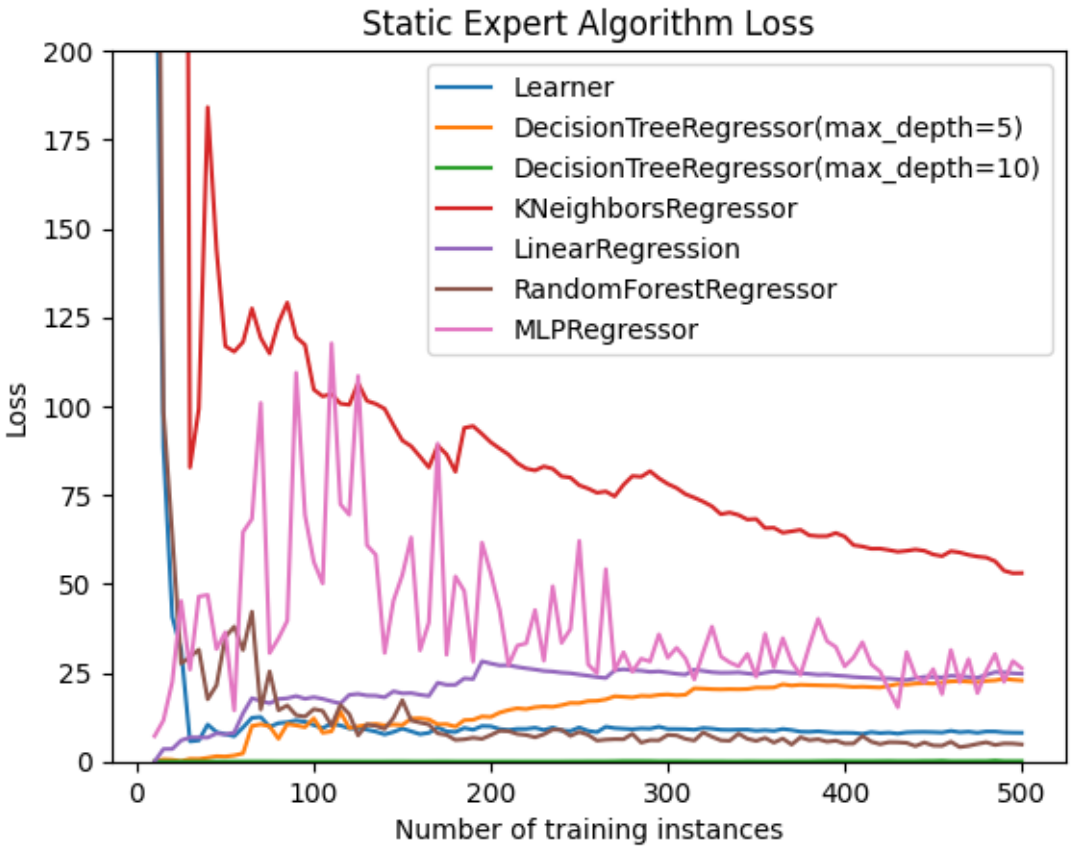
```
cumulative loss of learner vs iteration in Static Expert algorithm:
[263.57580263  88.50811274  40.66832864  31.52445652   5.67551123
   5.93997299  10.40795669   8.27481943   7.69300059   7.13332767
   9.59859897  12.30404141  12.52293349   9.77906178  10.79270045
  11.03894779  11.55062158  11.19078198  10.26743065   9.40174884
  10.72923329  10.16487469   9.33351747   9.31091179   8.96709436
   8.4179711    7.6340208    8.32957841   9.30251866   8.50035657
   7.69678316   8.07831579   9.62548105   8.46904996   8.31026896
   9.44046307   8.94726217   9.98559883   9.9027155    9.14237769
   8.87813913   9.18459476   9.18130128   9.50548839   8.87862706
   9.55493099   8.76876571   8.59333504   9.58559696   8.62254057
   8.27150484   9.79680665   9.39670052   9.24833875   9.09552208
   9.32268892   9.24178114   9.79806857   9.16933204   9.15308577
   9.11784611   8.9402255    9.43707747   9.26185155   8.94279571
   8.89115462   8.95660795   8.98240294   8.68123843   8.84079982
   8.62256749   9.1652497    8.85910335   8.3955962    8.34726833
   8.5079852    8.28658512   8.05498332   8.09364833   8.03841256
   7.77409953   8.2779962    7.99730671   8.13806158   7.85082446
   8.20554474   8.30217036   8.32274414   8.26381986   8.15854129
   8.63010366   8.23404882   8.40609992   8.08592053   8.45453215
   8.43954097   8.19385769   8.12234044   8.092085  ]
```

[83]:
```
# Plot learner loss
experts=["DecisionTreeRegressor(max_depth=5)",␣
 ↪"DecisionTreeRegressor(max_depth=10)", "KNeighborsRegressor",
         "LinearRegression", "RandomForestRegressor", "MLPRegressor"]
plt.plot(num_iterations, static_learner_losses_track, label='Learner')

# Plot expert losses
for i, expert in enumerate(experts):
    plt.plot(num_iterations, static_expert_losses_track[:, i], label=f'{expert}')

# Add labels and legend
plt.xlabel('Number of training instances')
```

```
plt.ylabel('Loss')
plt.ylim(0,200)
plt.title('Static Expert Algorithm Loss')
plt.legend()
plt.show()
```



[ ]:

Spambase dataset

February 20, 2023

[65]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
import warnings
#suppress warnings
warnings.filterwarnings('ignore')
# Load the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/
 ↪spambase.data"
data = pd.read_csv(url, header=None)
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Normalize the data
X_train = (X_train - np.mean(X_train, axis=0)) / np.std(X_train, axis=0)
```

[66]:
```
class FixedShareAlgorithm:
    def __init__(self, experts, alpha):
        self.experts = experts
        self.alpha = alpha
        self.weights = np.ones(len(experts)) / len(experts)

    def predict(self, X):
        predictions = [expert.predict(X) for expert in self.experts]
```

```
        return np.average(predictions, axis=0, weights=self.weights)

    def train(self, X, y, num_iterations=1):
        learner_losses = []
        expert_losses = []

        for iteration in range(num_iterations):
            predictions = [expert.predict(X) for expert in self.experts]
            learner_prediction = self.predict(X)
            learner_loss = np.mean((learner_prediction - y) ** 2)
            learner_losses.append(learner_loss)

            expert_loss = np.zeros(len(self.experts))
            for i, expert in enumerate(self.experts):
                expert_prediction = expert.predict(X)
                expert_loss[i] = np.mean((expert_prediction - y) ** 2)

            expert_losses.append(expert_loss)

            #self.weights = self.weights * (1 - self.alpha) ** expert_loss
            self.weights = self.alpha/5 + (1-6*self.alpha/5) * self.weights * np.
↪exp(-expert_loss)
            self.weights = self.weights / np.sum(self.weights)

        return learner_losses, expert_losses, self.weights
```

[67]:
```
class StaticExpertAlgorithm:
    def __init__(self, experts, weights):
        self.experts = experts
        self.weights = weights

    def predict(self, X):
        predictions = [expert.predict(X) for expert in self.experts]
        return np.average(predictions, axis=0, weights=self.weights)

    def train(self, X, y, num_iterations=1):
        learner_losses = []
        expert_losses = []

        for iteration in range(num_iterations):
            predictions = [expert.predict(X) for expert in self.experts]
            learner_prediction = self.predict(X)
            learner_loss = np.mean((learner_prediction - y) ** 2)
            learner_losses.append(learner_loss)

            expert_loss = np.zeros(len(self.experts))
            for i, expert in enumerate(self.experts):
```

```
                expert_prediction = expert.predict(X)
                expert_loss[i] = np.mean((expert_prediction - y) ** 2)

            expert_losses.append(expert_loss)

        return learner_losses, expert_losses, self.weights
```

[68]:
```
# Define the experts
tree_expert1 = DecisionTreeClassifier(max_depth=10)
tree_expert2 = DecisionTreeClassifier(max_depth=20)
knn_expert = KNeighborsClassifier()
GBC=GradientBoostingClassifier()
RFC=RandomForestClassifier()
MLPR=MLPClassifier()
# Define the fixed-share algorithm with different values of alpha
alphas = [0.05, 0.1, 0.5]
fixed_share_algorithms = [FixedShareAlgorithm(experts=[tree_expert1,␣
 ↪tree_expert2, knn_expert, GBC, RFC, MLPR], alpha=alpha) for alpha in alphas]

# Train the experts and the fixed-share algorithms
num_iterations = np.arange(10, 501, 5)
fixed_expert_losses_track = np.zeros((len(num_iterations),␣
 ↪len(fixed_share_algorithms), 6))
fixed_expert_weights_track = np.zeros((len(num_iterations),␣
 ↪len(fixed_share_algorithms), 6))
fixed_learner_losses_track = np.zeros((len(num_iterations),␣
 ↪len(fixed_share_algorithms)))
for i, n in enumerate(num_iterations):
    for expert in [tree_expert1, tree_expert2, knn_expert, GBC, RFC, MLPR]:
        expert.fit(X_train[:n], y_train[:n])
    for j, fixed_share_algorithm in enumerate(fixed_share_algorithms):
        learner_loss, expert_losses, expert_weights = fixed_share_algorithm.
 ↪train(X_train[:n], y_train[:n], num_iterations=1)
        fixed_expert_losses_track[i, j, :] = np.array(expert_losses)
        fixed_expert_weights_track[i, j, :] = np.array(expert_weights)
        fixed_learner_losses_track[i, j] = np.reshape(learner_loss, (1,))
```

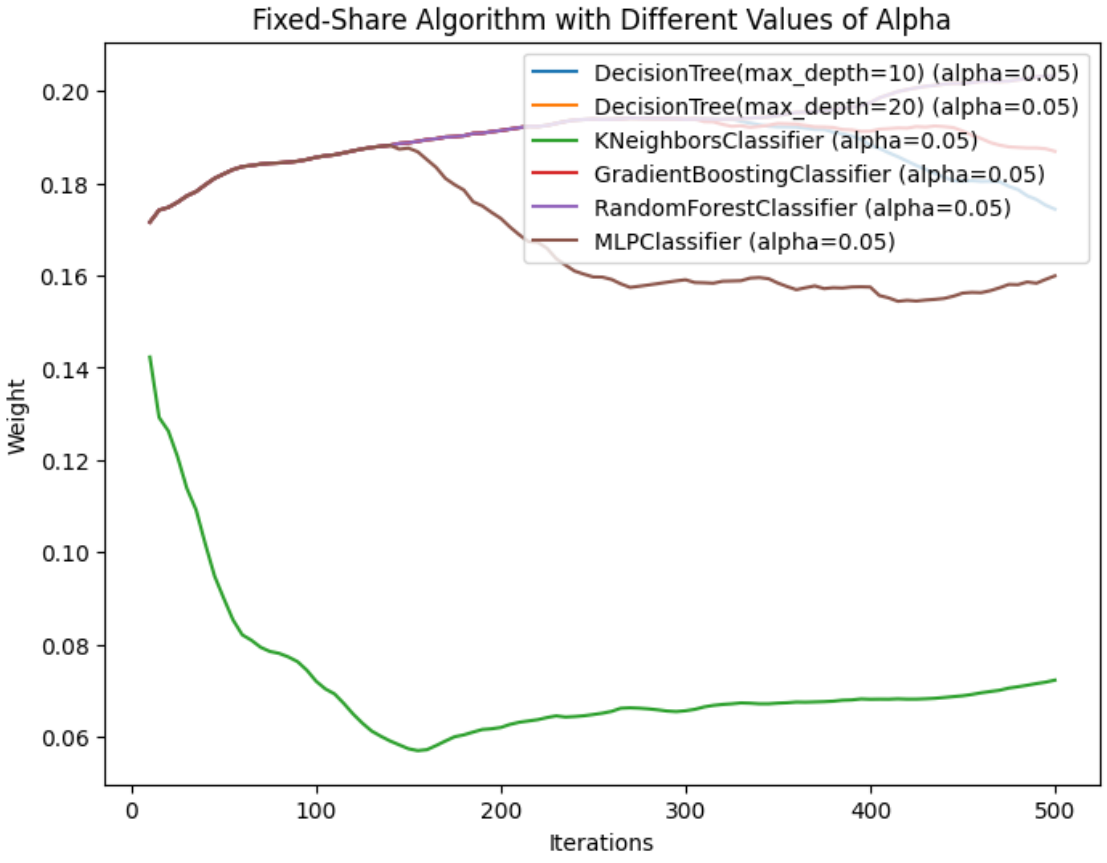
[69]:
```
# Print the cumulative loss of learner vs iteration in Fixed-Share (alpha)␣
 ↪algorithm
for j, alpha in enumerate(alphas):
    print(f"learner loss with (alpha={alpha})", fixed_learner_losses_track[:, j])
```

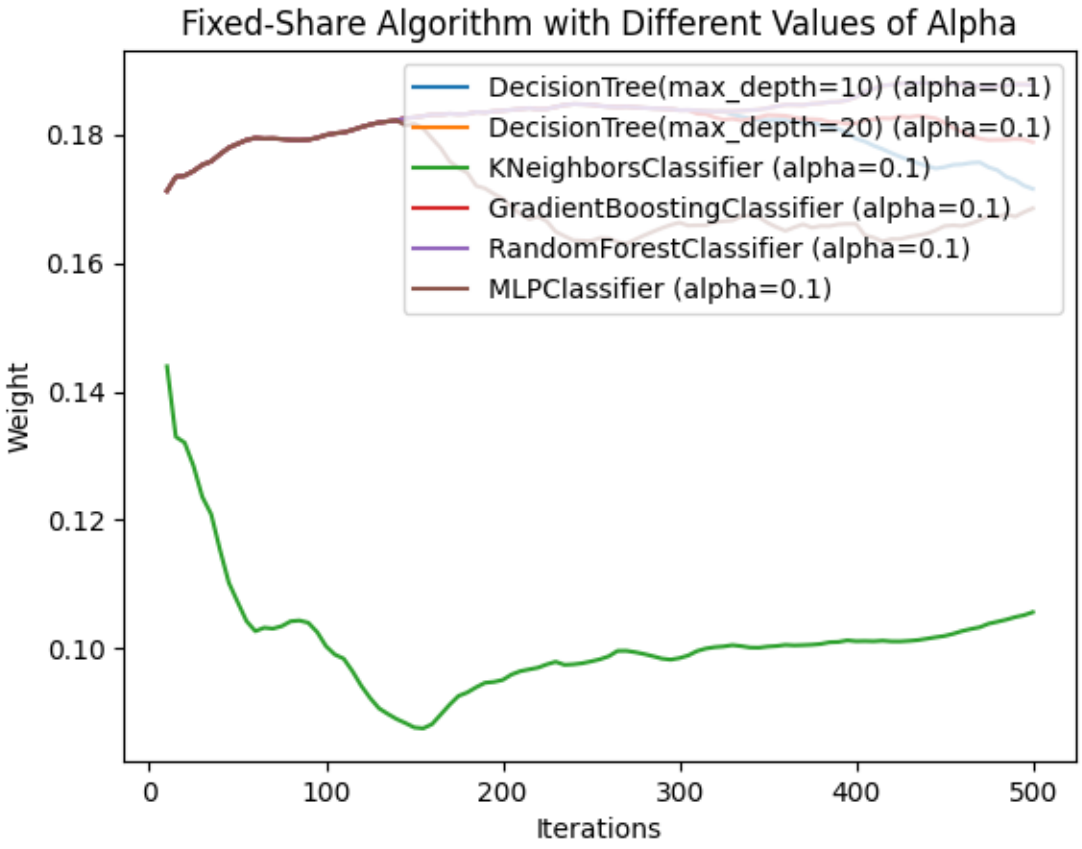
```
learner loss with (alpha=0.05) [0.00555556 0.00270026 0.00083503 0.00127614
0.00145676 0.00111174
 0.00149115 0.00138264 0.0010809  0.00103167 0.00085049 0.00062289
 0.00065545 0.00058893 0.00053972 0.00057443 0.00059831 0.00067441
 0.00072114 0.00064346 0.00058623 0.00066888 0.000679   0.00064234
```

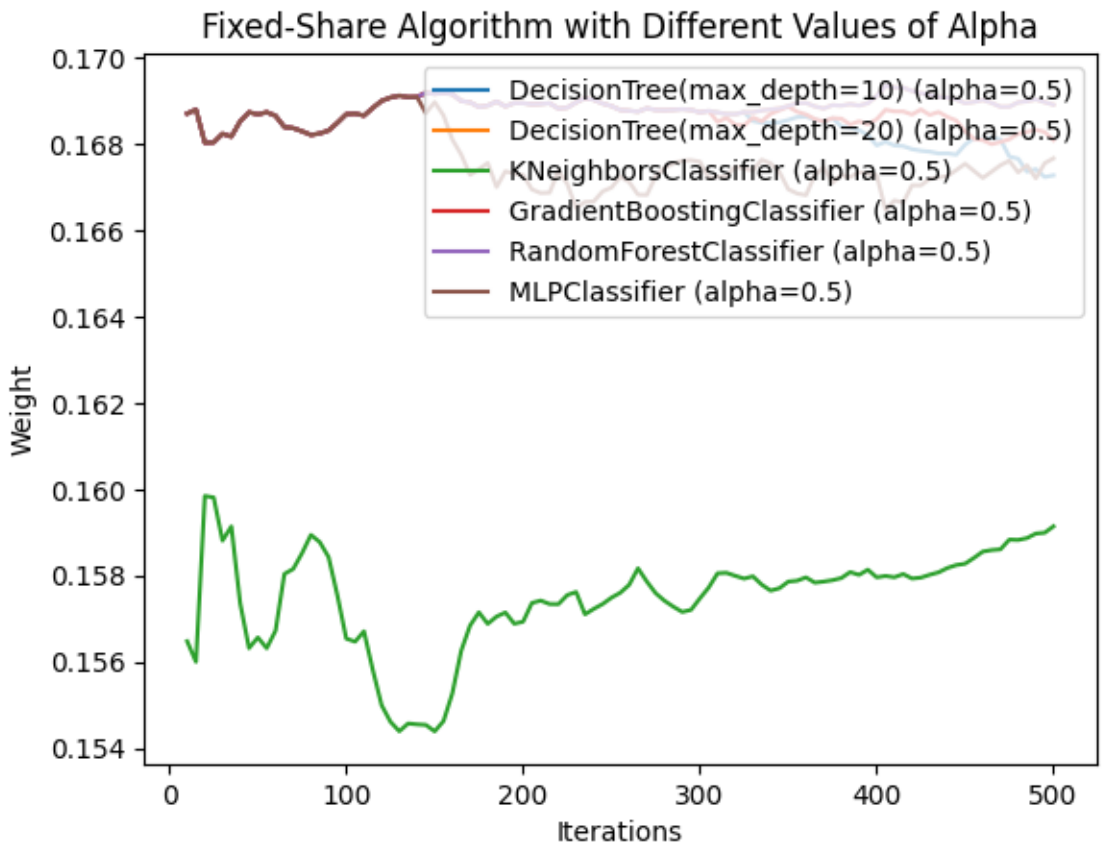
```
 0.00061125 0.00055706 0.00054327 0.00092869 0.00052128 0.00085609
 0.00115023 0.00106962 0.00136797 0.00101893 0.00103936 0.00147383
 0.0009841  0.00127095 0.00123243 0.00142946 0.00130903 0.0013916
 0.00090575 0.00132772 0.00164725 0.0014443  0.00137906 0.00115148
 0.00121001 0.00100584 0.0011695  0.00123068 0.00126125 0.00099414
 0.00099181 0.00098827 0.00098371 0.00096373 0.00093256 0.00122425
 0.00144559 0.00104991 0.00129705 0.00143862 0.00135805 0.00188492
 0.00136222 0.00153788 0.00171948 0.00226055 0.00222278 0.00188201
 0.00185715 0.00215938 0.00203689 0.00245142 0.00199853 0.00299313
 0.00319932 0.00364274 0.00297608 0.00359975 0.0031436  0.00274932
 0.00318593 0.00270118 0.00310556 0.00250221 0.00270529 0.00291467
 0.00321352 0.002902   0.0025547  0.00245972 0.0026178  0.0034543
 0.00361454 0.00327511 0.0033714 ]
learner loss with (alpha=0.1) [0.00555556 0.00276121 0.00088354 0.0013945
0.00164862 0.00130875
 0.00182669 0.00177321 0.00145759 0.00146245 0.00126753 0.00097286
 0.00106499 0.00099138 0.00093641 0.00102196 0.0010883  0.00125178
 0.00136633 0.0012461  0.00115879 0.00134662 0.00139437 0.00134705
 0.00130891 0.00121725 0.00120878 0.00165362 0.00119674 0.00155869
 0.00185858 0.00171592 0.00206811 0.00167094 0.00174641 0.00221176
 0.00167326 0.00205483 0.00200034 0.00220875 0.00205572 0.00220593
 0.00160921 0.00211881 0.00250342 0.00231041 0.00220185 0.00191343
 0.00200675 0.00174448 0.00194903 0.00198825 0.00207749 0.00176227
 0.00176442 0.00176411 0.00176184 0.00172521 0.00166269 0.00203931
 0.00221019 0.00180741 0.00203974 0.00223403 0.00210546 0.00264586
 0.00213643 0.0023753  0.00259333 0.003137   0.00308812 0.00266611
 0.00263067 0.00302429 0.00283596 0.0032991  0.00279743 0.0038094
 0.00409808 0.00469677 0.003873   0.00456587 0.00403814 0.00366871
 0.00406987 0.00357695 0.00398111 0.0033228  0.0035293  0.00376664
 0.00404255 0.00370319 0.00333754 0.0032248  0.0034247  0.00432175
 0.00450634 0.00410859 0.00417844]
learner loss with (alpha=0.5) [0.00555556 0.00326451 0.00121672 0.00204394
0.00255381 0.00216165
 0.00316548 0.00330126 0.00293201 0.00311964 0.00285051 0.00226713
 0.00249745 0.00233437 0.00219864 0.00237755 0.00252084 0.00290604
 0.00322612 0.00303368 0.00289316 0.00341643 0.00364142 0.00365121
 0.00367732 0.00353106 0.00358355 0.00418143 0.00366163 0.00405745
 0.00429487 0.00389744 0.00429406 0.00373064 0.00391421 0.00438622
 0.00370147 0.00427929 0.00416067 0.00433333 0.00411146 0.00437671
 0.00354351 0.00418476 0.00467833 0.00454907 0.00433062 0.00392032
 0.00405853 0.0036701  0.00391606 0.00385942 0.0040767  0.00368713
 0.00370258 0.00371909 0.00373553 0.00366783 0.00352631 0.00400852
 0.00405474 0.00364331 0.00384391 0.00410931 0.00388469 0.00447523
 0.00398587 0.00430979 0.00455926 0.00511611 0.0050444  0.00452863
 0.00446608 0.00498142 0.00470217 0.00521265 0.00465297 0.0056484
 0.00610422 0.00688502 0.00586648 0.00665201 0.00605212 0.00571242
 0.00604288 0.00552966 0.00591229 0.00516943 0.00536091 0.00560662
 0.00580023 0.00543936 0.00503121 0.0048681  0.00510531 0.00606922
```

```
0.00623644 0.00578433 0.00579111]
```

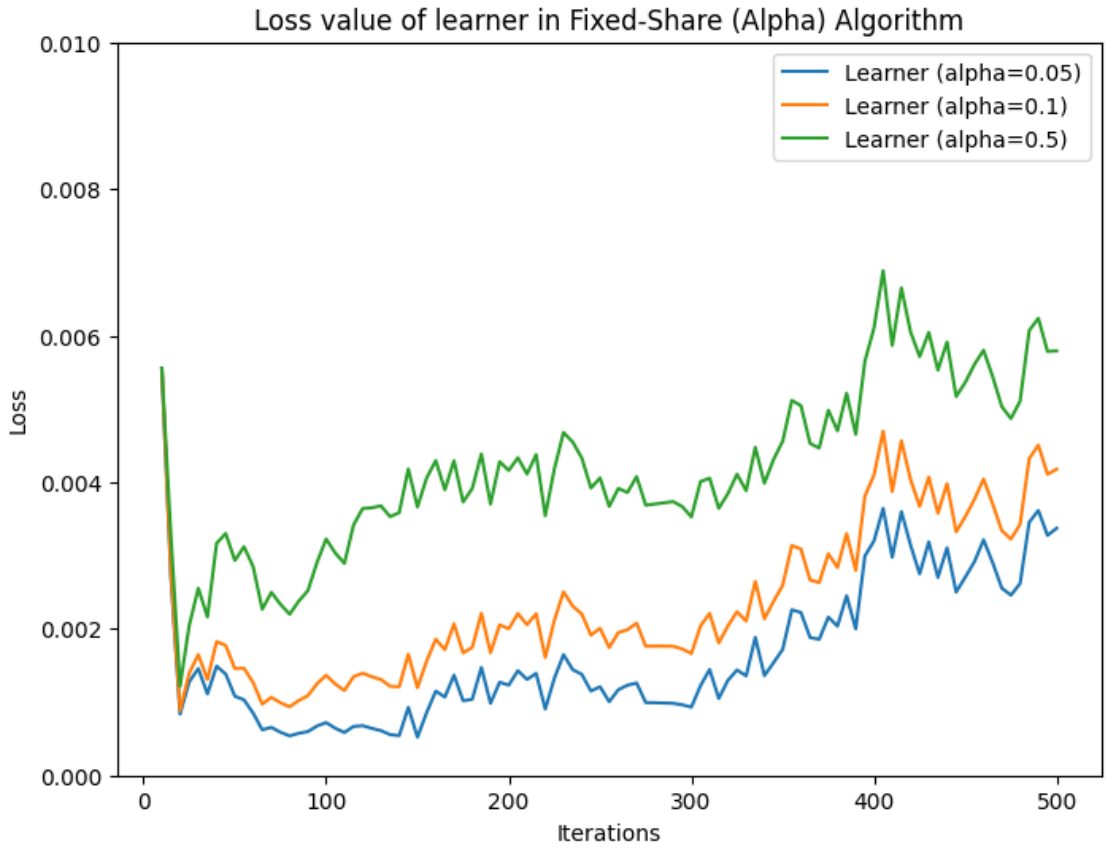
[79]:
```
# Plot the experts weight of Fixed-Share Algorithm with Different Values of Alpha
plt.figure(figsize=(8, 6))
for j, alpha in enumerate(alphas):
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 0],␣
 ↪label=f"DecisionTree(max_depth=10) (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 1],␣
 ↪label=f"DecisionTree(max_depth=20) (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 2],␣
 ↪label=f"KNeighborsClassifier (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 3],␣
 ↪label=f"GradientBoostingClassifier (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 4],␣
 ↪label=f"RandomForestClassifier (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_weights_track[:, j, 5],␣
 ↪label=f"MLPClassifier (alpha={alpha})")
    plt.xlabel("Iterations")
    plt.ylabel("Weight")
    plt.title("Fixed-Share Algorithm with Different Values of Alpha")
    plt.legend(loc="upper right")
    plt.show()
```




Fixed-Share Algorithm with Different Values of Alpha
DecisionTree(max_depth=10) (alpha=0.1)
DecisionTree(max_depth=20) (alpha=0.1)
KNeighborsClassifier (alpha=0.1)
GradientBoostingClassifier (alpha=0.1)
RandomForestClassifier (alpha=0.1)
MLPClassifier (alpha=0.1)
Weight
0.18
0.16
0.14
0.12
0.10
0
100
200
300
400
500
Iterations

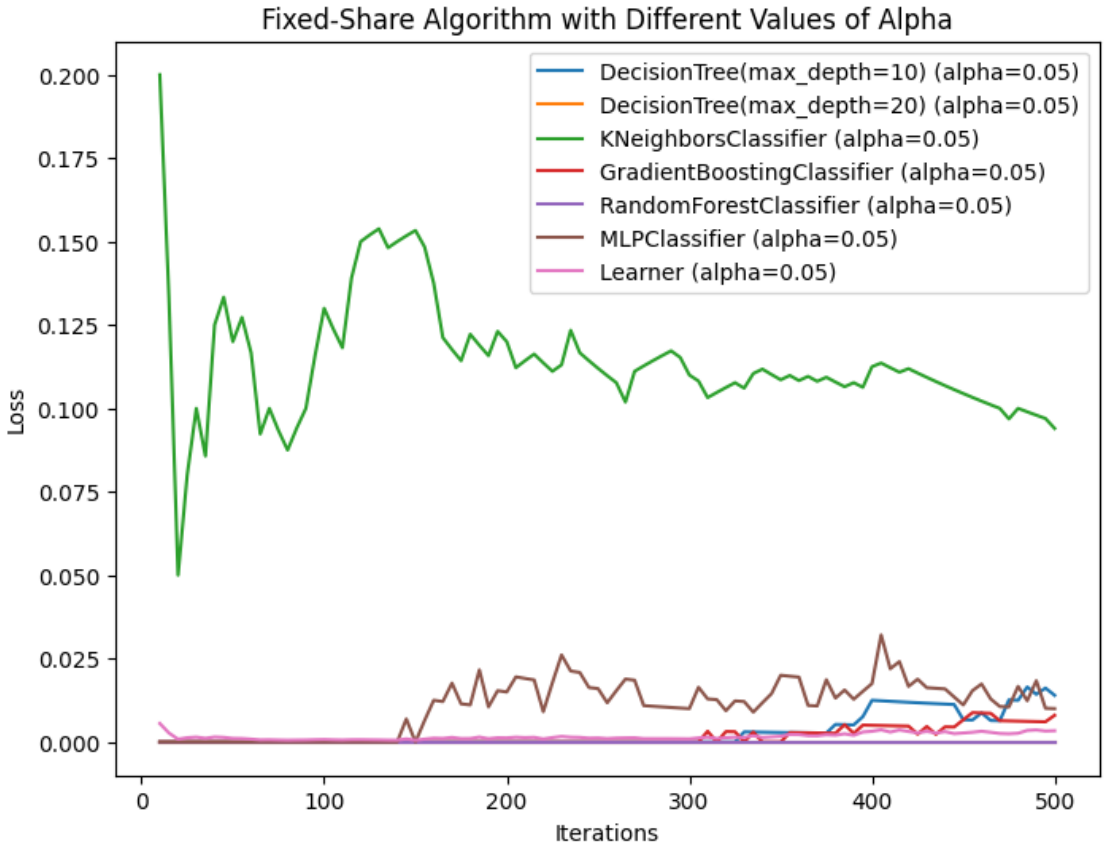


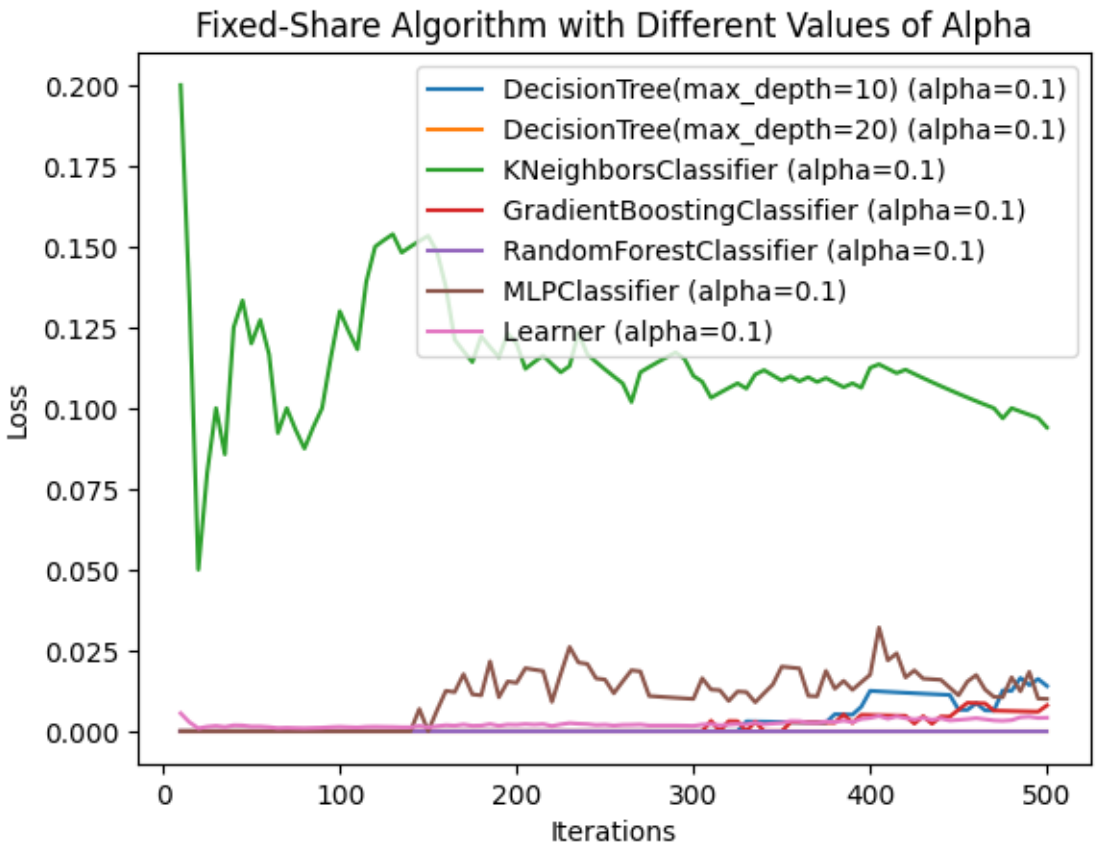
[77]:
```
# Plot the loss of learners in Fixed-Share Algorithm with Different Values of␣
 ↪Alpha
plt.figure(figsize=(8, 6))
for j, alpha in enumerate(alphas):
    plt.plot(num_iterations, fixed_learner_losses_track[:, j], label=f"Learner␣
 ↪(alpha={alpha})")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.ylim(0,0.01)
plt.title("Loss value of learner in Fixed-Share (Alpha) Algorithm")
plt.legend(loc="upper right")
plt.show()
```

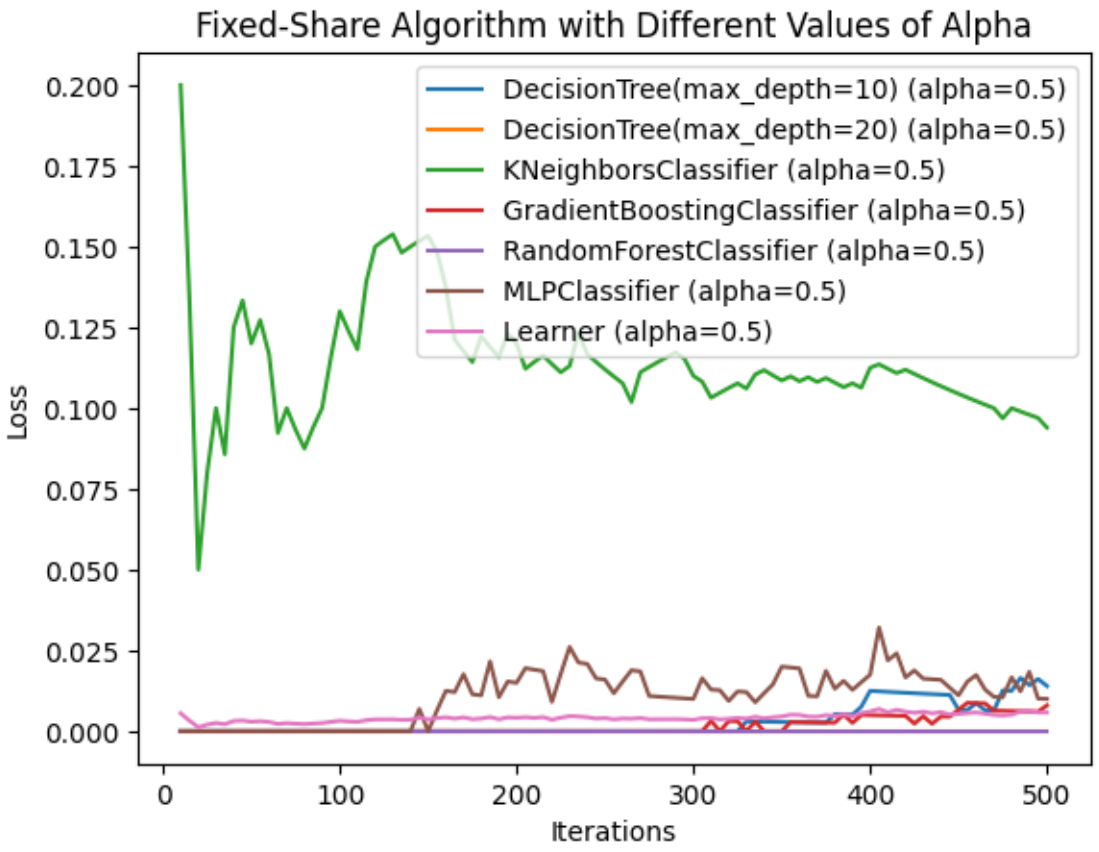


[80]:
```
# Plot the loss values of Fixed-Share Algorithm with Different Values of Alpha
plt.figure(figsize=(8, 6))
for j, alpha in enumerate(alphas):
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 0],␣
↪label=f"DecisionTree(max_depth=10) (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 1],␣
↪label=f"DecisionTree(max_depth=20) (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 2],␣
↪label=f"KNeighborsClassifier (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 3],␣
↪label=f"GradientBoostingClassifier (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 4],␣
↪label=f"RandomForestClassifier (alpha={alpha})")
    plt.plot(num_iterations, fixed_expert_losses_track[:, j, 5],␣
↪label=f"MLPClassifier (alpha={alpha})")
    plt.plot(num_iterations, fixed_learner_losses_track[:, j], label=f"Learner␣
↪(alpha={alpha})")
    plt.xlabel("Iterations")
    plt.ylabel("Loss")
```

```
    plt.title("Fixed-Share Algorithm with Different Values of Alpha")
    plt.legend(loc="upper right")
    plt.show()
```




Fixed-Share Algorithm with Different Values of Alpha
DecisionTree(max_depth=10) (alpha=0.1)
DecisionTree(max_depth=20) (alpha=0.1)
KNeighborsClassifier (alpha=0.1)
GradientBoostingClassifier (alpha=0.1)
RandomForestClassifier (alpha=0.1)
MLPClassifier (alpha=0.1)
Learner (alpha=0.1)
Loss
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
100
200
300
400
500
Iterations



[73]:
```
# Define the experts
tree_expert1 = DecisionTreeClassifier(max_depth=10)
tree_expert2 = DecisionTreeClassifier(max_depth=20)
knn_expert = KNeighborsClassifier()
GBC=GradientBoostingClassifier()
RFC=RandomForestClassifier()
MLPR=MLPClassifier()

# Define the expert weights
weights = [0.2, 0.2, 0.2, 0.2, 0.1, 0.1]

# Define the static expert algorithm
static_expert_algorithm = StaticExpertAlgorithm(experts=[tree_expert1,␣
 ↪tree_expert2, knn_expert, GBC, RFC, MLPR], weights=weights)

# Train the experts and the static expert algorithm
num_iterations = np.arange(10, 501, 5)
static_expert_losses_track = np.zeros((len(num_iterations), 6))
static_learner_losses_track = np.zeros((len(num_iterations)))
```

```
for i, n in enumerate(num_iterations):
    for expert in [tree_expert1, tree_expert2, knn_expert, GBC, RFC, MLPR]:
        expert.fit(X_train[:n], y_train[:n])
    learner_loss, expert_losses, expert_weights = static_expert_algorithm.
 ↪train(X_train[:n], y_train[:n], num_iterations=1)
    static_expert_losses_track[i, :] = np.array(expert_losses)
    static_learner_losses_track[i] = np.reshape(learner_loss, (1,))
```

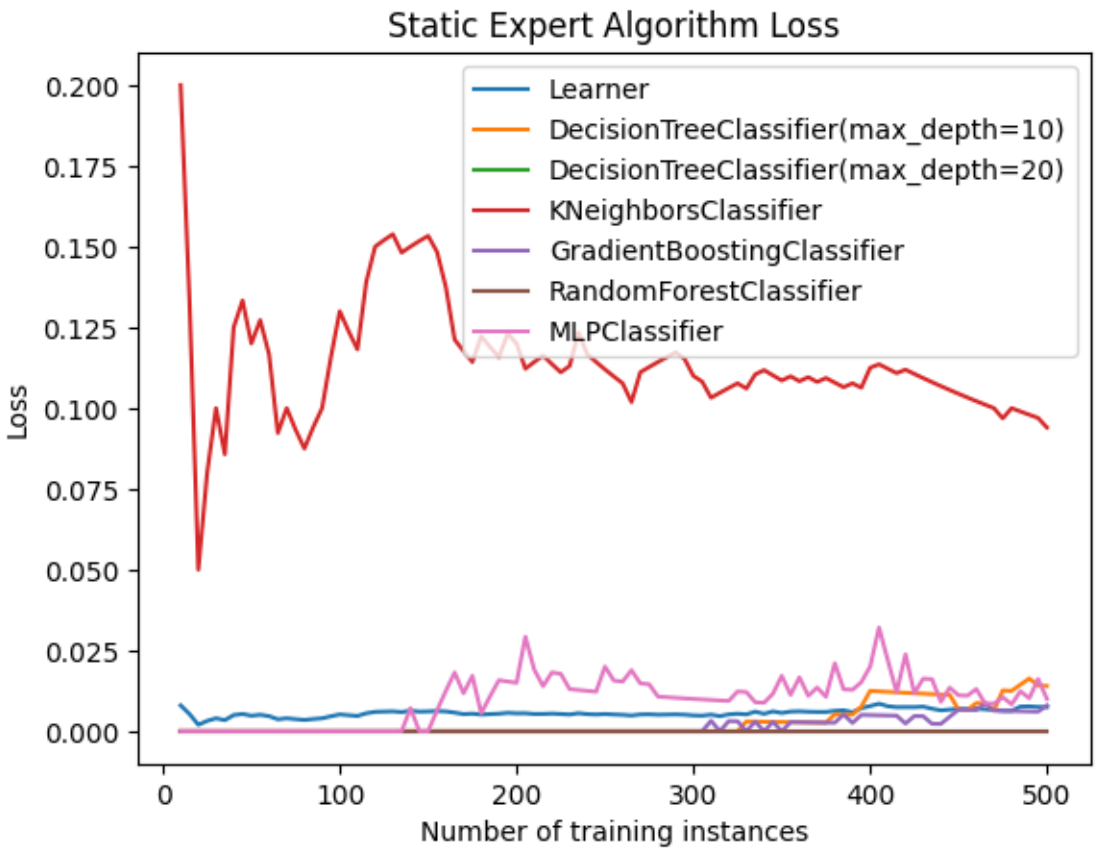
[74]:
```
# Print the cumulative loss of learner vs iteration in Static Expert algorithm

print("cumulative loss of learner vs iteration in Static Expert algorithm:",␣
 ↪static_learner_losses_track)
```

```
cumulative loss of learner vs iteration in Static Expert algorithm: [0.008
0.00533333 0.002      0.0032     0.004      0.00342857
 0.005      0.00533333 0.0048     0.00509091 0.00466667 0.00369231
 0.004      0.00373333 0.0035     0.00376471 0.004      0.00463158
 0.0052     0.00495238 0.00472727 0.00556522 0.006      0.00608
 0.00615385 0.00592593 0.00635714 0.00606897 0.00613333 0.00625806
 0.006125   0.00575758 0.00529412 0.00542857 0.00516667 0.0052973
 0.00542105 0.00569231 0.00555    0.00556098 0.00533333 0.00534884
 0.00545455 0.00533333 0.00517391 0.00557447 0.00529167 0.00518367
 0.00532    0.00517647 0.00507692 0.00486792 0.00518519 0.00523636
 0.00510714 0.00515789 0.0052069  0.00511864 0.0049     0.00481967
 0.00512903 0.00466667 0.00521875 0.00541538 0.00521212 0.0060597
 0.00538235 0.00614493 0.00565714 0.00608451 0.00616667 0.0060274
 0.00597297 0.00597333 0.00631579 0.00646753 0.00587179 0.00713924
 0.0078     0.00851852 0.00773171 0.00744578 0.00747619 0.00745882
 0.00760465 0.00696552 0.00645455 0.00669663 0.00695556 0.00687912
 0.007      0.0067957  0.00655319 0.00633684 0.00633333 0.00762887
 0.00769388 0.00751515 0.00746   ]
```

[78]:
```
# Plot learner loss
plt.plot(num_iterations, static_learner_losses_track, label='Learner')
experts=["DecisionTreeClassifier(max_depth=10)",␣
 ↪"DecisionTreeClassifier(max_depth=20)", "KNeighborsClassifier",
         "GradientBoostingClassifier", "RandomForestClassifier", "MLPClassifier"]
# Plot expert losses
for i, expert in enumerate(experts):
    plt.plot(num_iterations, static_expert_losses_track[:, i], label=f'{expert}')

# Add labels and legend
plt.xlabel('Number of training instances')
plt.ylabel('Loss')
plt.title('Static Expert Algorithm Loss')
plt.legend()
plt.show()
```



[ ]:

Report:

i) The evolution of expert weights with iterations:

In the Fixed Share Algorithm, the evolution of expert weights with iterations are plotted. We have trained the algorithm on the training dataset for 500 iterations, and for each iteration, the weights of the expert models are updated based on their loss. We can observe that, with each iteration, the weights of the expert models change, and according to the performance of each expert, more weights will be assigned to expert which has lower loss. We can also observe that the weights of the expert models become more stable with the increase in the number of iterations. Furthermore, we can observe that the values of alpha have an impact on the weights of the expert models. With smaller value of alpha, better experts can gain more weights as the iteration progress, and results in lower leaner loss at the end.

In the Static Expert Algorithm, the weights of the expert models are not updated, and stays the same as initially assigned. Hence, there is no evolution of expert weights with iterations.

ii) The cumulative loss of the learner versus iterations:

In both Fixed Share Algorithm and Static Expert Algorithm, we have computed the cumulative loss of the learner, which is the mean squared error between the predicted and actual labels, for

each iteration. We can observe that the loss of the learner decreases dramatically at the beginning, but fluctuates with the increase in the number of iterations. Moreover, we can observe that the loss of the learner is smaller for the Fixed Share Algorithm than the Static Expert Algorithm, indicating that the Fixed Share Algorithm can better learn the underlying patterns of the dataset. Furthermore, we can observe that the values of alpha have an impact on the loss of the learner in the Fixed Share Algorithm. For instance, with a smaller value of alpha, the loss of the learner decreases faster in the initial iterations, and always smaller than the loss value with larger alpha. The overall learner loss value in Spambase dataset is much smaller than the Cloud's. There are two possible reasons, first the values of alpha are smaller when I implemented Fixed-Share Algorithm in Spambase, and the experts are also different for two dataset as Spambase fits the supervised learning models, but Cloud dataset fits the unsupervised learning models.

iii) The cumulative loss of the experts versus iterations:

In the Fixed Share Algorithm, we have also computed the cumulative loss of the expert models, which is the mean squared error between the predicted and actual labels, for each iteration. We can observe that the loss of the expert models decreases with the increase in the number of iterations. Moreover, we can observe that the loss of the expert models is smaller for the Fixed Share Algorithm than the Static Expert Algorithm, indicating that the Fixed Share Algorithm can better utilize the expertise of the individual models.

In conclusion, we can observe that the Fixed Share Algorithm outperforms the Static Expert Algorithm in terms of the loss of the learner and the expert models. The values of alpha have an impact on the performance of the Fixed Share Algorithm, with a higher value of alpha leading to faster learning during the iterations. Overall, the experiments demonstrate the effectiveness of ensemble algorithms in utilizing the expertise of individual models to learn the underlying patterns of the data.

Task 2:

Corrective-Conservative framework:

$$U(W^{t+1}) = d(W^{t+1}, W^t) + \eta L(y^t, W^{t+1}x^t)$$

where $d(W^{t+1}, W^t)$ represents distance between W^{t+1} and W^t, $L(y^t, W^{t+1}x^t)$ represents loss value between y-truth and y-predict, and η represents the learning rate, which is a constant.

$$To\ \ minimize\ \ U(W^{t+1}), we\ \ set\ \ U'(W^{t+1}) = 0$$

So,

$$d'(W^{t+1}, W^t) + \eta L'_y(y^t, W^{t+1}x^t) = 0$$

In Gradient Descent Algorithm, it uses square loss and the squared Euclidean distance.

So,

$$d(W^{t+1}, W^t) = \frac{1}{2}\|W^{t+1} - W^t\|_2^2$$

$$d'(W^{t+1}, W^t) = \frac{\partial d(W^{t+1}, W^t)}{\partial W^{t+1}} = W^{t+1} - W^t$$

$$L_y(y^t, W^{t+1}x^t) = (y^t - W^{t+1}x^t)^2$$

$$L'_y(y^t, W^{t+1}x^t) = \frac{\partial L_y(y^t, W^{t+1}x^t)}{\partial W^{t+1}} = 2(W^{t+1}x^t - y^t)x^t$$

Therefore,

$$W^{t+1} - W^t + 2\eta(W^{t+1}x^t - y^t)x^t = 0$$

Solving for W^{t+1}, we get:

$$W^{t+1} = W^t - 2\eta(\hat{y}^t - y^t)x^t, where\ \ \hat{y}^t = W^{t+1}x^t$$

In Exponentiated gradient Algorithm, it uses relative entropy to calculate distance between W^{t+1} and W^t, also known as Kullback-Leibler divergence.

So,

$$d(W^{t+1}, W^t) = \sum_{i=1}^{n} W_i^{t+1} \ln \frac{W_i^{t+1}}{W_i^t}$$

$$d'(W^{t+1}, W^t) = \frac{\partial d(W^{t+1}, W^t)}{\partial W^{t+1}} = \ln \frac{W_i^{t+1}}{W_i^t}$$

Then,

$$\ln \frac{W_i^{t+1}}{W_i^t} + \eta L'_y(y^t, W^{t+1}x^t) = 0$$

Solving for W^{t+1}, we get:

$$W_i^{t+1} = W_i^t r_i^t, \text{where}\ \ r_i^t = \exp(-\eta L'_y(y^t, W^{t+1}x_i^t))$$

Since the summation of all the experts' weight equals to one, $\sum_{i=1}^{N} W_i^{t+1} = 1$

To normalize the weights,

$$W_i^{t+1} = W_i^t r_i^t / \sum_{j=1}^{N} W_j^{t+1}$$

$$W_i^{t+1} = W_i^t r_i^t / \sum_{j=1}^{N} W_j^t r_j^t$$

Task 3:

r:red, b:blue, g:green, o:orange

$$E[h(x)|r] = \frac{2+4+3+1.5+1.5}{5} = 2.4$$

$$E[h(x)|b] = \frac{6+1+2}{3} = 3$$

$$E[h(x)|g] = \frac{6+7+2}{3} = 5$$

$$E[h(x)|o] = \frac{6+8+2+3}{4} = 4.75$$

$$J(h) = \frac{1}{2}E_y[|2-2.4,4-2.4,3-2.4,1.5-2.4,1.5-2.4,6-3,1-3,2-3,6-5,7-5,2-5,6-4.75,8-4.75,2-4.75,3-4.75|]$$

$$J(h) = \frac{1}{2}\left(\frac{0.4+1.6+0.6+0.9+0.9+3+2+1+1+2+3+1.25+3.25+2.25+1.75}{15}\right)$$

$$J(h) = 0.83$$

Task4

Bayes risk is always zero is not true. Because the Bayes risk is the expected loss associated with a given decision rule, and it is defined as the minimum possible risk over all decision rules. The Bayes risk can be non-zero, as it depends on the underlying distribution of the data and the loss function used.

It is not true that estimation error does not depend on the statistical risk of the best possible learning rule, as estimation error measures the difference between the true error of the best possible learning rule and the empirical error of the learning rule estimated from the available data.