EFFECT OF ATTRIBUTE AND CLASS NOISE ON CLASSIFICATION ACCURACY

By:
Samardeep Verma

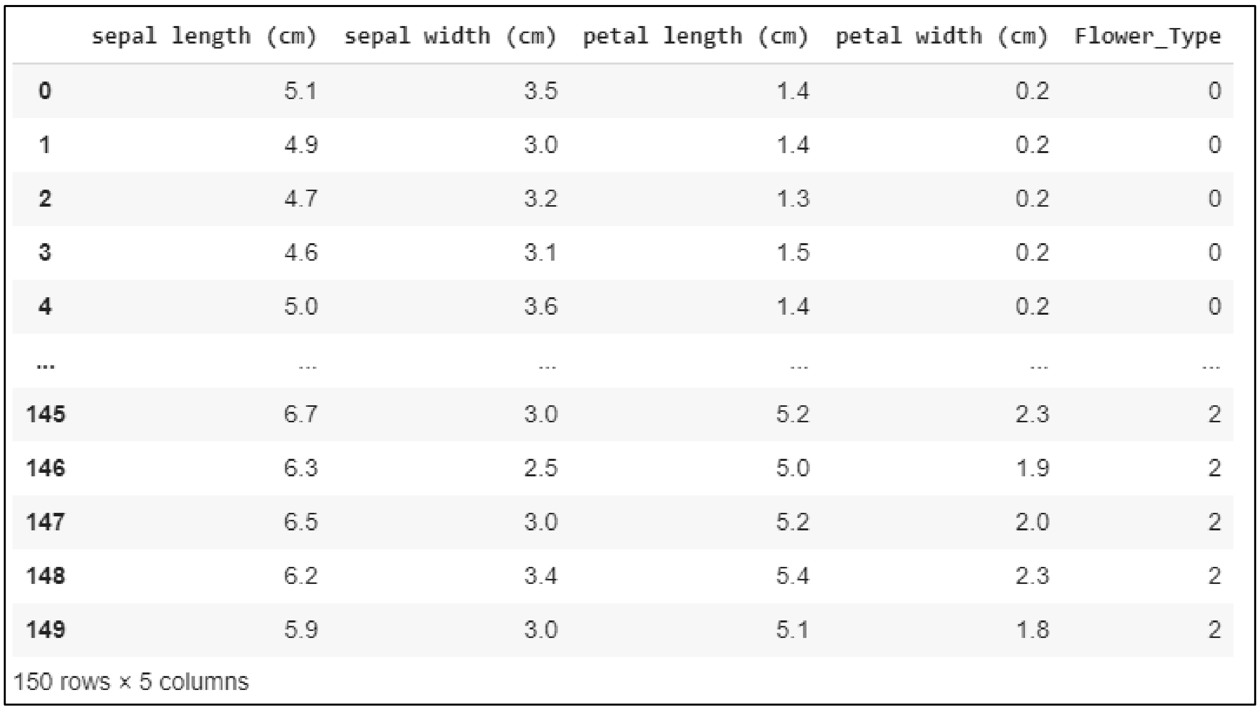
Contents

Introduction

Classification is a technique of categorizing data into a certain number of classes based on given parameters. It is a prominent field of research and study in the field of machine learning. Some real-life examples are recognizing handwritten digits, detecting spam emails, image segmentation, speech recognition etc. In real world data used for classification is full of anomalies. It includes missing values, misclassified labels, inaccurate values etc. Even after cleaning the data, some form of disturbances still prevails in the data. So, there is a need that the algorithms used should be robust enough to deal with such kinds of anomalies.

The purpose of this project is to check the robustness of a few classification algorithms under different levels of attribute and class noise. The dataset used is the very common Iris Flower Dataset (*Figure-1*) which is used to predict the species of the flowers based on available features. Following are the facts about dataset:

- It is available in **sklearn.datasets** module.
- Iris is a flowering plant and this dataset captures the features (sepal length, sepal width, petal length and petal width) of 3 species (Iris setosa, Iris virginica and Iris versicolor).
- There are a total 150 samples (50 samples of each species) in the dataset.
- As computers can only predict based on numbers so the three species Iris setosa, Iris versicolor and Iris virginica are classified as 0,1 and 2 respectively.
- The format for the data: (sepal length, sepal width, petal length, petal width)



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Flower_Type
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

Figure 1

Algorithms used in Project

- **K-Nearest Neighbors (KNN):** This is one of the most widely used machine learning algorithm helpful in both classification and regression. As its name suggests, it makes predictions based on the K (any integer) nearby values. It is a supervised learning algorithm which means the training data must be labelled i.e. contains both dependent and independent variables. For implementing this algorithm, KNeighboursClassifier is used from sklearn.neighbors module.

- **Support Vector Machines (SVM):** This technique is mainly used for classification. It is based on the construction of hyperplane in a N-dimensional space (N is number of features) that separates the data points belonging to different classes. E.g. If number of features are 2 then hyperplane is one dimensional, if features are 3 then 2-D hyperplane exists and so on. Objective is to find a plane that has maximum margin i.e. maximum distance from data points of both classes. Here SVC classifier is imported from sklearn.svm module for implementation.

- **Stochastic Gradient Descent (SGD):** In Gradient Descent we use the whole data for every iteration in order to get the weights which will minimize sum of squared residuals. But when dataset is huge then this process is very slow. To overcome that, SGD is used which uses some random values instead of whole dataset in every iteration. Here, SGDClassifier is imported from sklearn.linear_model module.

- **Logistic Regression:** This algorithm is based on logit function ($p = 1/(1 + \exp\{-(b_0 + b_1x_1 + \ldots + b_kx_k)\})$) which calculates probability of an event based on the value of linear function ($b_0 + b_1x_1 + \ldots + b_kx_k$). If value is high then probability is high and if value is low then probability is also less. This algorithm is one of the oldest and most widely used algorithms for classification. LogisticRegression classifier is imported from sklearn.linear_model module.

- **Decision Tree:** This is a tree-based classification algorithm in which population or sample is divided into 2 or more sub-populations based on a significant splitter. Then these subpopulations are further divided into more categories and this process continues. Depth of the tree depends upon number of features available in dataset. The splitter is decided based on Gini index. In python, sklearn.tree module is used to implement Decision Tree Classifier.

- **Extreme Gradient Boosting (XGBoost):** Gradient boosting is a technique to produce a strong prediction model from a group of weak prediction models. XGBoost is a decision tree based ML algorithm which uses gradient boosting framework. It provides parallel tree boosting to solve the data science problems quickly and efficiently. In Python, XGBClassifier is imported from xgboost package for classification.

- **Gaussian Naive Bayes (GaussianNB):** Bayes theorem calculates the probability of occurrence of an incident ‘A’ given that incident ‘B’ has already occurred. In case of classification, output variable is incident ‘A’ and incident ‘B’ are our predictor variables or features. For GaussianNB, predictor variables are continuous not discrete. GaussianNB classifier is imported from sklearn.naive_bayes module for classification.

- **Random Forest:** This algorithm is based on the concept of Wisdom of Crowds i.e. many people can predict better than an individual. Outputs from large number of uncorrelated decision trees are taken and the most common output is the final result of classification. RandomForestClassifier is imported from sklearn.ensemble module for implementation.

Methodology

Noise Types

Two types of noises are used to check the accuracy of algorithms:
Attribute Noise: It is generated when some values in the predictor variables are either missing, incomplete or have errors. E.g. In the Iris dataset if one of the samples has its value recorded as 500cm instead of 5cm then it is an attribute noise.
Here in order to insert attribute noise, we used **n*(random number between -1 & 1)** where n is the level of noise. These random values are inserted in all the variables in the training dataset. Based on this, accuracy scores of different models are predicted.

Class Noise: While performing classification, when certain values in a class are flipped or misinterpreted with some other values of that class then this generates class noise.eg. In the Iris dataset some samples of Iris-Setosa can be recorded as Iris-Versicolor or vice versa.
Here we are choosing a certain percentage(level) of values to be misclassified from the training dataset. Then we labelled:

- ❖ Iris Setosa (0) as Iris Versicolor (1)
- ❖ Iris Versicolor (1) as Iris Virginica (2)
- ❖ Iris Virginica (2) as Iris Setosa (0)

Procedure of generating the class noise:

- ❖ In the training dataset (y_train only) there are a total of 120 values. All the values are assigned different numbers ranging from (1/120) to (120/120).
- ❖ Then these numbers are shuffled randomly.
- ❖ Numbers above a certain level are selected e.g. numbers > (1- percentage of class noise)
- ❖ Then y_train values corresponding to those selected numbers are mislabeled.

Parts of Code

Refer Appendix for detailed code. The whole project is divided into following parts:
Part 1: Preparing the data set

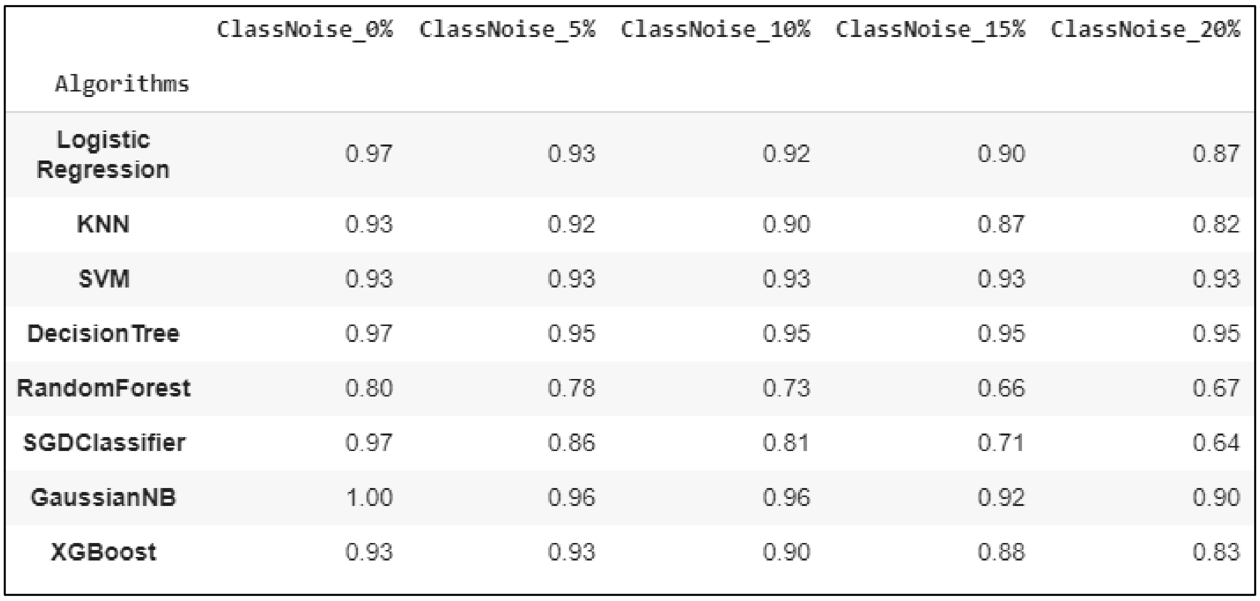
- ❖ At first all the four predictor variables are converted to Z-values in order to remove the influence of any particular variable in classification.
- ❖ After loading the data set from sklearn.datasets module, it is split into training and validation datasets in 80:20 ratio.
- ❖ Functions are defined to insert class noise or attribute noise based on desired level.
- ❖ And finally using these functions noise is added to the training dataset which is further used to train different models.

Part 2: Loading different classification algorithms

- ❖ As mentioned above, there are 8 different classification algorithms.
- ❖ All the algorithms are loaded from their respective modules.
- ❖ Separate functions are defined for every algorithm which includes training and testing data as their parameters and return the accuracy score of that algorithm.

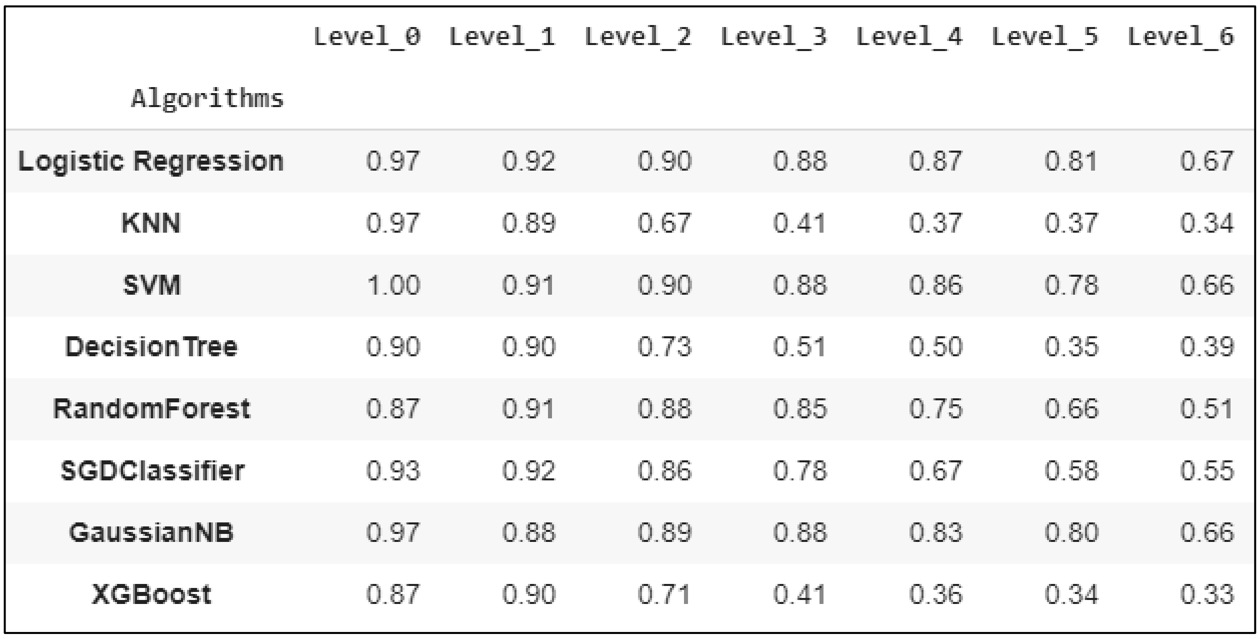
Part 3: Training and validation phase

- ❖ For a particular level of noise, one epoch consists of the following steps:
  - ➢ First a noisy training dataset is generated.
  - ➢ Then this dataset is used to train and predict the accuracy score of all the algorithms.
  - ➢ All the accuracies are recorded.
- ❖ For every level of noise, 100 epochs are run and averages of all these epochs for different algorithms are taken separately to report the final accuracy score.
- ❖ Finally, the data is displayed in the form of a table separately for class noise (*Figure-2*) and attribute noise (*Figure-3*).



	ClassNoise_0%	ClassNoise_5%	ClassNoise_10%	ClassNoise_15%	ClassNoise_20%
Algorithms					
Logistic Regression	0.97	0.93	0.92	0.90	0.87
KNN	0.93	0.92	0.90	0.87	0.82
SVM	0.93	0.93	0.93	0.93	0.93
DecisionTree	0.97	0.95	0.95	0.95	0.95
RandomForest	0.80	0.78	0.73	0.66	0.67
SGDClassifier	0.97	0.86	0.81	0.71	0.64
GaussianNB	1.00	0.96	0.96	0.92	0.90
XGBoost	0.93	0.93	0.90	0.88	0.83

Figure 2



	Level_0	Level_1	Level_2	Level_3	Level_4	Level_5	Level_6
Algorithms							
Logistic Regression	0.97	0.92	0.90	0.88	0.87	0.81	0.67
KNN	0.97	0.89	0.67	0.41	0.37	0.37	0.34
SVM	1.00	0.91	0.90	0.88	0.86	0.78	0.66
DecisionTree	0.90	0.90	0.73	0.51	0.50	0.35	0.39
RandomForest	0.87	0.91	0.88	0.85	0.75	0.66	0.51
SGDClassifier	0.93	0.92	0.86	0.78	0.67	0.58	0.55
GaussianNB	0.97	0.88	0.89	0.88	0.83	0.80	0.66
XGBoost	0.87	0.90	0.71	0.41	0.36	0.34	0.33

Figure 3

Part 4: Plotting the results

Now the accuracy scores for all the algorithms at every noise level are plotted on a line graph (Figure-4) in order to do comparisons. Matplotlib package is used. Accuracy is measured in terms of percentage e.g. 0.4 is 40%, 1 is 100% etc.

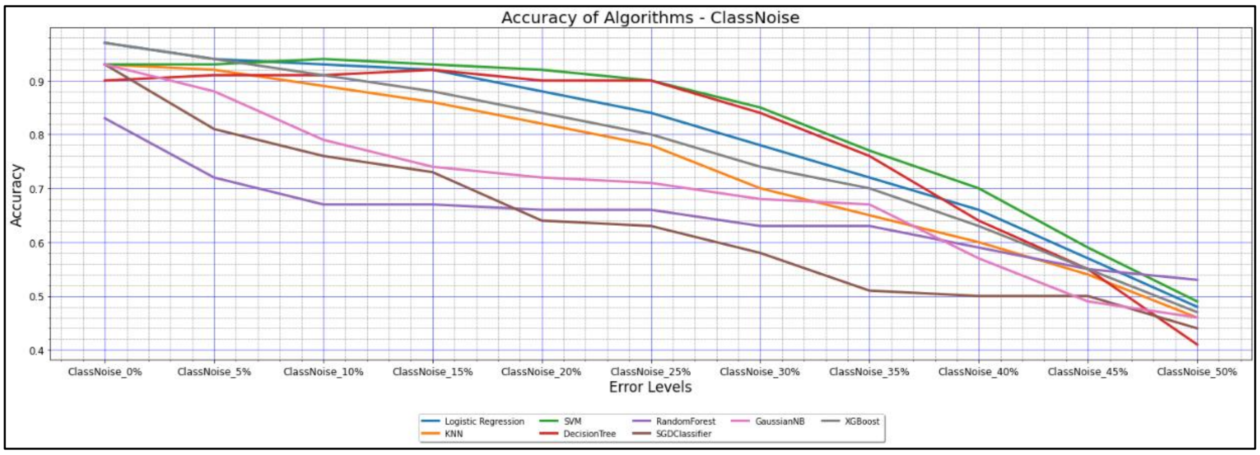


Figure 4

Part 5: Generating box plots to check variability of results

As we are taking average accuracy scores over 100 epochs, box plots will help us to discover variability of accuracy scores during these epochs for every algorithm at every noise level (*Figure-5*). Plotting all box plots for all 8 algorithms will make it difficult to understand. So, this code will require user inputs to draw boxplots based on any combination of noise level and algorithm. Following steps are required:

- Enter the number of noise levels required in the boxplot e.g. 2 or 3 or 4 etc.
- Enter individual noise levels e.g. 5,10,15,20 etc.
- Provide confirmation in form of Yes (y) or No (n). If you want to get out of the code at this point, type 'exit'.
- Enter the number of algorithms required in the boxplot e.g. 2 or 3 or 4 etc.
- Enter individual algorithm numbers as displayed on screen. e.g. 1,2,3 etc.
- Provide confirmation in form of Yes (y) or No (n). If you want to get out of the code at this point, type 'exit'. After this we can display the boxplot based on chosen combination.

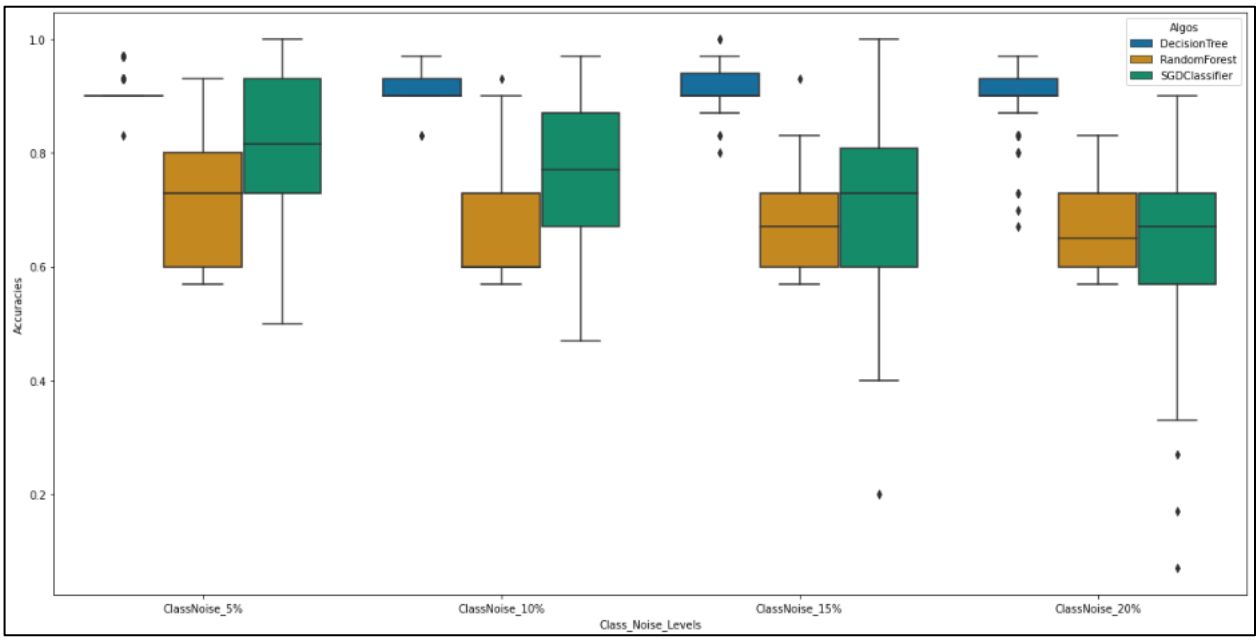


Figure 5

Observations

All the observations are recorded separately based on class noise and attribute noise. Some algorithms worked better in case of class noise while others performed well in attribute noise.

Attribute noise observations:

- ❖ All algorithms were more than 90% accurate on a clean dataset (Level 0 noise level). Refer (*Figure-6*)
- ❖ They worked well even at Level_1 noise but things changed drastically after that.
- ❖ Logistic regression, Support vector machines and Gaussian Naive Bayes maintained an accuracy around 80% till Level_5 noise.
- ❖ Random forest and Stochastic gradient descent were 60% accurate near Level_5 noise.
- ❖ Decision trees, XGBoost and K-nearest neighbors recorded a steep fall in accuracy after Level_1 only. By Level_5, their accuracy was below 40%.
- ❖ Accuracies of all the algorithms converged around 30% near extremely noise data (Level_10 noise).
- ❖ This shows that Logistic regression, Support vector machines and Naive Bayes can handle attribute noise better than other algorithms. Among them Logistic regression is the best.

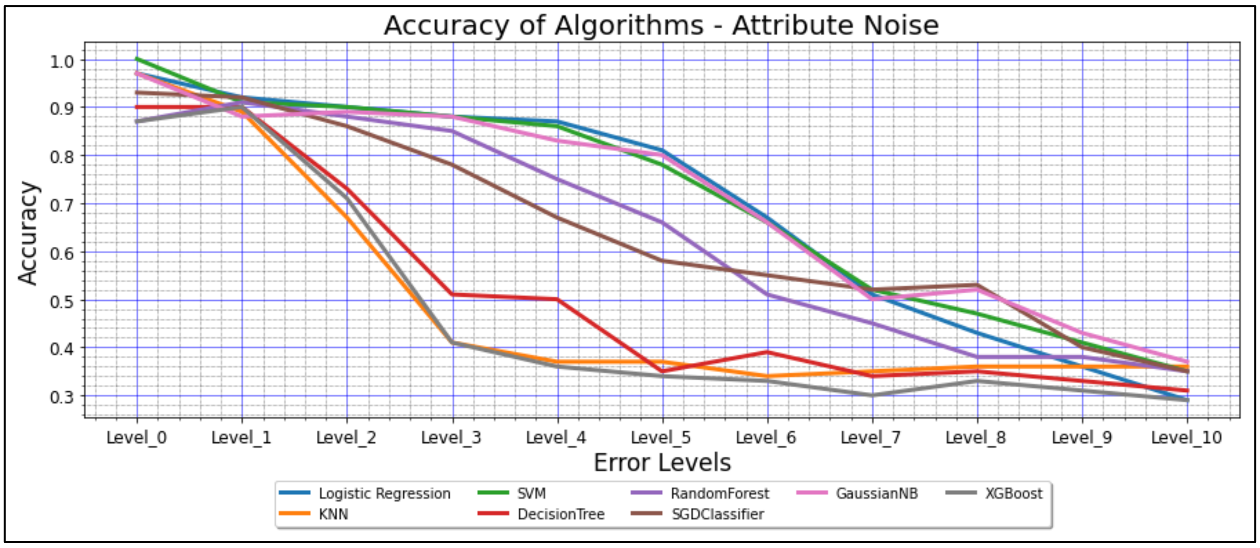


Figure 6

Class Noise observations:

- ❖ Till 20% class noise i.e. 20% misclassified values, all algorithms except Naive Bayes, Random forest and Stochastic gradient descent held an accuracy level above 80%. Refer (*Figure-7*).
- ❖ Decision trees and support vector machines are 90% accurate even at 25% class noise. They even performed better as compared to other algorithms even when class noise increased.
- ❖ Random forest performed poorly even at 10% class noise.
- ❖ At 50% class noise, all the algorithms were less than 50% accurate.
- ❖ Stochastic gradient descent and random forest are the worst choices for handling class noise.

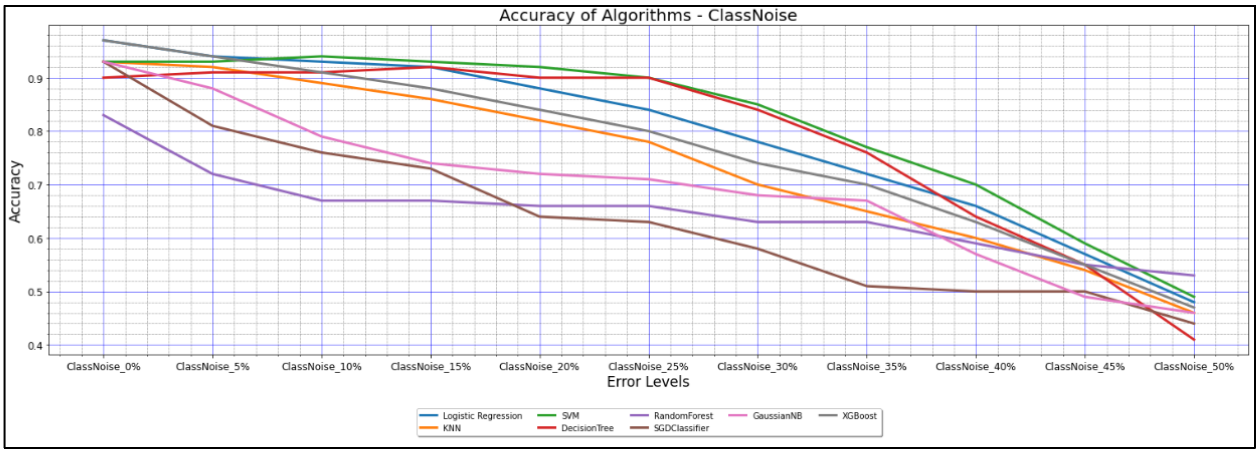


Figure 7

Algorithm wise observations:

- ❖ Logistic Regression and Support vector machines were able to handle both class noise and attribute noise better as compared to others.
- ❖ Decision trees handled class noise very well but showed poor accuracy in case of attribute noise.
- ❖ Similarly, KNN and XGBoost did a decent job while handling class noise but were poor in case of attribute noise.
- ❖ Random forest and stochastic gradient performed poorly in both types of noises.
- ❖ As we are calculating mean of 100 epochs for every level of noise for all algorithms, *Figure 8 & 9* show the variability of accuracies in 100 epochs. For both the noises as disturbance increases, mean accuracy decreases but variability increases.

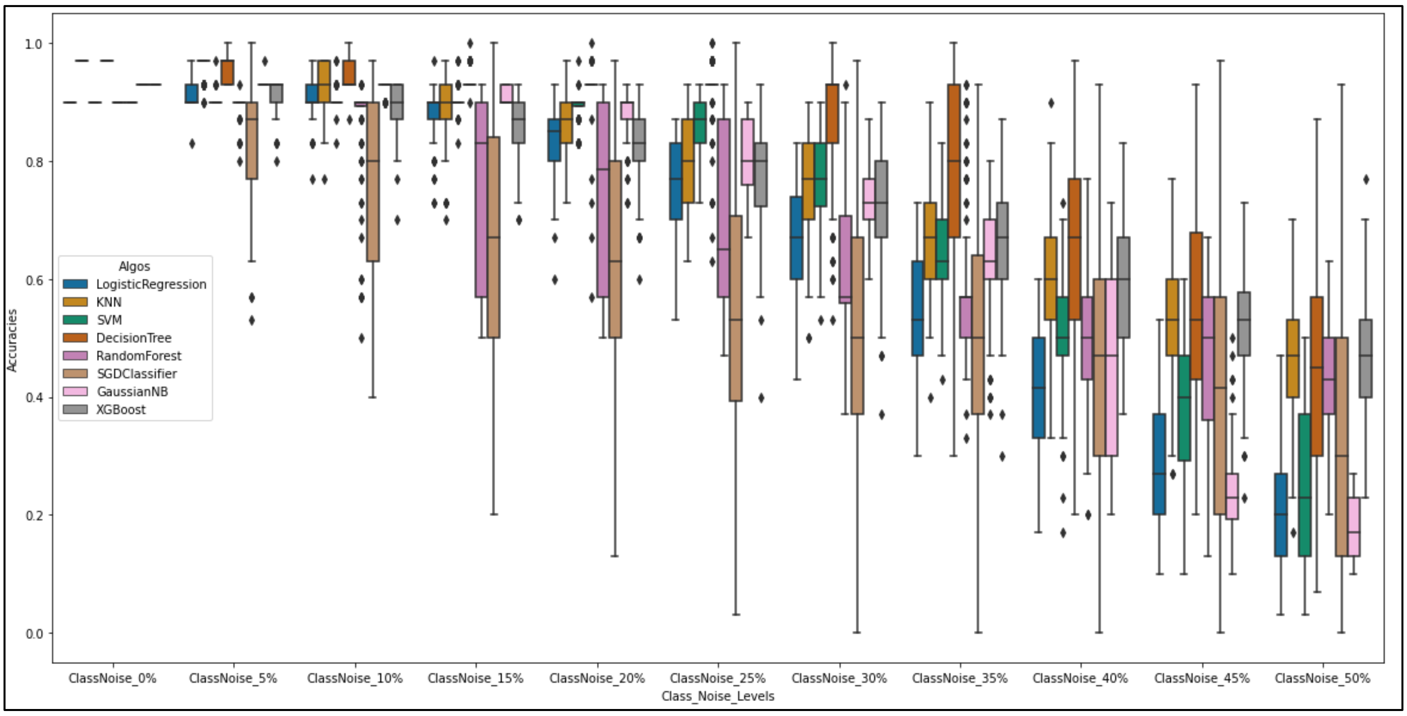


Figure 8

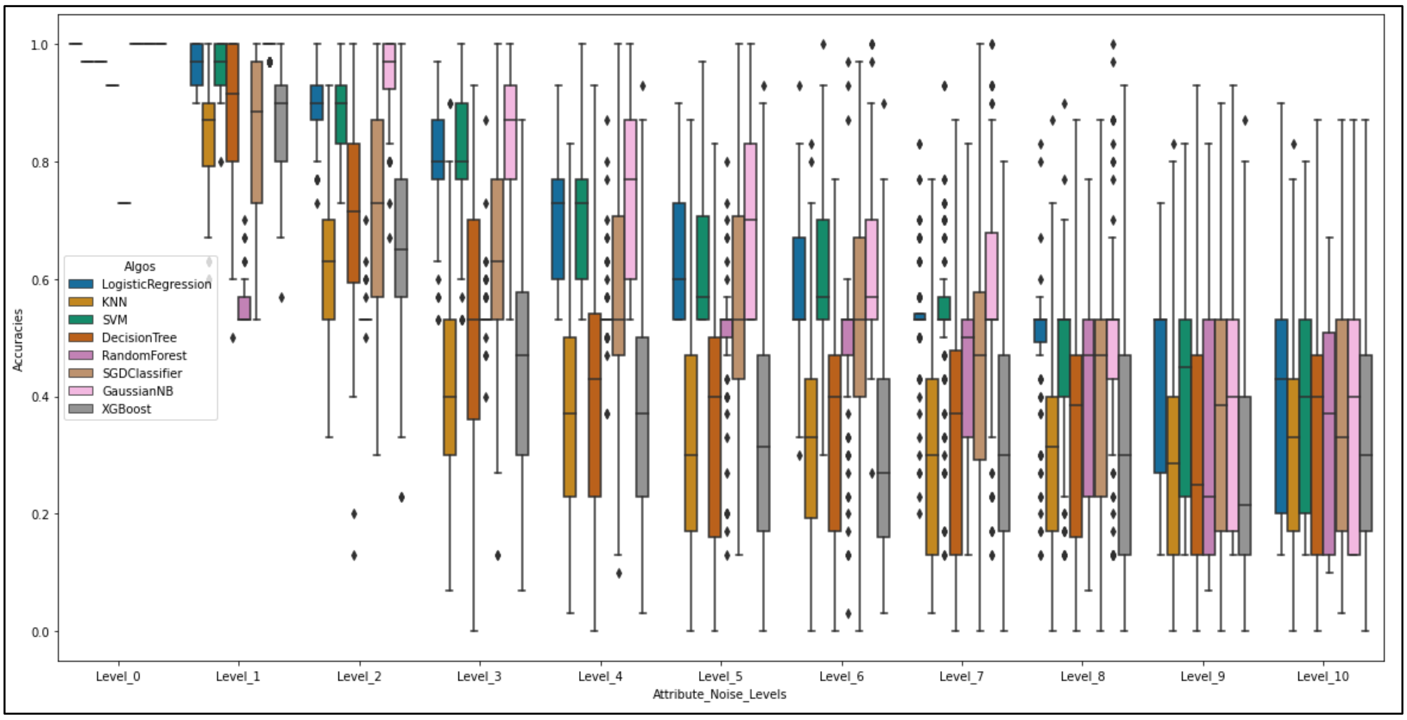


Figure 9

References

- "https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d"
- "https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47"
- "https://machinelearningmastery.com/logistic-regression-for-machine-learning/"
- "https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/"
- "https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c"
- "https://towardsdatascience.com/understanding-random-forest-58381e0602d2"

Appendix

Code for Class noise

Part 1: Preparing the data set

```
#Loading the IRIS Dataset & Printing the
from sklearn.datasets import load_iris
iris=load_iris()

#Printing the type of outputs
print(iris.target_names)
print(set(iris.target))

#Printing the whole dataset
import pandas as pd
iris_dataset = pd.DataFrame(iris.data, columns = iris.feature_names)
iris_dataset["Flower_Type"] = iris.target
iris_dataset

#Seperating the data into dependent and independent variables
X=iris.data
y=iris.target

#Normalizing the data and splitting into testing and training data set.
from sklearn.model_selection import train_test_split
from scipy.stats import zscore
X_Zscore = pd.DataFrame(X).apply(zscore)
X_Zscore = X_Zscore.to_numpy()
X_train,X_test,y_train,y_test=train_test_split(X_Zscore,y,test_size=0.2)
```

Part 2: Defining functions for different Algorithms

NOTE: In all the functions used to train and produce results for various algorithms, one paramete caller "print_flag" is used with value (0 or 1). If value is 1 then function will also print Test values and predicted values for comparison. If 0 then it will only return accuracy_score. By default it is made 0 in order to keep the results produced compact. While calling individual functions, one can change it to 1 to more detailed analysis.

```
#Defining logistic regression function
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import roc_auc_score
def training_function_LogisticRegression(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = LogisticRegression(solver = 'lbfgs',multi_class = 'auto')
    my_model.fit(x_train, y_train)
    y_pred = my_model.predict(x_test)

    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print ('Accuracy of Logistic Regression on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('------------------------------------------------')
    return (metrics.accuracy_score(y_test,y_pred))


#Defining KNN function
from sklearn.neighbors import KNeighborsClassifier
def training_function_KNN(x_train, y_train,x_test,y_test,neighbours, print_flag=0):
    my_model = KNeighborsClassifier(n_neighbors = neighbours)
    my_model.fit(x_train, y_train)
    y_pred = my_model.predict(x_test)

    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of KNN on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('------------------------------------------------')



    return (metrics.accuracy_score(y_test,y_pred))

#Defining fuction for SVM model
from sklearn.svm import SVC
def training_function_SVM(x_train, y_train,x_test,y_test,kernel_name, print_flag=0):
    my_model = SVC(kernel = kernel_name,probability=True)
    my_model.fit(x_train, y_train)
    y_pred = my_model.predict(x_test)

    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of SVM on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))


#Defining fuction for XGBoost model
from xgboost import XGBClassifier
def training_function_XGBoost(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = XGBClassifier()
    my_model.fit(x_train, y_train)
    y_pred = my_model.predict(x_test)

    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of XGBoost on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))


#Defining fuction for Naive Bayes algorithm
from sklearn.naive_bayes import GaussianNB
def training_function_GaussianNB(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = GaussianNB()
    y_pred = my_model.fit(x_train, y_train).predict(x_test)


    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of Naive Bayes Classifier on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')



    return (metrics.accuracy_score(y_test,y_pred))

#Defining fuction for Stochastic gradient descent
from sklearn.linear_model import SGDClassifier
def training_function_SGDClassifier(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = SGDClassifier(loss = 'modified_huber', shuffle = True, random_state = 101)
    y_pred = my_model.fit(x_train, y_train).predict(x_test)


    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of SGDClassifier on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))


#Defining fuction for Decision Tree
from sklearn.tree import DecisionTreeClassifier
def training_function_DecisionTree(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = DecisionTreeClassifier(max_depth = 10, random_state=101,
                        max_features = None, min_samples_leaf = 15)
    y_pred = my_model.fit(x_train, y_train).predict(x_test)

    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of DecisionTree classifier on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))


#Defining fuction for Random Forest
```

```
from sklearn.ensemble import RandomForestClassifier
def training_function_RandomForest(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = RandomForestClassifier(n_estimators = 70, oob_score = True,
                                      n_jobs = -1, random_state = 101,
                                      max_features = None, min_samples_leaf = 30)
    y_pred = my_model.fit(x_train, y_train).predict(x_test)


    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of Random forest classifier on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))
```

Part 3: Training and validation phase

```
import numpy as np
Training_output_length = len(y_train)
numbers = (np.arange(1,Training_output_length+1))/Training_output_length
numbers
# A function to return a dataframe containing y_train and corresponding random value
from random import shuffle
def random_number_generator():
  num = np.copy(numbers)
  shuffle(num)
  y_train_df = pd.DataFrame({'y_train': y_train, 'Random_numbers' :num})
  return y_train_df

print("Printing y_train after shuffling the numbers:")
random_number_generator()

# Function to add class noise based on required level.
# Level means the percentage of elemets in the y_train we want to be disturbed.
# Function will return disturbed y_train
def add_class_noise(level):
  a = random_number_generator()
  Training_output_length = len(y_train)
  new = []
  for i in range(Training_output_length):

    if (a.iloc[[i],[1]] >  1-(level/100)).bool():
      if ((a.iloc[[i],[0]]) == 0).bool():
        new.append(int(1))
      elif ((a.iloc[[i],[0]]) == 1).bool():
        new.append(int(2))
      else:
        new.append(int(0))

    else:
      k = a.iloc[i][0]
      new.append(int(k))

  a['new'] = new
  new_output_trainig = a['new']
  return np.array(new_output_trainig)
#Testing code to show how the number of values disturbed based on given level of disturbance.
# This code is used nowhere else, it is only for the examination of add_class_noise(level) function.
count = 0
new_training_data = add_class_noise(40) #Change this variable to add desired level of class noise in %terms
for i,j in zip(y_train, new_training_data):
  if (i!=j):
    count = count + 1

print ("Total counts in y_train: ",len(y_train))
print ("Total changes in the y_train: ",count)


#This function caluclates the average accuracy scores over 100 epochs for all the algorithms at every level of noise
# This function returns four types of values
# Level_error: List of Mean accuracies for every level of class noise for every algorithm

# Below three variables are used while plotting boxplots
# Accuracies: List of accuracy at every epoch for every level of class noise for every algorithm
# Level_name: List of class noise level at every epoch for every level of class noise for every algorithm
# Algo_name: List of algorithms used at every epoch for every level of class noise for every algorithm

def different_error_levels(input): #Input is a list of different levels of noise
```

```
    Accuracies = []
    Level_name = []
    Algo_name = []
    Algorithms = ['LogisticRegression','KNN','SVM','DecisionTree','RandomForest','SGDClassifier','GaussianNB','XGBoost']
    Level_error = [] # This is a list of lists which contains mean errors for each level of noise for all the used algorithms
    for i in input:
        level = i
        k = []
        for cnt in range(100): # For every level of noise, code is run 100 times and its average is taken
            y_train_new = add_class_noise(level)
            Errors = [
            training_function_LogisticRegression(X_train,y_train_new,X_test,y_test).round(2),
            training_function_KNN(X_train,y_train_new,X_test,y_test,3).round(2),
            training_function_SVM(X_train,y_train_new,X_test,y_test,'linear').round(2) ,
            training_function_DecisionTree(X_train,y_train_new,X_test,y_test).round(2) ,
            training_function_RandomForest(X_train,y_train_new,X_test,y_test).round(2) ,
            training_function_SGDClassifier(X_train,y_train_new,X_test,y_test).round(2),
            training_function_GaussianNB(X_train,y_train_new,X_test,y_test).round(2) ,
            training_function_XGBoost(X_train,y_train_new,X_test,y_test).round(2)
           ]
            k.append(Errors)
            Accuracies.append(Errors)
            Level_name.append(['ClassNoise_'+str(level)+'%']*8)
            Algo_name.append(Algorithms)

        mean_error = np.mean(k, axis = 0) #Calculates mean error for all algorithnms for a partular level of noise

        Level_error.append(mean_error)
    return Level_error,Accuracies,Level_name,Algo_name


#This code is used to get accuracy scores for various noise levels
class_noise_percentages = [0,5,10,15,20,25,30,35,40,45,50] #Change the values in the list to get data upto any level of noise
Level_error,Accuracies,Level_name,Algo_name = different_error_levels(class_noise_percentages) # Calling the above function (NOTE: This will take time to execute based on number of levels used)

#Printing the dataframe
Algorithms = [ 'Logistic Regression','KNN','SVM','DecisionTree','RandomForest','SGDClassifier','GaussianNB','XGBoost']
class_noise_length = list(range(len(class_noise_percentages)))
for (i,j) in zip(class_noise_percentages,class_noise_length):
    df['ClassNoise_'+str(i)+'%'] = Level_error[j].round(2)
df= df.set_index('Algorithms')
df
```

Part 4: Plotting the results

```
#Plotting the line graph for various noise levels
import matplotlib.pyplot as plt
n = len(class_noise_percentages)
plt.figure(figsize=(n+15,n/1.5))
for i in Algorithms:
    plt.plot(df.loc[i,:], label=i, linewidth=3)


plt.grid(which='major', linestyle='-', linewidth='0.5', color='blue')
plt.minorticks_on()
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.title("Accuracy of Algorithms",fontsize='20')
plt.xlabel('Error Levels',fontsize='17')
plt.ylabel('Accuracy',fontsize='17')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15),
          fancybox=True, shadow=True, ncol=5)
```

Part 5: Generating barplots

```
# The three variable generated in different_error_levels(input) function are combined to form a dataframe

Total_error_levels = len(class_noise_percentages)

dm = pd.DataFrame()

dm['Accuracies'] = sum(Accuracies,[])

dm['Class_Noise_Levels'] = sum(Level_name,[])

dm['Algos'] = sum(Algo_name,[])
```

```
dm

#Selecting the levels of noise for box plots
#This piece of code needs user input

def get_n_levels():
  n_levels = input ("Enter the number of levels: "+str(len(class_noise_percentages)))
  levels = []
  for i in range(int(n_levels)):
    s = input("Enter the class noise % "+str(class_noise_percentages)+" :")
    levels.append("ClassNoise_"+str(s)+'%')
  return (levels)



ans = 's'
levels = []
while(ans != "Y" and ans !="EXIT"):
  lev = []
  lev = get_n_levels()
  print(lev)
  ans = input(print(''' Do you want above levels: Y/N
  OR
  Enter EXIT to move out. '''))
  ans = ans.upper()
  Class_Noise_Levels = lev

#Selecting the Algorithms for box plots
#This piece of code needs user input

Algos = {1:'LogisticRegression',
         2:'KNN',
         3:'SVM',
         4:'DecisionTree',
         5:'RandomForest',
         6:'SGDClassifier',
         7:'GaussianNB',
         8:'XGBoost'}
print("Following Algos are availale:")
for i in Algos:
  print(str(i)+':'+Algos[i])



def get_algos():
  B_algos = []
  n_algos = input ("Enter the number of algos: ")
  for i in range(int(n_algos)):
    s = int(input("Enter the algo number: "))
    B_algos.append(Algos[s])
  return B_algos
```

```
ans = 's'
Box_algos = []
while(ans != "Y" and ans !="EXIT"):
  b = []
  b = get_algos()
  print(b)
  ans = input(print(''' Do you want above Algorithms in boxplots: Y/N
  OR
  Enter EXIT to move out. '''))
  ans = ans.upper()
  Box_algos = b

print("Algos used in Box plots: ",Box_algos)
print("Levels of noise used in Box plots: ",Class_Noise_Levels)
#Selecting the data based on the inputs of levels and Algos provided by the user
box_data = dm[dm['Algos'].isin(Box_algos) & dm['Class_Noise_Levels'].isin(Class_Noise_Levels)]
box_data

#Plotting the box plots of the selected data
import seaborn as sns
plt.figure(figsize=(20,10))
sns.boxplot(y='Accuracies', x='Class_Noise_Levels',
                data=box_data,
                palette="colorblind",
                hue='Algos')
```

Code for Attribute noise

Part 1: Preparing the data set

```
#Loading the IRIS Dataset & Printing the
from sklearn.datasets import load_iris
iris=load_iris()

#Printing the type of outputs
print(iris.target_names)
print(set(iris.target))

#Printing the whole dataset
import pandas as pd
iris_dataset = pd.DataFrame(iris.data, columns = iris.feature_names)
iris_dataset["Flower_Type"] = iris.target
Iris_dataset

#Seperating the data into dependent and independent variables
X=iris.data
y=iris.target

#Normalizing the data and splitting into testing and training data set.
from sklearn.model_selection import train_test_split
import pandas as pd
from scipy.stats import zscore
X_Zscore = pd.DataFrame(X).apply(zscore)
X_Zscore = X_Zscore.to_numpy()
X_train,X_test,y_train,y_test=train_test_split(X_Zscore,y,test_size=0.2)
```

Part 2: Defining functions for different Algorithms

NOTE: In all the functions used to train and produce results for various algorithms, one paramete caller "print_flag" is used with value (0 or 1). If value is 1 then function will also print Test values and predicted values for comparison. If 0 then it will only return accuracy_score. By default it is made 0 in order to keep the results produced compact. While calling individual functions, one can change it to 1 to more detailed analysis.

```
#Defining logistic regression function
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import roc_auc_score
def training_function_LogisticRegression(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = LogisticRegression(solver = 'lbfgs',multi_class = 'auto')
    my_model.fit(x_train, y_train)
    y_pred = my_model.predict(x_test)

    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print ('Accuracy of Logistic Regression on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))

#Defining KNN function
from sklearn.neighbors import KNeighborsClassifier
def training_function_KNN(x_train, y_train,x_test,y_test,neighbours, print_flag=0):
    my_model = KNeighborsClassifier(n_neighbors = neighbours)
    my_model.fit(x_train, y_train)
    y_pred = my_model.predict(x_test)

    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of KNN on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')
    return (metrics.accuracy_score(y_test,y_pred))


#Defining fuction for SVM model
from sklearn.svm import SVC
def training_function_SVM(x_train, y_train,x_test,y_test,kernel_name, print_flag=0):
    my_model = SVC(kernel = kernel_name,probability=True)
    my_model.fit(x_train, y_train)
    y_pred = my_model.predict(x_test)

    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of SVM on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))


#Defining fuction for XGBoost model
from xgboost import XGBClassifier
def training_function_XGBoost(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = XGBClassifier()
    my_model.fit(x_train, y_train)
    y_pred = my_model.predict(x_test)

    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of XGBoost on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))


#Defining fuction for Naive Bayes algorithm
from sklearn.naive_bayes import GaussianNB
def training_function_GaussianNB(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = GaussianNB()
    y_pred = my_model.fit(x_train, y_train).predict(x_test)


    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of Naive Bayes Classifier on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')



    return (metrics.accuracy_score(y_test,y_pred))


#Defining fuction for Stochastic gradient descent
from sklearn.linear_model import SGDClassifier
def training_function_SGDClassifier(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = SGDClassifier(loss = 'modified_huber', shuffle = True, random_state = 101)
    y_pred = my_model.fit(x_train, y_train).predict(x_test)


    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of SGDClassifier on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))


#Defining fuction for Decision Tree
from sklearn.tree import DecisionTreeClassifier
def training_function_DecisionTree(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = DecisionTreeClassifier(max_depth = 10, random_state=101,
                            max_features = None, min_samples_leaf = 15)
    y_pred = my_model.fit(x_train, y_train).predict(x_test)

    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of DecisionTree classifier on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))



#Defining fuction for Random Forest
from sklearn.ensemble import RandomForestClassifier
def training_function_RandomForest(x_train, y_train,x_test,y_test, print_flag=0):
    my_model = RandomForestClassifier(n_estimators = 70, oob_score = True,
                          n_jobs = -1, random_state = 101,
                          max_features = None, min_samples_leaf = 30)
    y_pred = my_model.fit(x_train, y_train).predict(x_test)


    if(print_flag == 1):
        print(('Pred values: %s'%y_pred))
        print(('Test values: %s'%y_test))
        print('Accuracy of Random forest classifier on test set: {:.2f}'.format(metrics.accuracy_score(y_test,y_pred)))
        print('-----------------------------------------------')


    return (metrics.accuracy_score(y_test,y_pred))
```

Part 3: Training and validation phase

```
#Defining a function in order to insert random noise into the training data set
# n - defines the level of noise
import numpy as np
def random_number_addition(n):
    data = X_train.copy()
    for i in range(len(X_train)):
        for j in range(len(X_train[i])):
            data[i][j] = X_train[i][j] + n*(1-2*np.random.rand())
    return data

#This function caluclates the average accuracy scores over 100 epochs for all the algorithms at every level of noise
# This function returns four types of values
# Level_error: List of Mean accuracies for every level of attribute noise for every algorithm

# Below three variables are used while plotting boxplots
# Accuracies: List of accuracy at every epoch for every level of attribute noise for every algorithm
# Level_name: List of attribute noise level at every epoch for every level of attribute noise for every algorithm
# Algo_name: List of algorithms used at every epoch for every level of attribute noise for every algorithm

def different_error_levels(input): #Input is upto how many levels of noise we want to get the results
    Accuracies = []
    Level_name = []
    Algo_name = []
    Algorithms = ['LogisticRegression','KNN','SVM','DecisionTree','RandomForest','SGDClassifier','GaussianNB','XGBoost']
    Level_error = [] # This is a list of lists which contains mean errors for each level of noise for all the used algorithms
    for i in range((input+1)):
        level = i
        k = []
        for cnt in range(100): # For every level of noise, code is run 100 times and its average is taken
            X_train_new = random_number_addition(level)
            Errors = [
            training_function_LogisticRegression(X_train_new,y_train,X_test,y_test).round(2),
            training_function_KNN(X_train_new,y_train,X_test,y_test,3).round(2),
            training_function_SVM(X_train_new,y_train,X_test,y_test,'linear').round(2) ,
            training_function_DecisionTree(X_train_new,y_train,X_test,y_test).round(2) ,
            training_function_RandomForest(X_train_new,y_train,X_test,y_test).round(2) ,
            training_function_SGDClassifier(X_train_new,y_train,X_test,y_test).round(2),
            training_function_GaussianNB(X_train_new,y_train,X_test,y_test).round(2) ,
            training_function_XGBoost(X_train_new,y_train,X_test,y_test).round(2)
            ]
            k.append(Errors)
            Accuracies.append(Errors)
            Level_name.append(['Level_'+str(level)]*8)
            Algo_name.append(Algorithms)

        mean_error = np.mean(k, axis = 0) #Calculates mean error for all algorithnms for a partular level of noise

        Level_error.append(mean_error)
    return Level_error,Accuracies,Level_name,Algo_name


#This code is used to get accuracy scores for various noise levels
# Calling the above function (NOTE: This will take time to execute based on number of levels used)

Total_error_levels = 10 #Change this variable to get data upto any level of noise
Level_error,Accuracies,Level_name,Algo_name = different_error_levels(Total_error_levels) # Calling the above function (NOTE: This will
take time to execute based on number of levels used)
Algorithms = [ 'Logistic Regression','KNN','SVM','DecisionTree','RandomForest','SGDClassifier','GaussianNB','XGBoost']
df = pd.DataFrame(Algorithms, columns =['Algorithms'])


#Printing the dataframe
for i in range(Total_error_levels+1):
    df['Level_'+str(i)] = Level_error[i].round(2)
df= df.set_index('Algorithms')
df
```

Part 4: Plotting the results

```
#Plotting the line graph for various noise levels
import matplotlib.pyplot as plt
n = Total_error_levels
plt.figure(figsize=(n+5,n/2))
for i in Algorithms:
    plt.plot(df.loc[i,:], label=i, linewidth=3)


plt.grid(which='major', linestyle='-', linewidth='0.5', color='blue')
plt.minorticks_on()
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.title("Accuracy of Algorithms",fontsize='20')
plt.xlabel('Error Levels',fontsize='17')
plt.ylabel('Accuracy',fontsize='17')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15),
          fancybox=True, shadow=True, ncol=5)
```

Part 5: Generating barplots

```
# The three variable generated in different_error_levels(input) function are combined to form a dataframe
dm = pd.DataFrame()
dm['Accuracies'] = sum(Accuracies,[])
dm['Levels'] = sum(Level_name,[])
```

```
dm['Algos'] = sum(Algo_name,[])
dm

#Selecting the levels of noise for box plots
#This piece of code needs user input

def get_n_levels():
  n_levels = input ("Enter the number of levels: "+str(Total_error_levels+1)+" :")
  levels = []
  for i in range(int(n_levels)):
    s = input("Enter the level number from 0 to "+str(Total_error_levels)+" :")
    levels.append("Level_"+str(s))
  return (levels)


ans = 's'
levels = []
while(ans != "Y" and ans !="EXIT"):
  lev = []
  lev = get_n_levels()
  print(lev)
  ans = input(print(''' Do you want above levels: Y/N
  OR
  Enter EXIT to move out. '''))
  ans = ans.upper()
  levels = lev

#Selecting the Algorithms for box plots
#This piece of code needs user input

Algos = {1:'LogisticRegression',
         2:'KNN',
         3:'SVM',
         4:'DecisionTree',
         5:'RandomForest',
         6:'SGDClassifier',
         7:'GaussianNB',
         8:'XGBoost'}
print("Following Algos are availale:")
for i in Algos:
  print(str(i)+':'+Algos[i])


def get_algos():
  B_algos = []
  n_algos = input ("Enter the number of algos: ")
  for i in range(int(n_algos)):
    s = int(input("Enter the algo number: "))
    B_algos.append(Algos[s])
  return B_algos


ans = 's'
Box_algos = []
while(ans != "Y" and ans !="EXIT"):
  b = []
  b = get_algos()
  print(b)
  ans = input(print(''' Do you want above Algorithms in boxplots: Y/N
  OR
  Enter EXIT to move out. '''))
  ans = ans.upper()
  Box_algos = b


print("Algos used in Box plots: ",Box_algos)
print("Levels of noise used in Box plots: ",levels)


#Selecting the data based on the levels and Algos to be used in boxplots
box_data = dm[dm['Algos'].isin(Box_algos) & dm['Levels'].isin(levels)]
box_data

#Plotting the box plots of the selected data
import seaborn as sns
plt.figure(figsize=(20,10))
sns.boxplot(y='Accuracies', x='Levels',
                data=box_data,
                palette="colorblind",
                hue='Algos')
```