Bahria University,
Karachi Campus

Course: CSC-221 Data Mining
Term: Spring 2020, Class: BSE- 6(B)

Project Report

"Best Algorithm Finder"

Submitted By:

Group Members:

- Gul Saba (02-131182-009)
- Qaiser Abbas (02-131182-030)
- Mahnoor Gohar (02-131182-023)

Submitted To:
Engr. Ramsha Mashood

Signed: ______ **Remarks:** ______ **Score:** ______

Table Of Content:

Contents

1. Introduction & Problem:

Our project is basically based on examining several ML algorithms for classification and clustering that could adjust to any Dataset and attempt to find which one is more accurate.

Our goal with this is to perform some initial data visualization and to determine which Algorithm handles any dataset the best.

Basically, as we see that in algorithm implementation on dataset, we are not able to predict which algorithm is best and more accurate for that particular dataset and it is time consuming to implement and test multiple algorithms, so this project is all about best algorithms finder and through this project we will be able to determine which algorithm is better and more accurate for dataset. That is why we decided to create this project.

Dataset Used:

In this project we can use any dataset from Kaggle. We have tested 2 to 3 datasets from Kaggle and perform classification and clustering analysis to determine best and accurate algorithms for that particular dataset.

- https://www.kaggle.com/aljarah/xAPI-Edu-Data
- https://www.kaggle.com/uciml/iris

Explanation Of Algorithms Used:

In this project we have used multiple classification and clustering algorithms which are discussed below.

Classification Algorithms:

Following classification algorithms are used to perform analysis on dataset.

➢ *Gradient Boosting Algorithm:*

Gradient Boosting builds an additive model in a forward stage-wise fashion; It allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

Pros:

Often provides predictive accuracy that cannot be beat. Lots of flexibility - can optimize on different loss functions and provides several hyperparameter tuning options that make the function fit very flexible. No data pre-processing required - often works great with categorical and numerical values as is.

Cons:

Gradient boosting is a greedy algorithm and can overfit a training dataset quickly.

Tips:

It can benefit from regularization methods that penalize various parts of the algorithm and generally improve the performance of the algorithm by reducing overfitting.

➢ *Bagging Classifier Algorithm:*

A Bagging Classifier is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

Pros:

Bagging takes the advantage of ensemble learning wherein multiple weak learners outperform a single strong learner. It helps reduce variance and thus helps us avoid overfitting.

Cons:

There is loss of interpretability of the model. There can possibly be a problem of high bias if not modelled properly. Another important disadvantage is that while bagging gives us more accuracy, it is computationally expensive and may not be desirable depending on the use case.

Tips:

Boosting and bagging are two ensemble methods capable of squeezing additional predictive accuracy out of classification algorithms. When using either method, careful tuning of the hyper-parameters should be done to find an optimal balance of model flexibility, efficiency & predictive improvement.

- ***Decision Tree Classifier Algorithm:***

Decision Tree Classifier is a simple and widely used classification technique. It applies a straightforward idea to solve the classification problem. Decision Tree Classifier poses a series of carefully crafted questions about the attributes of the test record. Each time it receives an answer, a follow-up question is asked until a conclusion about the class label of the record is reached.

Pros:

Compared to other algorithms decision trees requires less effort for data preparation during pre-processing. A decision tree does not require normalization of data. A decision tree does not require scaling of data as well. Missing values in the data also does NOT affect the process of building decision tree to any considerable extent. A Decision trees model is very intuitive and easy to explain to technical teams as well as stakeholders.

Cons:

A small change in the data can cause a large change in the structure of the decision tree causing instability. For a Decision tree sometimes, calculation can go far more complex compared to other algorithms. Decision tree often involves higher time to train the model. Decision tree training is relatively expensive as complexity and time taken is more. Decision Tree algorithm is inadequate for applying regression and predicting continuous values.

Tips:

Decision trees can become much more powerful when used as ensembles. Ensembles are clever ways of combining decision trees to create a more powerful model. These ensembles create state of the art machine learning algorithms that can outperform neural networks in some cases. The two most popular ensemble techniques are random forests and gradient boosting.

- ***AdaBoost Regressor Algorithm:***

An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.

Pros:

Very good use of weak classifiers for cascading; Different classification algorithms can be used as weak classifiers; AdaBoost has a high degree of precision; Relative to the bagging algorithm and Random Forest Algorithm, AdaBoost fully considers the weight of each classifier.

Cons:

Outliers will force the ensemble down the rabbit hole of working hard to correct for cases that are unrealistic. Noisy data, specifically noise in the output variable can be problematic.

Tips:

some heuristics for best preparing your data for AdaBoost: 1. Quality Data: Because the ensemble method continues to attempt to correct misclassifications in the training data, you need to be careful that the training data is of a high-quality. 2. Outliers: Outliers will force the ensemble down the rabbit hole of working hard to correct for cases that are unrealistic. These could be removed from the training dataset. 3. Noisy Data: Noisy data, specifically noise in the output variable can be problematic. If possible, attempt to isolate and clean these from your training dataset.

Clustering Algorithms:

Following clustering algorithms are used to perform analysis on dataset.

- ***K-Means Clustering Algorithm:***

K-Means clustering is an unsupervised learning algorithm that, as the name hints, finds a fixed number (k) of clusters in a set of data.

K-Means finds k number of centroids, and then assigns all data points to the closest cluster, with the aim of keeping the centroids small.

Pros:

Widely used method for cluster analysis.

Easy to understand and trains quickly.

Cons:

Euclidean distance is not ideal in many applications.

Performance is (generally) not competitive with the best clustering methods.

Small variations in the data can result in a completely different cluster (high variance).

- ***Agglomerative Clustering Algorithm:***

The agglomerative clustering is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It’s also known as AGNES (Agglomerative Nesting).

The algorithm starts by treating each object as a singleton cluster. Next, pairs of clusters are successively merged until all clusters have been merged into one big cluster containing all objects. The result is a tree-based representation of the objects, named dendrogram.

Pros:

Single-link algorithms are best for capturing clusters of different sizes and shapes.

It's also sensitive to noise.

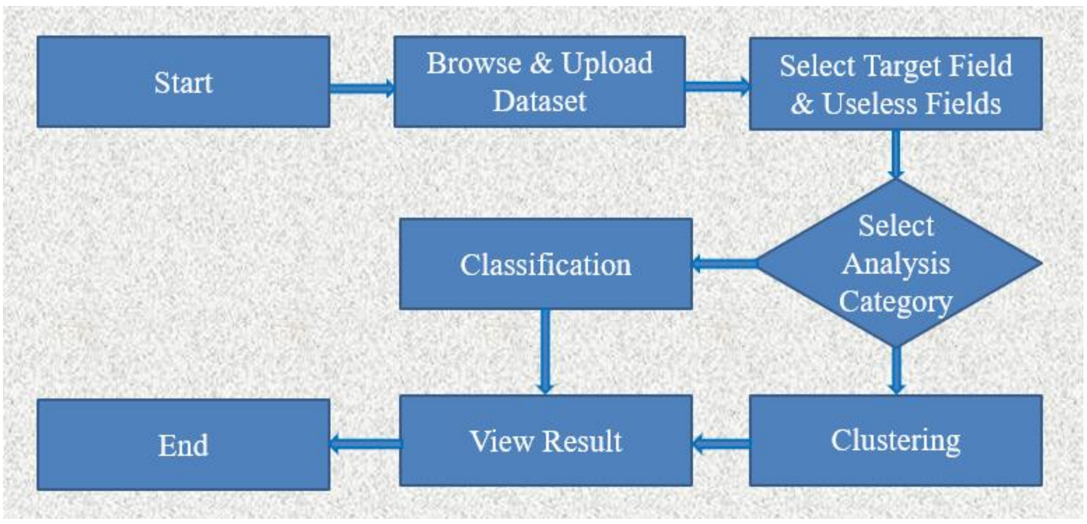
Complete link and group average are not affected by noise but have a bias towards finding global patterns.

Cons:

Only Single-Link is computationally possible for large datasets, but it doesn't give good results because uses too little information.

Overview & Flow Diagram:

- First, we will browse and upload any dataset file in csv, txt, tsv, xlsx format in order to read that dataset.
- Then we will choose analysis category (i.e., classification and clustering) in order to perform analysis on data.
- After that we will select target column and exclude useless data fields from dataset.
- Then we will view analysis results and will be able determine which ML Algorithm is best and more accurate for our dataset.



2. Project Scope:

In this **Best Algorithm Finder**, we have used different algorithms for classification and clustering to find the best algorithm among all for any dataset to find more accurate result. So, using this Project, we can overcome the problems that we can face during implementation of algorithms on any dataset and hence can reduce the chances of inaccurate results or less accurate results. So, the scope of our project is very vast as it will be beneficial to determine best and more accurate algorithm for any dataset. It will save time as we don't have to implement and test multiple algorithms one by one in order to find accurate results.

3. Technology:

- The technology used in this project includes Python language including following libraries: pandas, matplotlib, flask, numpy, sklearn, classifier, clusterer, scipy, preprocessor, algo_data.
- PyCharm is used as IDE to check the output and response.
- Datasets are available on Kaggle:
- https://www.kaggle.com/aljarah/xAPI-Edu-Data
- https://www.kaggle.com/uciml/iris
- For Classification Bagging Classifier, Gradient Boosting, AdaBoost Classifier, Ridge Classifier, Decision Tree Classifier and Extra Tree Classifier are used.
- For Clustering K-Means Clustering Algorithm and Agglomerative Clustering Algorithm are used.

4. Functionalities:

- We can read data from csv, txt, tsv, xlsx file.
- We can perform analysis on data through classification and clustering.
- We can select target column and exclude useless data fields from any dataset.
- We can determine which ML Algorithm is best and more accurate for our dataset.

5. Module Distribution:

- Mahnoor Gohar (Front end &Implementation of Clustering Algorithms)
- Gul Saba (Front end &Implementation of Classification Algorithms)
- Qaiser Abbas (Front end &Implementation of Classification Algorithms)

6. Code & Output:

- ***algo_data.py:***

```python
import pandas as pd

def getStats( algos ):
    data = pd.read_csv('DataMiningAlgoAnalysis.csv', header=0)
    data = data.to_dict('index')

    res = {}
    for d in data.keys():
        if data[d]['Name'] in algos:
            res[ data[d]['Name'] ] = data[d]
    return res

def test():
    algos = ['ExtraTreeClassifier' , 'AdaBoostClassifier']
    print( getStats(algos) )
```

- ***app.py:***

```python
import flask
import os
from flask import jsonify, request , render_template
from flask import flash, redirect, url_for, session
from joblib import load
from flask_cors import CORS, cross_origin
import requests, json
import pandas as pd
import requests
import random
from preprocessor import preprocess , findHeaderAndSEP , xnormalize
from classifier import classify
from clusterer import clustering , kmeans_cluster
from algo_data import getStats
import re

app = flask.Flask(__name__ ,
            static_url_path='',
            static_folder='static')

app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024
app.config["DEBUG"] = True
app.secret_key = 'super secret key'
cors = CORS(app, resources={r"/*": {"origins": "*"}})


MODELS = {}

@app.route('/test', methods=['GET','POST'])
def test():
    data = [ 1 , 2 , "ABC" , 3 , 4 , "XYZ" ]
    return jsonify( data )

ALLOWED_EXTENSIONS = set(['csv', 'txt', 'tsv', 'xlsx'])

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def secure_filename(filename):
    return 'data.txt'

@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        resp = jsonify({'message' : "WRONG FORM"})
        resp.status_code = 400
        return resp
    myfile = request.files['file']
    if myfile.filename == '' or not allowed_file(myfile.filename):
        resp = jsonify({'message' : "WRONG FORM"})
        resp.status_code = 400
        return resp
    else:
        filename = secure_filename(myfile.filename)
        myfile.save( os.path.join(app.config['UPLOAD_FOLDER'], filename) )
        resp = jsonify({'message' : 'File successfully uploaded' , 'filename' : filename})
        print("success" , resp)
        return redirect(url_for('details'))

@app.route('/details', methods=['GET'])
def details():
    fileloc = 'uploads/data.txt'
    f = open(fileloc , 'r+' , encoding='utf-8-sig')
    line = f.readline().strip()
    f.close()
    f = open(fileloc , 'r+')
    HEADER , SEP = findHeaderAndSEP(f)
    arr = []
    if SEP == '\s+':
        arr = re.split(SEP , line)
    else:
        arr = line.split(SEP)
    cols = pd.read_csv(fileloc , sep=SEP , header=HEADER).columns
    if HEADER is None:
        cols = [ i for i in range( len(arr) ) ]
    f.close()
    session['labels'] = list(cols)
    print(cols , HEADER , SEP , line)

    return render_template('form.html' , l = len(cols) , cols = cols)

@app.route('/predict', methods=['GET' , 'POST'])
def predict():

    if request.method == 'POST':
        antype = request.form['antype']
        TARGET = request.form['target']
        session['target'] = TARGET
        vals = request.form.keys()
        UNWANTED = []

        try:
            int(TARGET)
            TARGET = int(TARGET)
        except:
            pass

        for v in vals:
            if request.form[v] == 'on':
                UNWANTED.append(v)

        if antype == 'Clustering':
            data = preprocess('data.txt', None , UNWANTED )
            X_principal = xnormalize(data)
            result = clustering(X_principal)
            print(result)
            return render_template('cluster_analysis.html' , result = result)

        if antype == 'Association':
            data = preprocess('data.txt', None , UNWANTED )
            return "Under Construction"

        x_train,x_test,y_train,y_test = preprocess('data.txt', TARGET , UNWANTED )
        # print(y_test.dtype, "y_test")
        if y_test.dtype == 'float64':
            antype = 'Regression'

        if antype == 'Classification':
            result , models = classify( x_train,x_test,y_train,y_test )
            labels = list(result.keys())
            values = list(result.values())

            MODELS = models

            for i in range(4):
                values[i] = 100 * values[i]
            print(labels)
            print(values)
            analysis = getStats(labels)
            return render_template('classify_analysis.html' , labels = labels , values = values , analysis = analy
sis)
        else:
            return "Invalid Choice"

@app.route('/kmeans/<num>', methods=['GET'])
def km(num):
    num = int(num)
    data = preprocess('data.txt', None , [] )
    X_principal = xnormalize(data)
    km_name = kmeans_cluster(X_principal , num)
    return {"figure": "cluster/" + km_name}

@app.route('/testresult', methods=['POST'])
def test_result():
    labels = session['labels']
    testData = []
    print(request.form)
    return "done"

@app.route('/clustest', methods=['GET'])
def clustest():
    labels = session['labels']
    labels.remove( session['target'] )
    # labels = ["hel" , "temo"]
    result = {'dendo': 'cluster/dendo5.png', 'algo_3': 'cluster/aglo_33.png', 'algo_4': 'cluster/aglo_44.png', 'db
scan': 'cluster/dbscan14.png', 'kmean': 'cluster/kmeans8.png'}
    return render_template('cluster_analysis.html' , result = result , labels = labels)

@app.route('/', methods=['GET'])
def home():
    print("loaded")
    return render_template('index.html')
if __name__ == '__main__':
    app.run()
```

➢ ***classifier.py:***

```python
from sklearn import tree
from sklearn import ensemble
from sklearn import linear_model
from sklearn.metrics import accuracy_score

modelPack = {}

def trees( x_train, x_test, y_train, y_test ):

    res = []
    print("hello trees")

    m = tree.DecisionTreeClassifier()
    m.fit(x_train, y_train)
    print("fiting")
    predictions = m.predict(x_test)
    acc = accuracy_score(y_test,predictions)

    modelPack['DecisionTreeClassifier'] = m
    res.append( ( acc , "DecisionTreeClassifier" ) )

    m = tree.ExtraTreeClassifier()
    m.fit(x_train, y_train)
    predictions = m.predict(x_test)
    acc = accuracy_score(y_test,predictions)

    modelPack['ExtraTreeClassifier'] = m
    res.append( ( acc , "ExtraTreeClassifier" ) )

    print(res)
    return res

def ensembles( x_train, x_test, y_train, y_test ):
    res = []
    m = ensemble.AdaBoostClassifier()
    m.fit(x_train, y_train)
    predictions = m.predict(x_test)
    acc = accuracy_score(y_test,predictions)
    modelPack['AdaBoostClassifier'] = m
    res.append( ( acc , "AdaBoostClassifier" ) )

    # print(res)
    m = ensemble.BaggingClassifier()
    m.fit(x_train, y_train)
    predictions = m.predict(x_test)
    acc = accuracy_score(y_test,predictions)

    modelPack['BaggingClassifier'] = m
    res.append( ( acc , "BaggingClassifier" ) )
    m = ensemble.GradientBoostingClassifier()
    m.fit(x_train, y_train)
    predictions = m.predict(x_test)
    acc = accuracy_score(y_test,predictions)

    modelPack['GradientBoostingClassifier'] = m
    res.append( ( acc , "GradientBoostingClassifier" ) )
    return res

def lines( x_train, x_test, y_train, y_test ):

    res = []
    m = linear_model.RidgeClassifier()
    m.fit(x_train, y_train)
    predictions = m.predict(x_test)

    acc = accuracy_score(y_test,predictions)

    modelPack['RidgeClassifier'] = m

    res.append( ( acc , "RidgeClassifier" ) )

    m = linear_model.SGDClassifier()
    m.fit(x_train, y_train)
    predictions = m.predict(x_test)
    acc = accuracy_score(y_test,predictions)


    modelPack['SGDClassifier'] = m

    res.append( ( acc , "SGDClassifier" ) )

    return res

def classify( x_train, x_test, y_train, y_test ):

    result = {}

    r1 = trees( x_train, x_test, y_train, y_test )
    r2 = lines( x_train, x_test, y_train, y_test )
    r3 = ensembles( x_train, x_test, y_train, y_test )

    res = r1 + r2 + r3
    res.sort(reverse=True)
    print(res)

    models = {}
    for val , name in res[:4]:
        result[name] = val
        models[name] = modelPack[name]
    return result , models
```

➢ ***clusterer.py:***

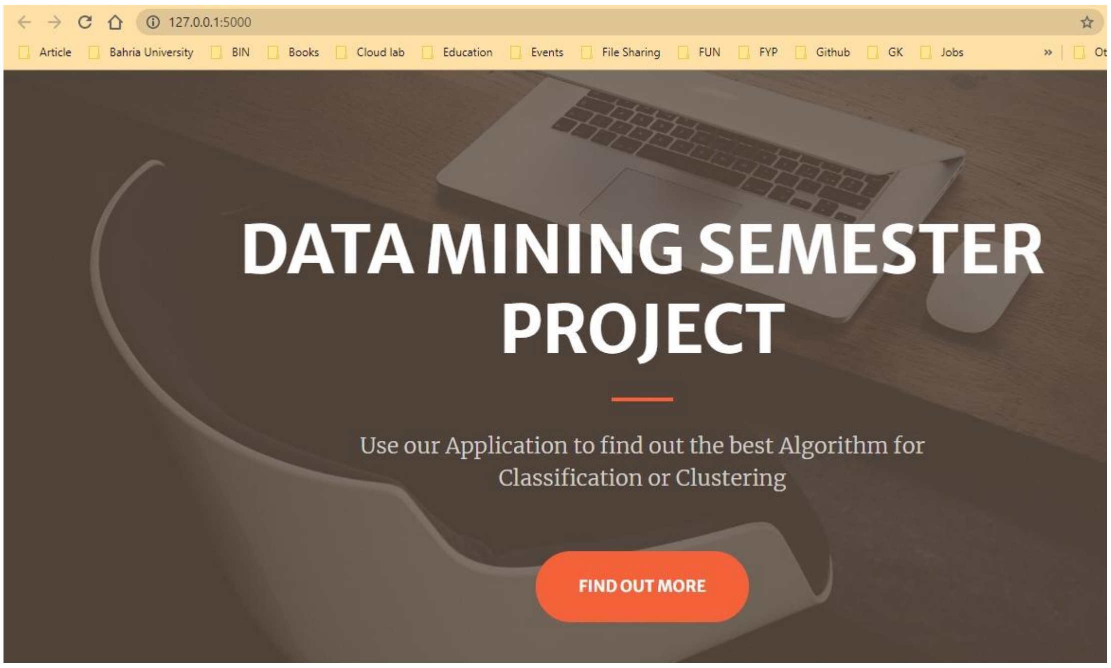
```python
import numpy as np
import pandas as pd
import matplotlib.colors as mcolors
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering , KMeans
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc
from sklearn.utils import resample
from subprocess import call
from random import random


def kmeans_cluster( X_principal , n_cluster):
    X = X_principal
    kmeans = KMeans(n_clusters=n_cluster, init='k-means++', max_iter=25, n_init=10, random_state=0)
    pred_y = kmeans.fit_predict(X)

    plt.figure(figsize =(6, 6))
    plt.scatter(X_principal['P1'], X_principal['P2'])
    plt.title('Number of Clusters = ' + str(n_cluster))
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
    name = 'kmean_clust' + str(int(random()*15)) + '.png'
    plt.savefig('static/cluster/'+name)
    return name

def kmeans(X_principal):
    wcss = []
    for i in range(1, 25):
        kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=15, n_init=10, random_state=0)
        kmeans.fit(X_principal)
        wcss.append(kmeans.inertia_)

    plt.figure(figsize =(6, 6))
    plt.plot(range(1, 25), wcss)
    plt.title('Elbow Method')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
    name = 'kmeans' + str(int(random()*15)) + '.png'
    plt.savefig('static/cluster/'+name)
    return name

def aglo(X_principal):
    #Dendogram
    plt.figure(figsize =(8, 8))
    plt.title('Visualising the data')
    Dendrogram = shc.dendrogram((shc.linkage( resample(X_principal, n_samples=350, random_state=0) , method ='ward
')))
    name1 = 'dendo' + str(int(random()*15)) + '.png'
    plt.savefig('static/cluster/'+name1)

    ac2 = AgglomerativeClustering(n_clusters = 3)

    # Visualizing the clustering
    plt.figure(figsize =(6, 6))
    plt.title('Number of Clusters = 3')
    plt.scatter(X_principal['P1'], X_principal['P2'],
            c = ac2.fit_predict(X_principal), cmap ='rainbow')

    name3 = 'aglo_3' + str(int(random()*15)) + '.png'
    plt.savefig('static/cluster/'+name3)

    ac2 = AgglomerativeClustering(n_clusters = 4)

    # Visualizing the clustering
    plt.figure(figsize =(6, 6))
    plt.title('Number of Clusters = 4')
    plt.scatter(X_principal['P1'], X_principal['P2'],
            c = ac2.fit_predict(X_principal), cmap ='rainbow')
    # plt.show()
    name4 = 'aglo_4' + str(int(random()*15)) + '.png'
    plt.savefig('static/cluster/'+name4)
    return name1 , name3 , name4

def clustering(X_principal):
    call('rm -r static/cluster/*.png',shell=True)
    dendo , algo_3 , algo_4 = aglo(X_principal)
    kmean = kmeans(X_principal)
    return { "dendo" : "cluster/"+dendo , "algo_3" : "cluster/"+algo_3 ,
             "algo_4" : "cluster/"+algo_4 , "kmean" : "cluster/"+kmean }
```

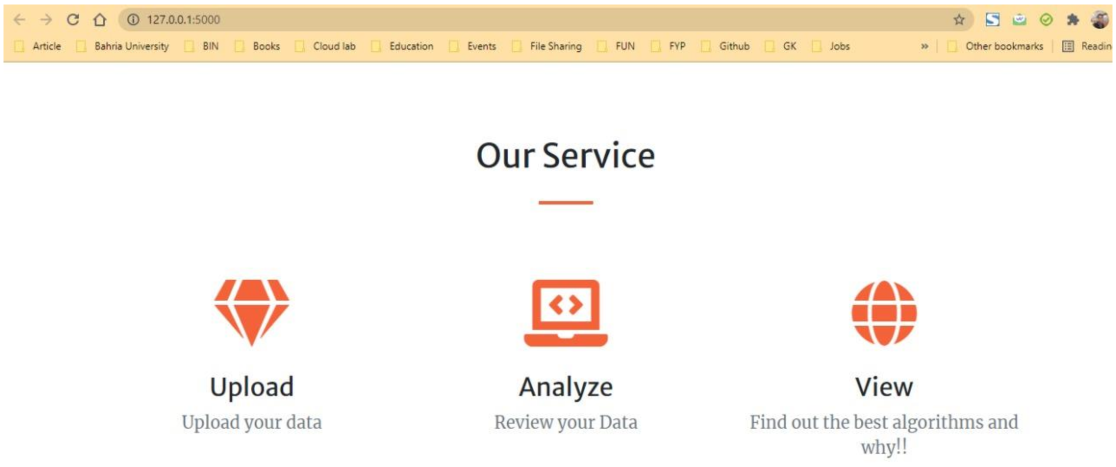
- ***preprocessor.py:***

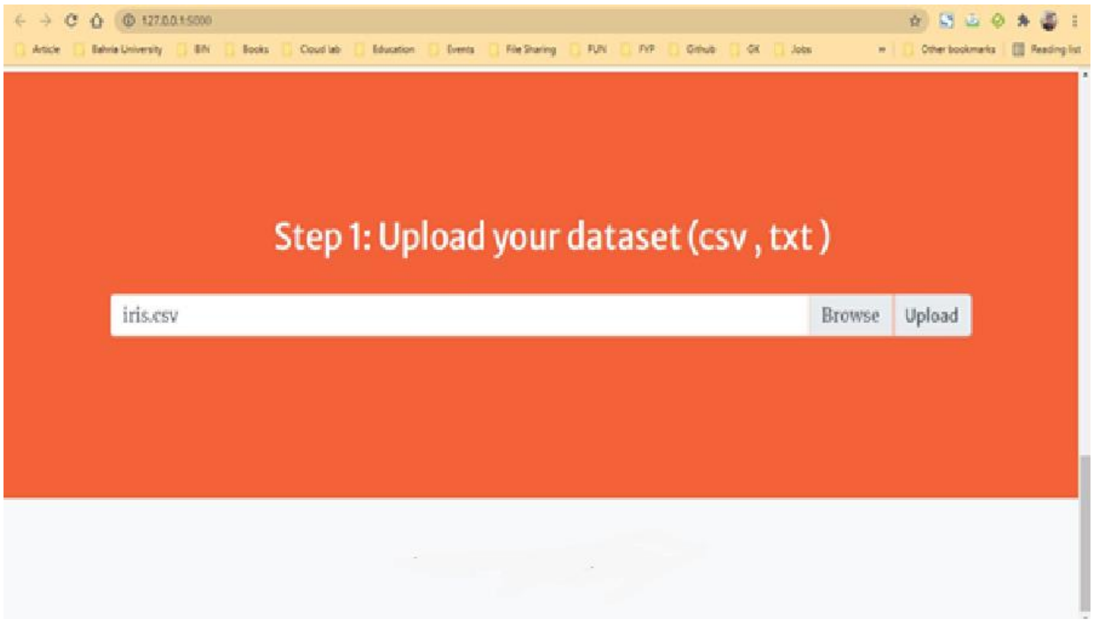
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA

def preprocess( FILE , TARGET , UNWANTED ):
    FILE = 'uploads/' + FILE
    f = open(FILE , 'r+')
    HEADER , SEP = findHeaderAndSEP(f)
    f.close()

    # Open File
    data = pd.read_csv(FILE , sep=SEP , header=HEADER )
    #Remove Unwanted
    data = data.drop(UNWANTED , axis=1)

    # Keep Data with Finite Target
    if TARGET is not None:
        try:
            data = data[ np.isfinite( data[TARGET] ) ]
        except:
            pass
    #Remove more than half missing data
    data = data.dropna(thresh=0.5,axis=1)

    #Seperating X and Y
    if TARGET is not None:
        Y = data[TARGET]
        X = data.drop([TARGET] , axis = 1)
        data = X
    # One Hot Encode String data
    def encode_and_bind(original_dataframe, feature_to_encode):
        dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
        res = pd.concat([original_dataframe, dummies], axis=1)
        res = res.drop([feature_to_encode] , axis=1)
        return res

    columns = data.columns
    for c in columns:
        if data[c].dtype == 'object':
            data = encode_and_bind(data , c)
    imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
    data = imputer.fit_transform(data)
    if TARGET is None:
        return data

    # Train-Test Split
    x_train,x_test,y_train,y_test = train_test_split( data ,Y,test_size=.34)
    return x_train,x_test,y_train,y_test

def findHeaderAndSEP(f):
    # Finds Seperator
    line = f.readline().strip()
    SEP = None
    if ',' in line:
        SEP = ','
    elif ':' in line:
        SEP = ':'
    elif ';' in line:
        SEP = ';'
```

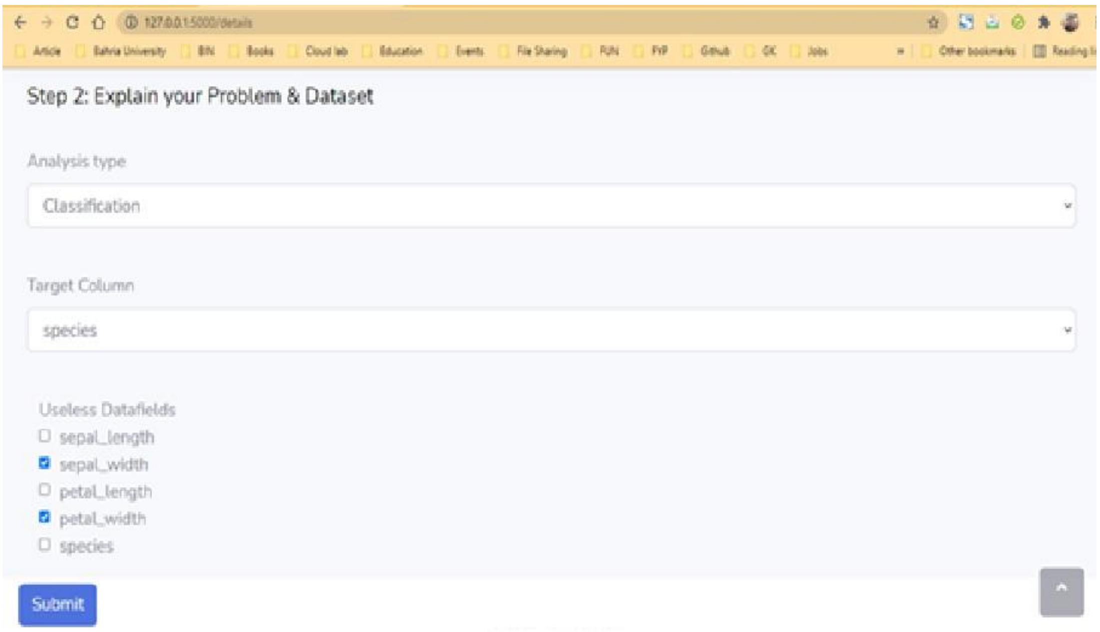
```python
    line1 = line.split(SEP)
    line2 = f.readline().strip().split(SEP)

    if SEP is None:
        SEP = '\s+'

    types1 = []
    types2 = []
    # Finds Header
    for l in line1:

        try:
            float(l)
            types1.append('float')
        except:
            types1.append('str')

    for l in line2:
        try:
            float(l)
            types2.append('float')
        except:
            types2.append('str')

    HEADER = None
    for a , b in zip( types1 , types2 ):
        if a != b:
            HEADER = 0
            break

    # print("HEADER, SEP", HEADER , SEP)
    return HEADER , SEP

def xnormalize(X):
    # Scaling the data to bring all the attributes to a comparable level
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Normalizing the data so that
    # the data approximately follows a Gaussian distribution
    X_normalized = normalize(X_scaled)

    # Converting the numpy array into a pandas DataFrame
    X_normalized = pd.DataFrame(X_normalized)

    pca = PCA(n_components = 2)
    X_principal = pca.fit_transform(X_normalized)
    X_principal = pd.DataFrame(X_principal)
    X_principal.columns = ['P1', 'P2']
    return X_principal

# preprocess('uploads/data.txt', 'quality' , [])
```

7. **Interfaces:**









Step 2: Explain your Problem & Dataset

Analysis type

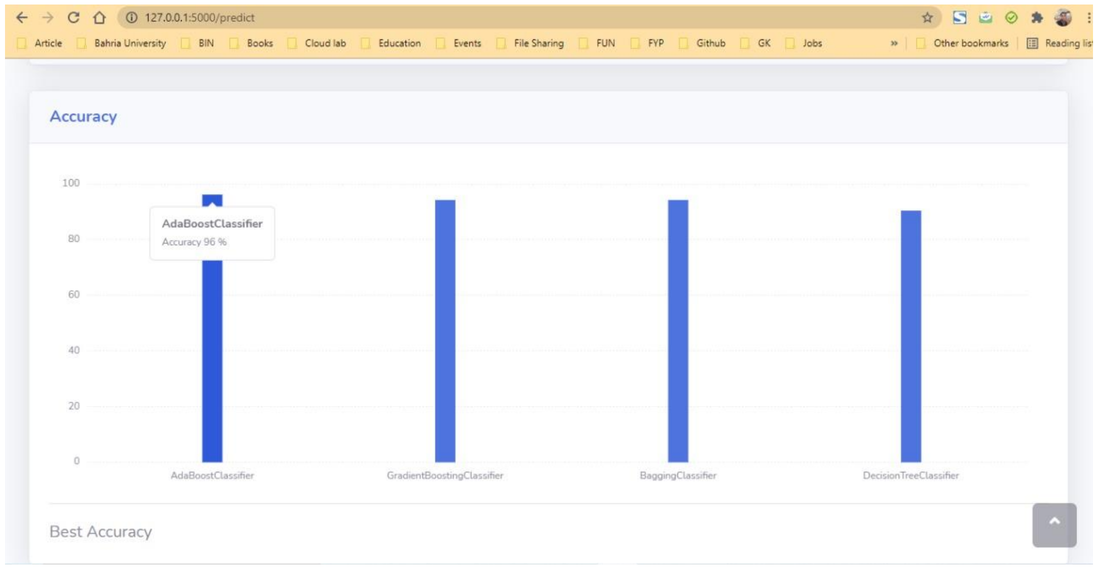
Classification

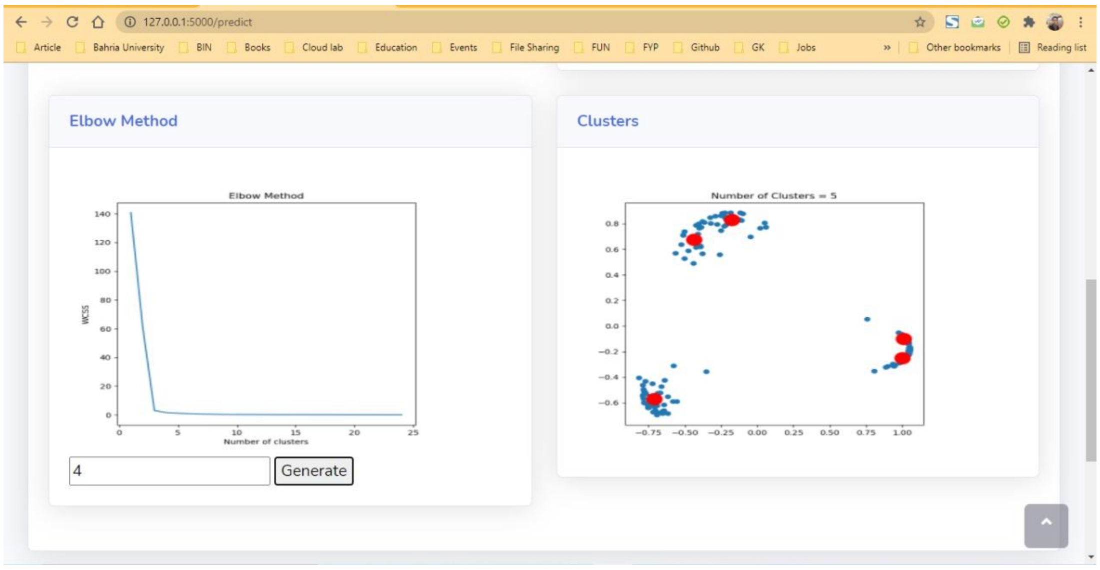
Target Column

species

Useless Datafields

- [ ] sepal_length
- [x] sepal_width
- [ ] petal_length
- [x] petal_width
- [ ] species





8. Conclusion:

In this project **Best Algorithm Finder**, we have used different algorithms to predict the accuracy of the given dataset for multiple features in order to determine which algorithm handles the dataset better. So, using this Project, we can overcome the problems that we can face during implementation of algorithms on any dataset and hence can reduce the chances of inaccurate results or less accurate results.