Differential Evolution
For
Feature Engineering

By,
Pritish
16MT10030
Department of Metallurgical and Materials Engineering
IIT Kharagpur

<u>Introduction</u>

In this project we are going to introduce a new way to **<u>create new features</u>** in a dataset for machine learning algorithms. We will find **<u>interaction between features using differential evolution</u>**. We have applied the differential evolution in its crude form which give us a lot of future scope for improvement also.

<u>What are we going to do:</u>

1. *Differential Evolution*
2. *Dataset Used*
3. *Implementation*
4. *Result*
5. *Future Scope*

<u>Differential Evolution</u>

Differential Evolution (DE) is a vector population based stochastic optimization method which has been introduced in 1995 by Storn and Price. Like genetic algorithm (GA), this method is able to optimize objective functions which are function of discrete variables.

An unconstrained optimization problem can be stated as follows:

Find X = (x1,x2,...xn) which minimizes f (X)

where X is an n-dimensional vector called design vector and f (X)
is the objective function.

Differential evolution can be briefly explained as follows:

The Population

Differential evolution is a population based optimization method. Assume the population contains Np individuals. $X_{i,g}$,is the ith individual of gth generation of the population. The first population is selected randomly in differential evolution. Figure 1 shows the first random population in a two dimensional problem

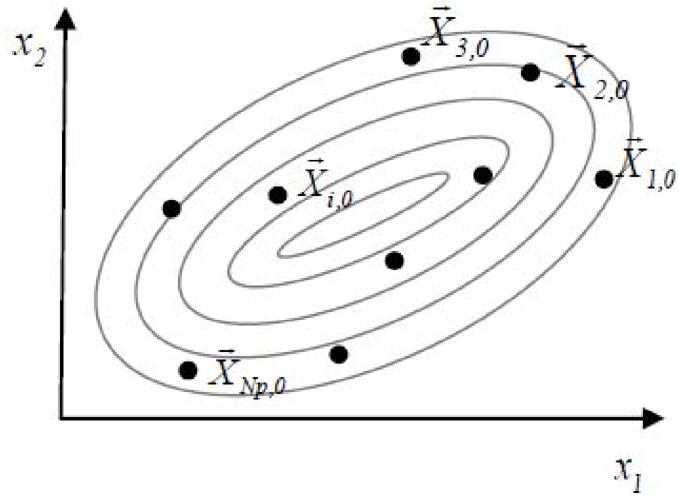


Figure.1 The first random population in a two dimensional problem

The Mutation

There are several techniques for mutation of individuals in differential evolution. In general, the mutant individual can be defined as follows:

$$V_{i,g} = Y_{i,g} + F.(1/N).\sum_{n=0}^{N-1}(X_{r(2n+1),g} - X_{r(2n+2),g})$$

Where **$Y_{i,g}$ is the base vector** and **F is a constant parameter called mutation scale factor** and subscript r shows that the individual is selected randomly in the population. Based on this general equation, there are four mutation techniques which are very popular in the literature:

DE/rand/1/bin: $\vec{V}_{i,g} = \vec{X}_{r0,g} + F.\left(\vec{X}_{r1,g} - \vec{X}_{r2,g}\right)$

DE/best/1/bin: $\vec{V}_{i,g} = \vec{X}_{best,g} + F.\left(\vec{X}_{r1,g} - \vec{X}_{r2,g}\right)$

DE/current-to-best/1/bin: $\vec{V}_{i,g} = \vec{X}_{i,g} + F.\left(\vec{X}_{best,g} - \vec{X}_{i,g}\right) + F.\left(\vec{X}_{r1,g} - \vec{X}_{r2,g}\right)$

DE/best/2/bin: $\vec{V}_{i,g} = \vec{X}_{best,g} + F.\left(\vec{X}_{r1,g} - \vec{X}_{r2,g} + \vec{X}_{r3,g} - \vec{X}_{r4,g}\right)$

where **$X_{best,g}$** is the individual which has the best fitness in the population.

The Crossover

The most common crossover in differential evolution is uniform crossover which can be defined as follows:

$$\vec{U}_{i,g} = u_{j,i,g} = \begin{Bmatrix} v_{j,i,g} & if & r_j \le Cr \ or \ j = j_{rand} \\ x_{j|i,g} & if & r_j > Cr \end{Bmatrix} \quad j = 1..n$$

The Selection

The final step in DE algorithm is the selection of the better individual for the minimization of the objective function f (X). This process can be defined as follows

$$\vec{X}_{i,g+1} = \begin{Bmatrix} \vec{U}_{i,g} & if & f(\vec{U}_{i,g}) \le f(\vec{X}_{i,g}) \\ X_{i,g} & if & f(\vec{U}_{i,g}) > f(\vec{X}_{i,g}) \end{Bmatrix}$$

The selection process involves a simple replacement of the original individual with the obtained new individual if it has a better fitness.

There are three control parameters in the DE algorithm: the **mutation scale factor F**, the **crossover constant Cr**, and the **population size Np**. Storn and Price suggested the following values for these control variables:

$$F \in [0.5, 1.0]$$
$$Cr \in [0.8, 1.0]$$
$$Np \approx 10.n$$

Differential Evolution Algorithm

Below shows the differential evolution algorithm

Step 1: Select Np individuals $\vec{X}_{i,g}$ randomly.

Step 2: For $i = 1$ to Np let $f_i = f(\vec{X}_{i,g})$

Step 3: While (convergence criterion not yet met) do steps 4 to 10

Step 4: For $i = 1$ to Np do steps 5 to 10

Step 5: Select three different random indexes r_0, r_1 and r_2 between 1 to Np $(i \neq r_0 \neq r_1 \neq r_2)$

Step 6: Let $\vec{V}_{i,g} = \vec{X}_{r0,g} + F.(\vec{X}_{r1,g} - \vec{X}_{r2,g})$

Step 7: For $j = 1$ to n do steps 8 to 9

Step 8: Select randomly r_j variable $(0 \leq r_j < 1)$ and j_{rand} index $(1 \leq j_{rand} \leq \text{n})$

Step 9: If $r_j \leq Cr$ or $j=j_{rand}$ then $u_{j,i,g} = v_{j,i,g}$ else $u_{j,i,g} = x_{j,i,g}$

Step 10: If $f(\vec{U}_{i,g}) \leq f_i$ then $\vec{X}_{i,g+1} = \vec{U}_{i,g}$; $f_i = f(\vec{U}_{i,g})$ else $\vec{X}_{i,g+1} = \vec{X}_{i,g}$

Dataset Used

Dataset: Churn Dataset for telecom industry

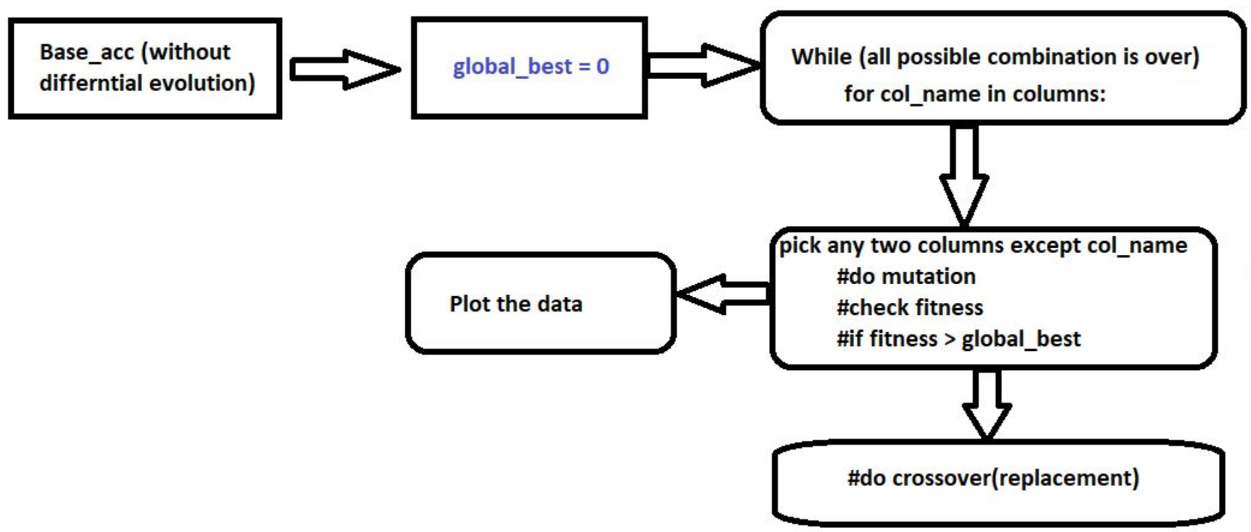
Features: Id, state, account_length, area_code, phone_number, international_plan, voice_mail_plan, number_vmail_messages, total_day_minutes, total_day_calls, total_day_charge, total_eve_minutes, total_eve_calls, total_eve_charge, total_night_minutes, total_night_calls, total_night_charge, total_intl_minutes, total_intl_calls, total_intl_charge, number_customer_service_calls

Output: Churn(**Binary: False or True**)

Detail	#
Cross Validation Used	5 Fold validation
Classifier	DecisionTreeClassifier(**Default**)
Total Data points	5000
Training Size(Population)	4000(in each pass of 5 Fold Validation)
Validation Set	1000(in each pass of 5 Fold Validation)

Implementation

Detail	#
Fitness Function	Accuracy of Classifier
F **(mutation scale factor)**	0.1 - 0.9
Preprocessing	convert_to_numeric_value



In this two variation has been used:
- **Global_best = 0 (initially)**
- **Global_best = Base_acc (initially)**

Preprocessing function

```python
#source pythonprogramming.com (sentdex)
def handle_non_numeric_data(df,show_mapping=False):

    columns = df.columns.values

    def convert_to_numeric_value(val):
        #print (val)
        return temp[val]

    for column in columns:
        temp ={}
        i=0
        if df[column].dtype != np.float64 and df[column].dtype != np.int64:
            column_contents = df[column].values.tolist()
            column_contents = set(column_contents)
            #print(column_contents)
            for content in column_contents:
                temp[content] = i
                i+=1
            df[column] = list(map(convert_to_numeric_value,df[column]))
            if show_mapping == True:
                print(temp)
```

Fitness Function code screenshot can be seen below

```python
def check_fitness(X,y,clf=None):
    if clf == None:
        clf = DecisionTreeClassifier
    kf = KFold(n_splits=5,random_state=42)
    avg = []
    #X_train,X_test,y_train,y_test = cross_validation.train_test_split(X,y,test_size =.4,random_state=1)
    for train_index, test_index in kf.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]

        reg = clf().fit(X_train,y_train)

        y_predicted = reg.predict(X_test)

        """print ("Classification report for %s" % reg)
        print (metrics.classification_report(y_test, y_predicted))
        print ("Confusion matrix")
        print (metrics.confusion_matrix(y_test, y_predicted))"""

        accuracy = reg.score(X_test,y_test)
        avg.append(accuracy)

    print('accuracy :: ',avg)
    return np.mean(avg)
```

Plotting Function screenshot can be seen below

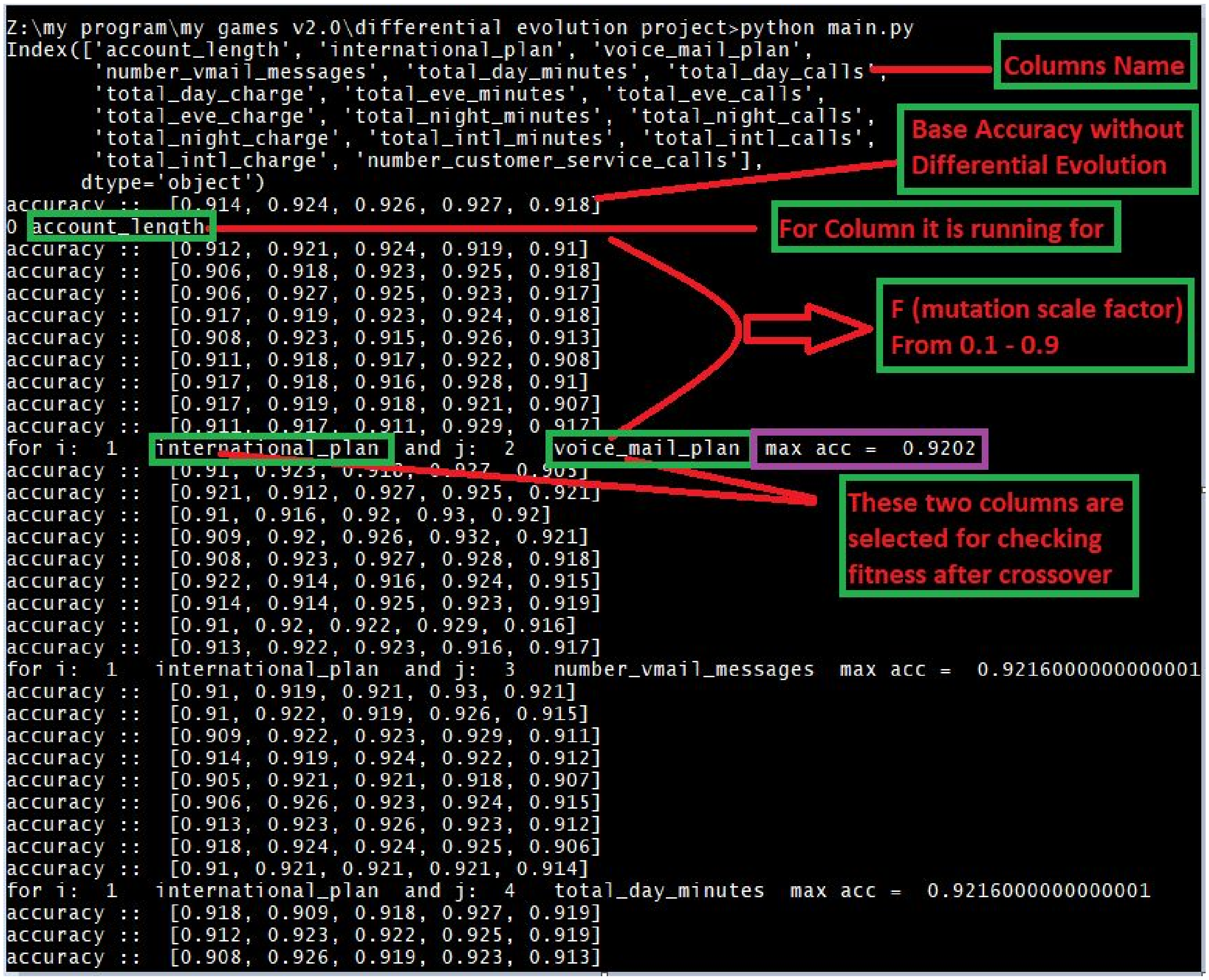
```
plt.scatter(x, y)

#plt.scatter(base_x,base_y,color='green',s=70)
plt.plot(base_x,base_y,color='green')

plt.scatter(high_x,high_y,color='red',s=70)
plt.plot(high_x,high_y,color='black')
plt.pause(.1)
plt.savefig('graphs/'+file_name)
if a+1 != d:
    plt.clf()
else:
    if b != a-1:
        plt.clf()
    else:
        if c != b-1:
            plt.clf()
```

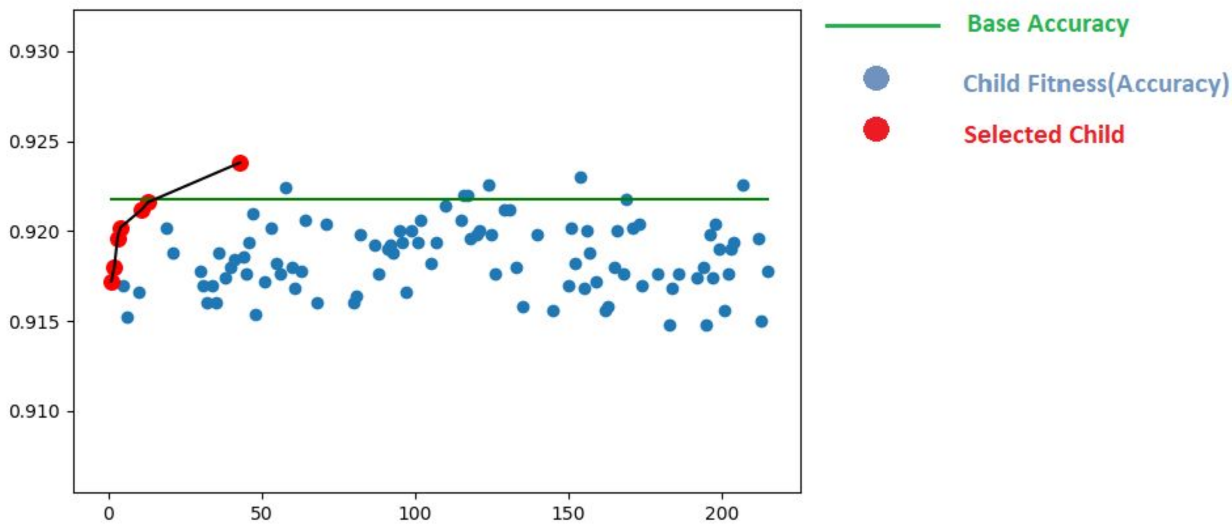
Result

- Console output
  - It shows output for all corresponding intermediate state



```
Z:\my program\my games v2.0\differential evolution project>python main.py
Index(['account_length', 'international_plan', 'voice_mail_plan',
       'number_vmail_messages', 'total_day_minutes', 'total_day_calls',
       'total_day_charge', 'total_eve_minutes', 'total_eve_calls',
       'total_eve_charge', 'total_night_minutes', 'total_night_calls',
       'total_night_charge', 'total_intl_minutes', 'total_intl_calls',
       'total_intl_charge', 'number_customer_service_calls'],
      dtype='object')
accuracy ::  [0.914, 0.924, 0.926, 0.927, 0.918]
0 account_length
accuracy ::  [0.912, 0.921, 0.924, 0.919, 0.91]
accuracy ::  [0.906, 0.918, 0.923, 0.925, 0.918]
accuracy ::  [0.906, 0.927, 0.925, 0.923, 0.917]
accuracy ::  [0.917, 0.919, 0.923, 0.924, 0.918]
accuracy ::  [0.908, 0.923, 0.915, 0.926, 0.913]
accuracy ::  [0.911, 0.918, 0.917, 0.922, 0.908]
accuracy ::  [0.917, 0.918, 0.916, 0.928, 0.91]
accuracy ::  [0.917, 0.919, 0.918, 0.921, 0.907]
accuracy ::  [0.911, 0.917, 0.911, 0.929, 0.917]
for i:  1   international_plan  and j:  2   voice_mail_plan  max acc =  0.9202
accuracy ::  [0.91, 0.923, 0.918, 0.927, 0.905]
accuracy ::  [0.921, 0.912, 0.927, 0.925, 0.921]
accuracy ::  [0.91, 0.916, 0.92, 0.93, 0.92]
accuracy ::  [0.909, 0.92, 0.926, 0.932, 0.921]
accuracy ::  [0.908, 0.923, 0.927, 0.928, 0.918]
accuracy ::  [0.922, 0.914, 0.916, 0.924, 0.915]
accuracy ::  [0.914, 0.914, 0.925, 0.923, 0.919]
accuracy ::  [0.91, 0.92, 0.922, 0.929, 0.916]
accuracy ::  [0.913, 0.922, 0.923, 0.916, 0.917]
for i:  1   international_plan  and j:  3   number_vmail_messages  max acc =  0.9216000000000001
accuracy ::  [0.91, 0.919, 0.921, 0.93, 0.921]
accuracy ::  [0.91, 0.922, 0.919, 0.926, 0.915]
accuracy ::  [0.909, 0.922, 0.923, 0.929, 0.911]
accuracy ::  [0.914, 0.919, 0.924, 0.922, 0.912]
accuracy ::  [0.905, 0.921, 0.921, 0.918, 0.907]
accuracy ::  [0.906, 0.926, 0.923, 0.924, 0.915]
accuracy ::  [0.913, 0.923, 0.926, 0.923, 0.912]
accuracy ::  [0.918, 0.924, 0.924, 0.925, 0.906]
accuracy ::  [0.91, 0.921, 0.921, 0.921, 0.914]
for i:  1   international_plan  and j:  4   total_day_minutes  max acc =  0.9216000000000001
accuracy ::  [0.918, 0.909, 0.918, 0.927, 0.919]
accuracy ::  [0.912, 0.923, 0.922, 0.925, 0.919]
accuracy ::  [0.908, 0.926, 0.919, 0.923, 0.913]
```

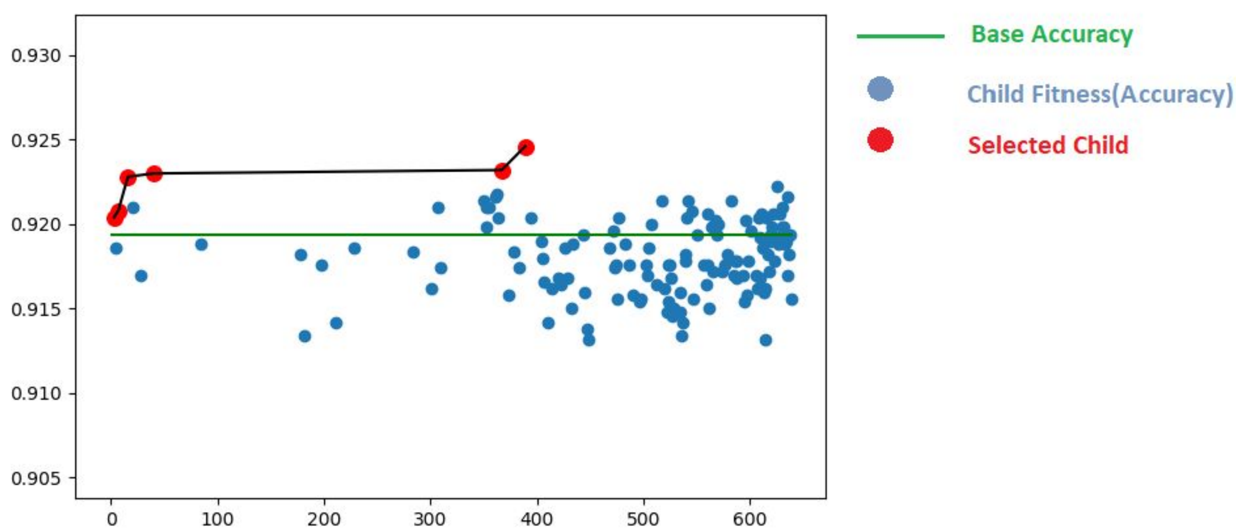
- Graph
  - **For Global_best = 0 initially**



Global_best is intially = 0

Global_best is updated every time if we get fitness greater than previous Global_best

  - **For Global_best = Base_accuracy initially**



Global_best is intially = Base_Acc

Global_best is updated every time if we get fitness greater than previous Global_best

Good results were obtained in both cases:

- Increase in accuracy was observed
- Upto 5% increase in accuracy wa observed
- New features were helpful to cover interaction between different columns.
- With models like SVM lot of improvement was observed

Future Scope

- Can use improved variant of differential evolution for good and fast convergence.
- Can use different algorithm first to decide importance of features so that we can run our algorithm on it first.
- Can find feature depence on each other so that instead of using brute force we can run our algorithm on certain feature only.