Error Generator

Table of content

Problem statement **2**

Dimensions of the problem 2

Implementation **4**

Change feature one by one 4

Change feature Randomly 4

Optimize Random 5

Most important feature 5

Check all combination: 6

Strategies: **6**

Algorithm 1 - Change_Combination_Min 6

Algorithm 2 - Change_Combination_Feature_Min 6

Algorithm 3 - Change_Uncertainty_Rankfeatures 7

Algorithm 4 - Change_ProbabilityDistance_RankFeature 7

How to use the strategies? **7**

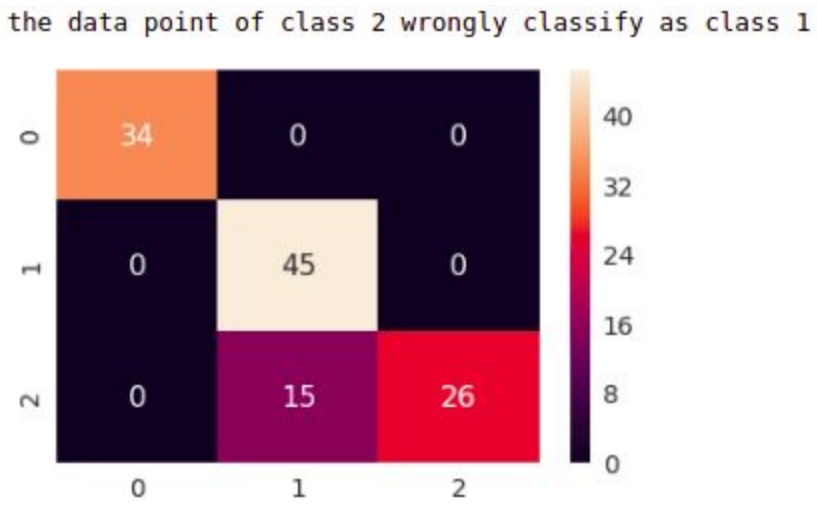
Experiments & Some hints **8**

Problem statement

Machine learning practitioners need to know how their machine learning models act in case of errors. Therefore, we provide them with a framework that allows them to design different error stress test cases. Stress tests allow them to better understand the weaknesses of their model and features.

For example, if we consider the Iris classification task and take a look at the confusion matrix, we see that the classifier wrongly classifies 15 data points to class one instead of the right class which is two.

As a result, generating errors in the way that is close to the weakness of the classifier, can help machine learning scientist to robust their algorithm against this misclassifying.



Dimensions of the problem

- ➔ what kind of machine learning task?
  - ◆ Classification
    - Binary classification
    - Multi-class classification
  - ◆ Regression
  - ◆ Clustering

- ➔ Which kind of Attribute/Feature Type?
  - ◆ Categorical
  - ◆ Numerical
  - ◆ Mixed

- ➔ Resulting question: do we apply feature transformations are not?
- ➔ Do we need a feature transformation?
  - ◆ Yes
    - The user should provide all feature transformations that are used for the task
  - ◆ No
    - Consider default transformer (I.e. word2vec for text)
    - We can work only on numeric data

- ➔ How we can select the tuple?
  - ◆ Random
    - All type of random number generator can include:
  - ◆ The user can specify tuples
  - ◆ Tuples that belong to a certain class label
  - ◆ Tuples that need minimum feature change for changing their label
  - ◆ Most important tuples of the training set (see instance selection)

- ➔ Should we change one feature or multiple features in the selected tuples?
  - ◆ One feature
    - One feature randomly
    - The most important feature for the machine learning task, while changing as few features as possible that result in target change
  - ◆ Several features
    - How many?
    - Several random features
    - The most important combination of the features for the machine learning task, with minimum variation, change the target

- ➔ How should we change the selected features?
  - ◆ Empty it
  - ◆ Inject it with our API
    - Missing value
      - Explicit Missing Value
      - Implicit Missing Value
    - Switch value
      - Randomly Active Domain
      - Similar Based Active Domain
    - Typo
      - Typo Butterfinger
      - Typo Keyboard
    - Semantic
      - Word2vec
    - Noise
      - Gaussian Noise
      - White Noise

- ➔ Which measurement?
  - ◆ Accuracy
  - ◆ F-1

- ➔ What is our assumption?
  - ◆ Classification Task (multi-class)
  - ◆ Numerical Features
  - ◆ We consider the following user constraint :
    - Accuracy drop
    - Number of feature changes per instance
    - Total number of feature changes
    - Type of the changes
    - How the change should affect the prediction (target label)

Implementation

Change feature one by one

In this implementation, we ask the user to provide us with a dictionary that contains which and how many labels have to be changed to which other labels:

```
change_plan={"key":[[0,1],[2,1]],"number":[3,3]}
```

In this implementation, we randomly pick the rows. For the selected rows, we start from feature one and make it zero and then apply the classifier to this new data point. If the label changed in the way that the user requested, the instance is changed successfully. Otherwise, we change the next feature and continue this loop until we reach the desired prediction change. If the user-desired changes are not possible, we raise an exception.

```
Pick row randomly:
   For the selected row:
      If the real label == request of the user
         If Change feature one can check the label:
            If new predict == request of user:
               Done
     else:
           Change the other features respectively
   Pick another row
```

Change feature Randomly

In this implementation, a change plan is needed like before.
We randomly pick the row, this time features also pick randomly. The selected feature has changed to zero and label check if the label changed, one change did otherwise another feature will be selected until the label change. If we can't change the label by changing one feature randomly and we meet all features another row will be selected. This process will continue until the requested number of the change happen or exception raise.

```
Pick row randomly:
   For the selected row:
        If the real label == request of the user
           Pick feature randomly
              If change feature can change label:
                 If new predict == request of user:
                     Done
              else:
                  Pick another feature randomly
   Pick another row
```

Optimize Random

In this implementation, a change plan is needed like before. We first index all rows that classifier classifies them as same as the request of the user then we only randomly pick one feature and change it until the label change and equal to the request of the user. If we can't change the label by changing one feature randomly and we meet all features already, another row will be selected from our index list.

```
Pick row according to user request :
   For the selected row:
      Pick feature randomly
           If change feature can change label:
                If new predict == request of user:
                    Done
           else:
                  Pick another feature randomly
 Pick another row
```

Most important feature

In this implementation, a change plan is needed like before. According to a library of Sequential Feature Selector, we find the most important feature and use the indexed list to select a row. This time the feature that we want to change it is fixed. We use the most important feature and make it zero. If this variation changes the label we did one change otherwise we pick another row from our list. This process will continue until the requested number of the change happen or exception raise.

```
Pick row according to user request :
   For the selected row:
      Pick the most important feature
           If change feature can change label:
                If new predict == request of user:
                    Done
           else:
                  Pick another row
```

➔ What we understood from implementation:

- We shouldn't ask the user for a change plan When the number of features increases, changing only one feature is not enough for changing the label. (Therefore, we need to change more than one feature!)
- For select the row we should once index the rows and labels and then start our task

Check all combination:

```
 Loop on the instances (i) that satisfy the user constraints:
    Loop on all combination (c) of the features for changing the target:
        new_instance = i.apply_changes(c)
            If prediction(new_instance) satisfies user constraints:
               Done!
```

Strategies:

Algorithm 1 - Change_Combination_Min

check all possible combination for changing the feature and then apply the combination that required the minimum number of the change for satisfying the user request. To make it clear, first rows that require only one change for changing the target and then rows by two feature and so on will select by the algorithm.

```
 Loop on the instances (i) that satisfy the user constraints:
    Loop on all combination (c) of the features for changing the target:
        new_instance = i.apply_changes(c)
            If prediction(new_instance) satisfies user constraints:
                    Store number of changes that require for changing each row
    Apply changes according to the minimum number of changes
```

Algorithm 2 - Change_Combination_Feature_Min

For each row, for each combination (from len1 to N) change the feature and check the target changes. first, check all rows that can change them only with changing one feature then two feature and so on.

```
Loop on the instances (i) that satisfy the user constraints(rows):
       Loop on all combination (c) of the features for changing the target
       (start from length 1 to N):

       new_instance = i.apply_changes(c)
             If prediction(new_instance) satisfies user constraints:
                    Done!- stop checking
```

Algorithm 3 - Change_Uncertainty_Rankfeatures

This method first sorts the rows according to uncertainty and features according to information gain and then pick first rows that classifer is not sure about them and features are more likely to change.

```
Sort the rows according to uncertainty
Sort the features according to information gain
loop on the instances (i) that satisfy the user constraints(sorted rows):
       Loop on all combination (c) of the sorting features for changing the target
       (start from length 1 to N):
              new_instance = i.apply_changes(c)
              If prediction(new_instance) satisfies user constraints:
                    Done!- stop checking
```

Algorithm 4 - Change_ProbabilityDistance_RankFeature

This method sorts the rows according to the user request and finds the difference of the probability for each class and then sorts the rows according to them.

for example, if we have [0.1 0.6 0.3] as a probability for three class classification and request of a user be [0,1] we compute 0.5 for this row and sort the row Ascending.

```
Sort the rows according to distance probability
Sort the features according to information gain
loop on the instances (i) that satisfy the user constraints(sorted rows):
       Loop on all combination (c) of the sorting features for changing the target
       (start from length 1 to N):
              new_instance = i.apply_changes(c)
              If prediction(new_instance) satisfies user constraints:
                    Done!- stop checking
```

How to use the strategies?

In the project, there is a user_interface_accuracy_drop.py which is the user interface of the project.

- As the first step, you need to define the loader to load your .csv file. Some predefine dataset is available in the project.

```
loader=Iris_Loader()
loader=Abalone_loader()
loader=Digits_Loader()
loader = EEG_Loader()
x_train, x_test, y_train, y_test = loader.load()
```

- Train the model is the second step, train the model and report the accuracy

```
model = MultinomialNB()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
```

- As next step, pick your strategy

```
mymethod =Change_Combination_Min()     #Alg1
mymethod=Change_Combination_Feature_Min() #Alg2
mymethod = Change_Uncertainty_Rankfeatures() #Alg3
mymethod = Change_ProbabilityDistance_RankFeature() #Alg4
```

- Define your change plan
  For example, change the class 4 to 2 forty times and class 3 to 5 ten times.

```
change_plan={"key":[[4,2],[3,5]],"number":[40,10]}
```

- Run the strategy

```
out=mymethod.change(x_test,y_test,Percntage,model,change_plan)
```

- Put the dirty version of the dataset for evaluation

```
y_pred=model.predict(out)
```

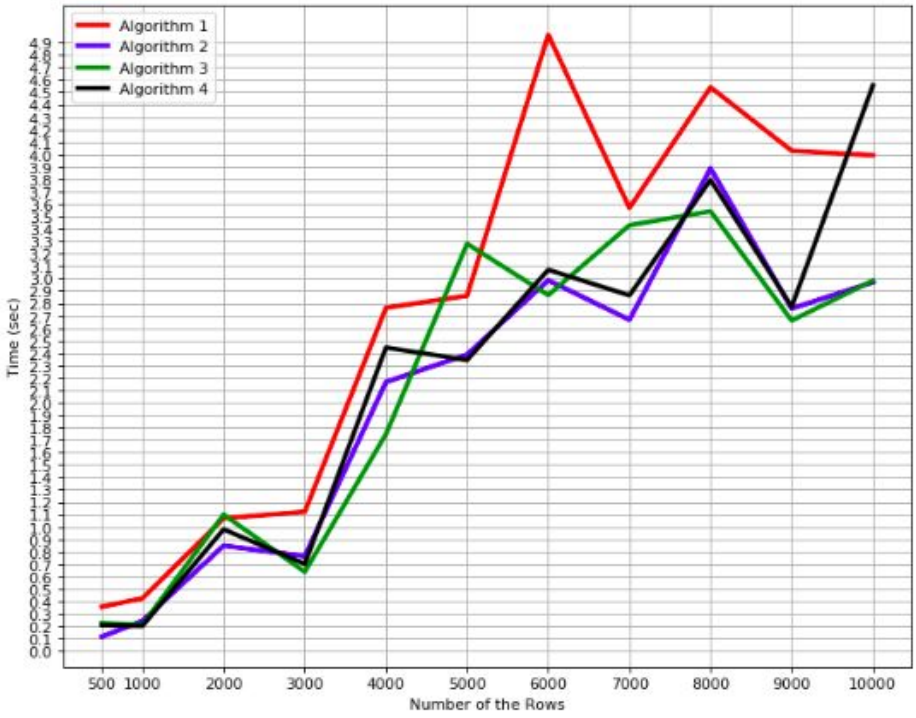
- Report the result

```
print ("\n Accuracy is: {:.2f} ".format(accuracy_score(y_test, y_pred)))
```

Experiments & Some hints

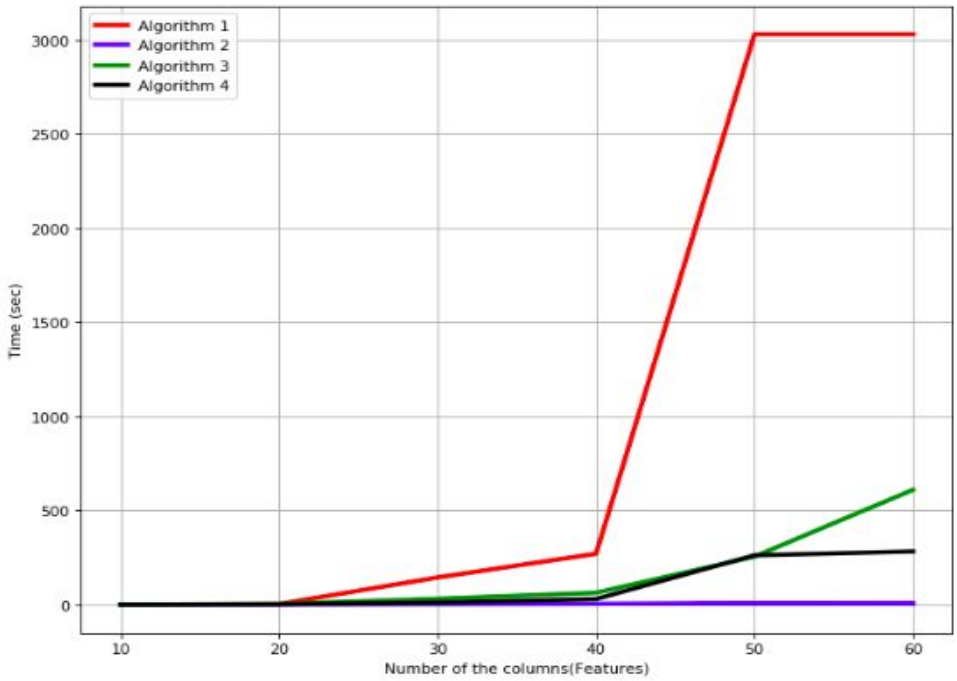
The following experiments done on the EEG dataset which is availabe in the GitHub directory of the project.
In the first experiment, we increase the number of rows by fixing the number of features and compare the performance of four algorithms together.

10 features



In the second experiment, the number of rows was fixed and the number of features have been increased.

1000 rows



-For wide dataset, sorting the features is really time-consuming.
-For long dataset, Algorithm 2 would be slow.