Number Array Machine Learning Project

Module Imports

```
[ ] !pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\muhammad fahad alam\anaconda3\lib\site-packages (0.8.1)
Requirement already satisfied: scipy>=0.19.1 in c:\users\muhammad fahad alam\anaconda3\lib\site-packages (from imbalanced-learn) (1.3.1)
Requirement already satisfied: scikit-learn>=0.24 in c:\users\muhammad fahad alam\anaconda3\lib\site-packages (from imbalanced-learn) (1.0)
Requirement already satisfied: numpy>=1.13.3 in c:\users\muhammad fahad alam\anaconda3\lib\site-packages (from imbalanced-learn) (1.16.5)
Requirement already satisfied: joblib>=0.11 in c:\users\muhammad fahad alam\anaconda3\lib\site-packages (from imbalanced-learn) (0.13.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\muhammad fahad alam\anaconda3\lib\site-packages (from scikit-learn>=0.24->imbalanced-learn) (3.0.0)
```

```
[ ] import numpy as np
    import pandas as pd
    import itertools

    from imblearn.over_sampling import SMOTE
    from sklearn.model_selection import train_test_split
    from sklearn.naive_bayes import GaussianNB
```

Class with all Methods

```
[ ] class NumberArray():
      """
      It contain all the methods used in this project.

      Methods:
      - create_dataset => Method for creating dataset
      - label_dataset => Method for labeling the dataset
      - label_sample => Method returns the sample of the provided sample
      """

      def __init__(self, n, k):
        """
        Constructor of the class. Maps the provided arguments to respective
        attributes.
        """

        self.n = n
        self.k = k

      def create_dataset(self):
        """
        Uses n and k class parameters and create an array with all possible
        combinations with repetition. It converts this array into a pandas dataframe
        for dealing with data in easy and better way.

        Dataframe containing all data is return when this method is called.
        """

        arr = [i for i in range(1, self.n + 1)]
        all_combinations = np.array(list(itertools.product(arr, repeat = self.k)))
        self.dataset = pd.DataFrame(all_combinations, columns=range(1, self.k + 1))
        return self.dataset

      def create_random_sample_dataset(self, no_of_samples = 1000):
        """
        This method creates create random samples of the permutation. It takes no_of_samples
        in argument and create a dataset with that number of random samples. Default no_of_samples
        is set to 1000.
        """

        arr = [i for i in range(1, self.n + 1)]
        combinations = set()
        temp = len(combinations)
        working_fine = no_of_samples

        while len(combinations) != no_of_samples and working_fine:
          combinations.add(tuple(np.random.choice(arr, self.k)))
          if len(combinations) == temp:
            working_fine -= 1
          else:
            temp = len(combinations)
            working_fine = no_of_samples

        if not working_fine:
          print(f"{no_of_samples} samples are not possible. Max permutation for our case are {len(combinations)}")
        combinations_tuple = tuple(combinations)
        self.dataset = pd.DataFrame(combinations_tuple, columns=range(1, self.k + 1))
```

```
    return self.dataset

  def label_dataset(self):
    """
    It creates a labeled dataset. Copies the dataset so that changes doesnot
    effect original dataset. Create a new column for output labels and returns
    the dataset with labels.
    """

    self.labeled_dataset = self.dataset.copy()
    self.labeled_dataset['unique'] = self.labeled_dataset.apply(
        lambda sample: self.unique(sample.values), axis=1
        )
    self.labeled_dataset['difference_max'] = self.labeled_dataset.apply(
        lambda sample: self.difference_max(sample.values), axis=1
        )
    self.labeled_dataset['difference_last'] = self.labeled_dataset.apply(
        lambda sample: self.difference_last(sample.values), axis=1
        )
    self.labeled_dataset['output'] = self.labeled_dataset.apply(
        lambda sample: self.label_sample(sample), axis=1
        )
    return self.labeled_dataset

  def unique(self, sample):
    """
    Method to return unique value for creating a new feature of the dataset
    """

    unique = len(list(np.unique(sample)))
    return unique

  def difference_max(self, sample):
    """
    Method to return difference between maximum and minimum value of the sample for creating a new feature of the dataset
    """

    unique = list(np.unique(sample))
    return max(unique) - min(unique)

  def difference_last(self, sample):
    """
    Method to return difference between maximum and second largest value of the sample for creating a new feature of the dataset
    """

    unique = sorted(list(np.unique(sample)))
    new_unique = sorted(list(np.unique(sample[:-1])))
    return max(unique) - max(new_unique)

  def label_sample(self, sample):
    """
    Return labels of each sample provided.
    """

    sample_value = sample.values
    unique = list(np.unique(sample_value))
    if sorted(unique) == list(range(min(unique), max(unique) + 1)):
      if len(unique) <= 0.6 * self.k:
        return 30
      return 20
    else:
      new_sample_value = list(sorted(sample_value)[:-1])
      new_unique = list(np.unique(new_sample_value))
      if sorted(new_unique) == list(range(min(new_unique), max(new_unique) + 1)):
        if max(unique) - max(new_unique) > 0.7 * self.n:
          if len(unique) <= 0.6 * self.k:
            return 10
      return 20

  def train_model(self, test_size = 0.3 ):
    """
    Method for training a Naive Bayes Model and predict the output using provided features.

    It returns a dataframe with all the features, i.e, engineered features, input features, output label and predicted label.
    """

    labeled_dataset_copied = self.labeled_dataset.copy()
    X = labeled_dataset_copied.drop(["output"], axis=1).values
    y = self.labeled_dataset["output"].values

    oversample =SMOTE(k_neighbors=2)
    X, y = oversample.fit_resample(X, y)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=0)
    self.guassian_nb = GaussianNB()
    self.guassian_nb.fit(X_train, y_train)
    y_pred = self.guassian_nb.predict(X_test)
    print("Number of mislabeled points out of a total %d points : %d" % (X_test.shape[0], (y_test != y_pred).sum()))

    columns = [i for i in range(1, self.k + 1)] + ["unique", "difference_max", "difference_last"]

    self.predicted_dataset = pd.DataFrame(X_test, columns=columns)
    self.predicted_dataset["Original Output"] = y_test
    self.predicted_dataset["Predicted Output"] = y_pred
    return self.predicted_dataset

  def predict_sample(self, sample):
    """
    Method for predicting a single sample of the dataset.
    """

    unique = self.unique(np.array(sample))
    difference_max = self.difference_max(np.array(sample))
    difference_last = self.difference_last(np.array(sample))

    sample.append(unique)
    sample.append(difference_max)
    sample.append(difference_last)

    pred_sample = np.array([sample])
    prediction = self.guassian_nb.predict(pred_sample)
    return prediction

  def export_dataset(self):
    """
    Exports dataset as .csv file
    """

    self.labeled_dataset.to_csv("number_array_dataset.csv",index=False)
```

Make object of class

This line of code creates an Object of the class and all methods are present in that class.

```
[ ] number_array = NumberArray(10, 10)
```

Create dataset using create_dataset() method

We discourage using this method as there may be very large amount of permutations and you may get memory error due to memory constraints.

Use create_random_sample_dataset(no_of_sample) instead. It takes no of samples as argument and create dataset.

```
[ ] #number_array.create_dataset()
```

Create Dataset with Random Samples of the permutation

```
[ ] number_array.create_random_sample_dataset(1000000)
```

	1	2	3	4	5	6	7	8	9	10
0	2	1	6	8	3	7	4	3	4	5
1	6	8	2	10	7	10	10	10	9	8
2	10	2	6	8	3	7	6	3	6	9
3	1	1	6	3	6	4	4	8	1	10
4	5	2	6	6	9	2	4	8	10	3
...	...	...	...	...	...	...	...	...	...	...
999995	8	9	1	8	2	7	7	1	9	1
999996	8	10	8	7	3	7	1	10	1	7
999997	3	3	9	4	9	10	4	8	10	6
999998	5	7	5	9	7	8	4	6	9	8
999999	8	5	6	8	1	1	2	2	8	1

1000000 rows × 10 columns

Label dataset and create new features using feature engineering

```
[ ] label_df = number_array.label_dataset()
```

```
[ ] label_df
```

	1	2	3	4	5	6	7	8	9	10	unique	difference_max	difference_last	output
0	2	1	6	8	3	7	4	3	4	5	8	7	0	20
1	6	8	2	10	7	10	10	10	9	8	6	8	0	20
2	10	2	6	8	3	7	6	3	6	9	7	8	0	20
3	1	1	6	3	6	4	4	8	1	10	6	9	0	20
4	5	2	6	6	9	2	4	8	10	3	8	8	0	20

...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
999995	8	9	1	8	2	7	7	1	9	1	5	8	0	20
999996	8	10	8	7	3	7	1	10	1	7	5	9	0	20
999997	3	3	9	4	9	10	4	8	10	6	6	7	0	20
999998	5	7	5	9	7	8	4	6	9	8	6	5	0	20
999999	8	5	6	8	1	1	2	2	8	1	5	7	0	20

1000000 rows × 14 columns

```
[ ] print("Occurence of 10 => ",label_df[label_df['output'] == 10].shape[0])
    print("Occurence of 20 => ",label_df[label_df['output'] == 20].shape[0])
    print("Occurence of 30 => ",label_df[label_df['output'] == 30].shape[0])
```

```
Occurence of 10 =>  0
Occurence of 20 =>  999338
Occurence of 30 =>  662
```

Train Model

```
[ ] df = number_array.train_model(test_size=0.3)
```

```
Number of mislabeled points out of a total 599603 points : 1412
```

```
[ ] df
```

	1	2	3	4	5	6	7	8	9	10	unique	difference_max	difference_last	Original Output	Predicted Output
0	4	5	3	3	1	3	1	3	4	1	5	4	0	30	30
1	4	9	2	5	1	5	9	10	9	2	6	9	0	20	20
2	1	4	3	3	2	5	1	1	2	5	5	4	0	30	30
3	6	3	8	1	6	9	7	4	7	3	7	8	0	20	20
4	1	1	3	1	5	4	3	4	1	3	5	4	0	30	30
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
599598	1	1	3	3	1	3	4	4	2	1	4	3	0	30	30
599599	2	1	1	3	1	1	4	4	2	3	4	3	0	30	30
599600	3	1	2	5	2	2	1	3	2	2	4	4	0	30	30
599601	2	9	3	6	2	9	10	5	1	3	7	9	0	20	20
599602	3	2	2	5	3	2	2	2	1	1	4	4	0	30	30

599603 rows × 15 columns

```
[ ] print("Original Occurences")
    print("Occurence of 10 => ",df[df['Original Output'] == 10].shape[0])
    print("Occurence of 20 => ",df[df['Original Output'] == 20].shape[0])
    print("Occurence of 30 => ",df[df['Original Output'] == 30].shape[0])

    print("\nY Predict Occurences")
    print("Occurence of 10 => ",df[df['Predicted Output'] == 10].shape[0])
    print("Occurence of 20 => ",df[df['Predicted Output'] == 20].shape[0])
    print("Occurence of 30 => ",df[df['Predicted Output'] == 30].shape[0])
```

```
Original Occurences
Occurence of 10 =>  0
Occurence of 20 =>  299962
Occurence of 30 =>  299641

Y Predict Occurences
Occurence of 10 =>  0
Occurence of 20 =>  298550
Occurence of 30 =>  301053
```

Predict Sample using model

This method could be used for predicting the output of a single sample supplied.

```
[ ] number_array.predict_sample([1,4,4,2,6,1,2,1,1,1])
```

```
array([30], dtype=int64)
```

Export Dataset as .csv file

This method exports the dataset as a .csv file

```
[ ] number_array.export_dataset()
```

[ ]

[ ]

[ ]

[ ]