Report: Multiclass Classification of Plastics

~Kevin Amit Shah

Given Data:

- XPlastic:
  This is a 2D numpy array which has the dimensions of 500x36. The columns values represent the absorbance values at 36 different wavelengths for a set of 500 datapoints.
  The minimum absorbance in the array was 0.0, this implies that the transmittance is full 100%
  The maximum value is 5112.85

- YPlastic:
  This is a 1D numpy array of the shape 500x1, consisting of the names of different types of plastics. The are 5 types of plastic in YPlastic are: Plastic 1, Plastic 2, Plastic 4, PVC and White Plastic.
  The allow_pickle was set to True to read the YPlastic array because it is necessary to read the numpy array having object elements.

Process of Using Different Classifiers:

1. **KNN Classifier**:

   K-nearest neighbors (KNN) is a type of supervised learning algorithm which is used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is close to the test data.

   Initial imports:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

   According to our dataset I used the KNN Classifier to classify the different types of plastics. The KNN classifier is dependent on a number of parameters. Only the parameter n_neighbors, which was set to 5, was considered out of all the parameters. The number of datapoints to consider while deciding the category of a new data point is specified by n_neighbors. When n_neighbors is set to 1, the accuracy is maximized, but there is a risk of overfitting. As a result of observing the accuracy at various values, the accuracy was greatest at n_neighbors = 5. The default values are used for all other parameters. Their values can be modified as well.

   The accuracy_score obtained was **0.9333** (93.33%).
   Upon tuning the hyperparameters, ‘p’ and ‘algorithm’, I still get an accuracy of 0.9333 for all the cases.
   Classification models are evaluated using accuracy score, while regression models are evaluated using r2 score. Because the model was created for classification, accuracy score was utilized to determine efficiency.

Later, to visualize the relationship between the factor k (representing the K-nearest neighbors), I have plotted a graph between the accuracy_score and k. The graphs reflect the fact that accuracy is maximum for k =1 and decreasing eventually, but it has overfitting at k = 1. Therefore, to reduce overfitting, k = 5 is used.

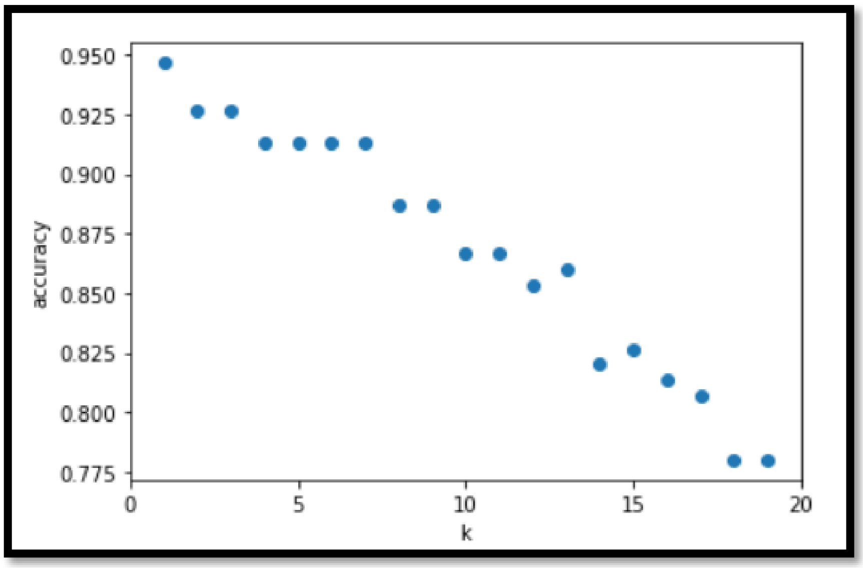

Figure1: Graph between accuracy_score and k

2. **Decision Tree Classifier**:

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too. The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data (training data).

<u>Initial imports:</u>

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
```

Initially, I applied the DecisionTreeClassifier without hyperparameter tuning to check the accuracy with tuning certain parameters:
Estimator: `DecisionTreeClassifier(max_depth = 5, min_samples_leaf = 1, random_state = 0)`

Results:
Accuracy of DT classifier on training set: 1.00
Accuracy of DT classifier on test set: 0.97
Accuracy score: **0.9866666666666667**
accuracy_score was used to find the accuracy and the maximum obtained value was **0.9866** (98.66%)

Later, I used the Grid Search on Decision trees to tune the hyperparameters of the classifier.

Grid Search:

Grid Search uses a different combination of all the specified hyperparameters and their values and calculates the performance for each combination and selects the best value for the hyperparameters. This makes the processing time-consuming and expensive based on the number of hyperparameters involved.

There are many parameters that the classifier depends on out of which to perform the grid search we make a dictionary of several parameters to optimize such as:
criterion, max_depth, min_sample_split and min_sample_leaf.

For the parameters specified, hyperparameter tuning was done to get maximum accuracy. Other parameters were set as follows:
random_state = 0, cv = 10 (cross-validation), verbose=1, n_jobs = -1.

Fitting 10 folds for each of 648 candidates, totalling 6480 fits.
Best parameters obtained:
`{'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}`
Best Estimator: `DecisionTreeClassifier(max_depth=5, random_state=0)`
Best Score: **0.9742857142857144 (**97.428%**)**

3. Logistic Regression

Logistic Regression is used as a classifier. I am using RepeatedStratifiedKFold as the cross-validator, which performs K-fold n times with different randomization in each repetition. I have also used GridSearchCV to tune the hyperparameters of Logistic Regression Classifier.

Imports:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
```

Repeated k-fold cross-validation provides a way to improve the estimated performance of a machine learning model. This involves simply repeating the cross-validation procedure multiple times and reporting the mean result across all folds from all runs. This mean result is expected to be a more accurate estimate of the true unknown underlying mean performance of the model on the dataset, as calculated using the standard error.

```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
```

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. Logistic regression, by default, is limited to two-class classification problems. Some extensions like one-vs-rest can allow logistic regression to be used for multi-class classification problems.

The hyperparameters used in GridSearchCV to tune are ‘solver’, ‘penalty’ and ‘C’.

Results:

Best Score: **0.9926666666666667**
Best Hyperparameters: {'C': 100, 'penalty': 'l2', 'solver': 'newtoncg'}

4. **Kernelized Support Vector Classification (Kernelized SVC)**:

The kernel function is a way for taking data as input and transforming it into the format needed for processing. The term "kernel" refers to a set of mathematical functions used in Support Vector Machine to provide a window through which data can be manipulated.

Imports:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC
```

In this classification, the hyperparameter tuning is done in two different methods.

- Part1: Using GridSearchCV:
  The hyperparameters used to tune is: 'C', 'gamma' and 'kernel'.

```
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}
```

  Best Parameters: {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
  Best Estimator: SVC(C=10, gamma=0.0001)
  Best Score: **0.919999**

- Part2: Using For three loops
  C = 100, decision_shape_function = 'ovr'
  The loops are iterating three different parameters: 'kernel', 'gamma' and 'degree'

  Result:
  Accuracy using Kernalized SVC: **0.9533334**

5. **Naive Bayes Classification**:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Import:

```
from sklearn.naive_bayes import GaussianNB
```

Results:
Accuracy of GaussianNB classifier on training set: 0.99

Accuracy of GaussianNB classifier on test set: 0.97
Accuracy score **0.9733333333333334**

6. **Random Forest Classification**:
   Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

   Import:

```
from sklearn.ensemble import RandomForestClassifier
```

- Part1: Using Random Forests without Hyperparameter tuning
  Results:
  Accuracy of RF classifier on training set: 1.00
  Accuracy of RF classifier on test set: 0.99
  Accuracy score **0.9866666667**

- Part2: Plotted graph between using matplotlib

  I have plotted a graph between the *'Testing Accuracy' and 'Value of n_estimators for Random Forest Classifier'*

  The main aim to plot the graph was to understand at what value of n_estimator do we have a greater accuracy.

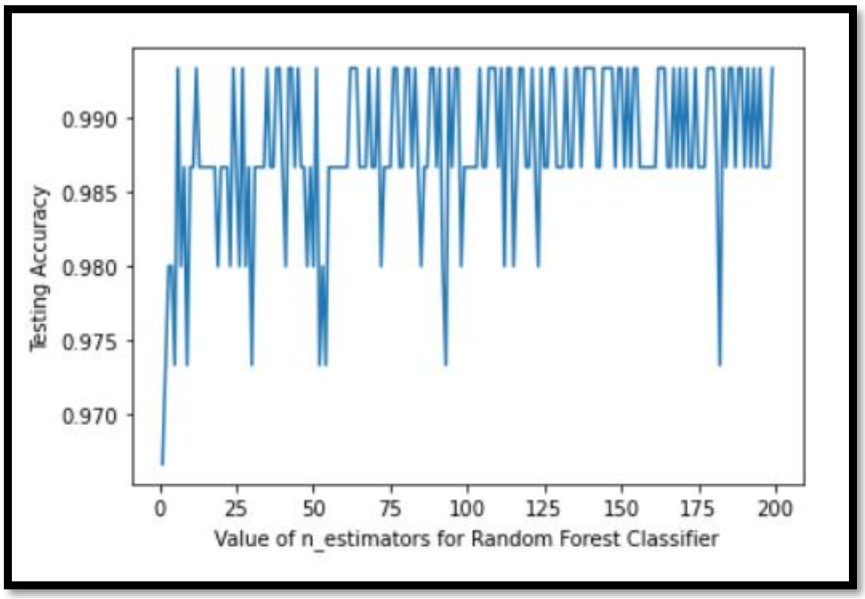

Figure2: Plot between accuracy and n_estimator

- Part3: Using GridSearchCV for hyperparameter tuning
  Parameters to tune:

```
forest_params = [{'max_depth': list(range(10, 15)), 'max_features':
list(range(0,14))}]
```

Results:
Best Parameters: `{'max_depth': 10, 'max_features': 1}`
Accuracy of RF classifier on training set: 1.00
Accuracy of RF classifier on test set: 0.99
Accuracy score **0.993333333**

- Part4: Using RandomizedSearchCV for hyperparameter tuning
  Imports:

  ```
  from sklearn.model_selection import RandomizedSearchCV
  from sklearn.ensemble import RandomForestClassifier
  ```

  RandomizedSearchCV is very useful when we have many parameters to try and the training time is very long. Grid Search is good when we work with a small number of hyperparameters. However, if the number of parameters to consider is particularly high and the magnitudes of influence are imbalanced, the better choice is to use the Random Search.

  This RandomizedSearchCV is executed using two models and their respective dictionary range.

  -
    ```
    param = {'max_depth': [6,9, None],
              'n_estimators':[50, 70, 100, 150],
               'max_features': [1,2,3,4,5,6],
               'criterion' : ['gini', 'entropy'],
               'bootstrap':[True, False]}
    ```
    Estimator:

    ```
    RandomizedSearchCV(RandomForestClassifier(), param, n_iter =10,
    cv=9)
    ```

    Results:

    Best parameters:

    ```
    {'n_estimators': 100, 'max_features': 2, 'max_depth': None,
    'criterion': 'entropy', 'bootstrap': False}
    ```

    Best Score: **0.9970760233918128**

  - random grid = `{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}`

    Fitting 3 folds for each of 100 candidates, totalling 300 fits

    Estimator:

    ```
    RandomizedSearchCV(RandomForestClassifier(), random_grid, n_iter =
    100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
    ```

    Results:

    Best parameters:

```
{'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 4,
'max_features': 'auto', 'max_depth': 10, 'bootstrap': True}
```

Best Score: **0.9971264367816092**

7. **Gradient Boosted Decision Tree (GBDT) Classifier**:
   It generates a prediction model in the form of a collection of weak prediction models, which are essentially decision trees. GBDT is utilized when decision trees are poor. It works in a similar stage-wise approach as the other boosting methods, but GBDT goes a step further by allowing optimization of the differentiable loss function. The random_state was set to 0 to avoid randomness of the estimator.

   Import:

   ```
   from sklearn.ensemble import GradientBoostingClassifier
   ```

   Results:
   Accuracy of GBDT classifier on training set: 1.00
   Accuracy of GBDT classifier on test set: 0.98
   Accuracy score: **0.98**
   The accuracy obtained was **0.98** (98%).

8. **Multi-Layer Perceptron (MLP) Classifier**
   The multilayer perceptron (MLP) is a feedforward artificial neural network model that maps input data sets to a set of appropriate outputs. An MLP consists of multiple layers and each layer is fully connected to the following one. The nodes of the layers are neurons with nonlinear activation functions, except for the nodes of the input layer. Between the input and the output layer there may be one or more nonlinear hidden layers.

   Import:

   ```
   from sklearn.neural_network import MLPClassifier
   ```

   Results:
   Accuracy of MLP classifier on training set: 0.92
   Accuracy of MLP classifier on test set: 0.90
   Accuracy score: **0.9**

Final Results:

- Amongst all the classifiers, the best accuracy score I did achieve is **0.99333** from Random Forest Classifier using GridSearchCV.
- The Best score after hyperparameter tuning achieved is **0.99712643** from Random Forest Classifier using RandomizedSearchCV.