CS5014: Report

Dry Beans Classification

220026989

Section 1

Loading and cleaning the data given is a crucial and important step in machine learning as it is found to impact the accuracy and quality of the models created by training with the said data. In the coursework the data is loaded with the help of pandas library:

1. Import pandas as pd
   ds = pd.read_csv("drybeans.csv")

   ds or dataset contains the loaded data.
2. Next the characteristics or the features of the acquired data is checked with the help of:
   ds.describe()
   Since the dataframe contains of numerical data(except for the Class)

count - The number of not-empty values.
mean - The average (mean) value.
std - The standard deviation.
min - the minimum value.
25% - The 25% percentile*.
50% - The 50% percentile*.
75% - The 75% percentile*.
max - the maximum value.

are provided.

3. Removal of duplicate values.
   Duplicate value removal is one of the important steps in data cleaning as duplicate data can lead to:
   -Overfitting, where the model learns to memorize the training data instead of generalizing or having the ability to predict unseen data accurately.
   -Bias in model.
   -Computational cost.

Data imbalance. With more number of data in a specific class creates an imbalance of data causing the model to become biased towards one class and perform poorly on the other one.

We check the number of duplicate data given by:

ds.duplicated().sum()

which gives the total count of duplicate data present.
Later, we drop these duplicates in the code by:
df = ds.drop_duplicates()

4. Null values:
   Missing values in data set also can have significant affect on the performance of the model:
   - May potentially reduce the accuracy and generalizability of the model.

   -Biased results, could be one of the results of not handling null values. They can lead to the model learning to predict result in favour of other classes.

   -Inaccurate predictions

   -data leakage, where the model learns to use the null value as a data to make further predictions which could lead to overfitting and poor classification of the new data.

   In the python file, we check for null values using:

   df.isnull().count()

   Which gives the count of all the rows of each feature supplied in the csv file for which the data is not null. Since there are no null values in the data set we continue.

5. Correlated features, are two or more features in a dataset that are highly correlated with each other. That is when one feature changes the other feature changes in a similar way(either positively or negatively).Not removing correlated features can lead to:

   -Overfitting , where the model relies heavily on a particular set of features leading to inaccurate predictions.
   -Reduced interpretability, making it difficult to understand which feature is contributing more to model's output.
   -Causes model instability, where small changes in the input data may lead to large fluctuations in the model's output.

   In the coursework first we find a table of how much each features are correlated with each other using:

df.corr()

from which we manually store the features that are highly correlated with each other. It should be noted that sometimes there are some features that are highly correlated with multiple features. In these cases the feature that correlates highly to most number of features is kept while the others are dropped. I've dropped only those features that have a correlation of more than 94%.

The correlated features are then stores into an array of correlated features to be later dropped.

Splitting data into train and test is one of the important steps for an accurate model. The train set is used to fit the model, which is used for setting the parameters of the model using the input features to the corresponding output set. The test set is then used to evaluate the performance of the trained model, which for the model is new unseen data. By this process we can understand how well the model will perform on a new data. If this process is not done there is either no working proof that the data is working as expected with unseen data, or if same data is used to test the model overfitting may occur -(When model learns to fit the noise in the data rather than the pattern, thus resulting in poor performance on the new data).The split occurs randomly and equally from all the classes(or outcome) is not biased.

In the python code, data split is achieved by:

```
x = df.drop(['Class'],axis = 1)
y = df['Class']

x_train_data,x_test_data, y_train_data,y_test_data = train_test_split (x, y , train_size=.80, random_state=50, stratify=df.Class)
```

where only 80% of the given data is taken for training, while 20% is kept away for testing. Here random_State is a parameter that specifies the seed value – which when kept the same, ensures a deterministic and repeatable split. stratify helps maintain equal proportions of 'Class' both in test and train.

Data preprocessing is another crucial step in training a model in machine learning.
Encoding refers to converting categorical data into numerical data.
These are important because many machine learning algorithms cannot directly handle categorical data. To use categorical data as input to a machine learning algorithm, it must be first encoded as numerical data.
In the program file, this is attained with the help of LableEncoder().
In the program we use:

```
encoding_target = LabelEncoder()
```

y_train_data = encoding_target.fit_transform(y_train_data)
y_test_data = encoding_target.transform(y_test_data)

The fit_Transform() method fits the encoder on the training data,i.e; it computes the mapping between the unique labels in y_train_Data and integer values, which is then applied to transform the labels in y_train_data to the corresponding integer values. After fitting the encoder on the training_Data, the same encoder is used to transform the labels in y_test_Data to their corresponding integer values just by using transform().

Scaling refers to transforming the range of numeric data given into a common scale. In the Code we have used standardScaler() function that standardizes the features by subtracting the mean of the feature and dividing by standard deviation. This transformation results in each feature having a mean of 0 and a standard deviation of 1. Scaling is another important step in machine learning models as:

-Improves model performance, as many models are sensitive to the scale of the input features. Scaling the features can improve the performance of these models by making the optimization process much more efficient and reducing the chances of the model getting stuck in the local optima.

-Equalizes feature importance; ensuring that each feature contributes equally to the model fitting process. Without scaling, features with larger values may influence the learning process leading to biased results.

-Normalizing the data can improve the performance.

-Removes the effect of outliers if any;

Data leakage, is averted by using the same mapping(fit) as made in the train data – incase of both scaling and encoding to be applied to the testdata. In the fit_transform() method (both incase of StandardScaler and LabellEncoder), once the fit_Transform() maps the required input features to a specific value, the same mapping is made to be used in the testing data. This is because if we use fit_Transform() on both the data it can introduce data leakage.

Section 2.

a) The given dataset has 7 classes which are not equally represented in the dataset. Since the classes are imbalanced the accuracy affects the model since class weight is given as none. In this case, each class is treated equally during training, regardless of its frequency or importance. Using penalty = 'None' means there is no regulation applied during training, which may result in overfitting if the model is complex. Some form of regularization to prevent overfitting generally improves the performance of the mode. So, the accuracy of this classification model may be

equal to or less than that of when using penalty = l2 and class_Weight = balanced as introducing these is said to bring better performance for the model.

b)

1. penalty – This parameter specifies the type of regularization to be applied during the training of the model. It is used to prevent overfitting of the model to the training data by adding a penalty term whenever the model tries to overfit while studying the data. The two types of regularization that is supported by Logistic Regression are – 'L1' and 'L2'.While L1 adds the absolute value of the weights to the loss function, L2 adds the squared value of the weights.

2.tol – parameter used to specify the tolerance for stopping the iterative optimization process during training. To minimise the loss function, the algorithm iteratively adjusts the model's weights until convergence or until the change in the loss function is smaller than the tol value set. Although the process wil take a lot longer with a smaller tol value, the model outcome/ prediction might be more accurate.

3. max-iter : is the parameter that specifies maximum number of allowed iterations for the optimization process during the training stage. If the algorithm does not converge to the required range within the specific number of iterations, this will help stopping the iterations and return the current solution. Larger the value of max-iter more accurate might be the solution, sacrificing the training time and may also overfit the model. So, it is always good to have an optimal value of max-iterations.

c)

The class_weight = 'balanced' assings weight to each class inversely proportional to the class frequencies. That is, the model will give more weight to the class with lower concentration of datapoints while less weight to the one with higher concentration of data points. The purpose of using this is to take care of the class imbalance of the input data, which otherwise leads to biased predictions,where the model accuracy for the majority class is a lot higher(depending on the concentration) compared to the one with lower concentration. By assigning relatively higher weight with respect to their weightage the model has reduced to no bias and improves overall accuracy.

d)

1.predict() – Method takes in a set of input samples and returns a binary vector indicating the predicted class label for each sample.The vector contains either 0 or 1, representing the negative and positive classes. The predicted class labels are based on decision threshold of 0.5, which meands that the positive class is predicted if the output of the logistic regression for a given sample is larger that 0.5 and negative class comes to the picture when the it is lower.

2.decision_function() – Takes a set of input samples and returns a vector of signed distances to the hyperplane used to separate the classes, which is determined by the weights learned during training. Positive values indicate that the sample is likely to be in the positive class. The magnitude of the value reflects the prediction strength.

3.predict_probab() – Takes a set of input samples and returns a matrix of probabilities of samples belonging to each class. The n_sample x n_classes matrix contains the probabilities that are estimated using the logistic function.

Section 3 :-

1.Classification accuracy of the model:

```
0.9258028792912514
```

Formula for calculating :

Accuracy = (True Positives + True Negatives)/Total number of instances

Total instances = TP + TN + FP + FN

TP – number of true positives (correctly predicted positive instances)

TN – number of true negatives (correctly predicted negative instances)

FP – number of false positives (incorrectly predicted positive instances)

FN – number of false negatives (incorrectly predicted negative instances)

2.Balanced accuracy of the model:

```
0.9348159661361118
```

Formula for calculating :

Balanced Accuracy = (True Positives + True Negatives)/2

3. Confusion matrix

a) Unbalanced:

```
array([[235,   0,  22,   0,   0,   2,   6],
[  0, 104,   0,   0,   0,   0,   0],
[  6,   0, 312,   0,   4,   1,   3],
[  1,   0,   0, 653,   1,   7,  47],
[  1,   0,   8,   2, 354,   0,   7],
[  5,   0,   0,   8,   0, 379,  14],
[  1,   0,   1,  44,   5,   5, 471]], dtype=int64)
```

b) Balanced:

```
array([[240,   0,  20,   0,   0,   2,   3],
[  0, 104,   0,   0,   0,   0,   0],
[  9,   0, 310,   0,   4,   1,   2],
[  1,   0,   0, 635,   1,   7,  65],
[  1,   0,   9,   2, 353,   0,   7],
[  5,   0,   0,   8,   0, 379,  14],
[  2,   0,   2,  37,   7,   5, 474]], dtype=int64)
```

Comparing the confusion matrices,

- In the balanced matrix, there are more number of correct observations(True positives) correctly classified for all the classes compared to the unbalanced matrix(with just a small exception in case of class 3). The small exception in case of class 3 could be because of the removal of correlated features. Setting up a higher percentage of correlation, narrowing the threshold could result in a better TP. Apart from that this is to be expected for the balanced model.

  Overall the balanced confusion matrix has higher numbers of true positives and true negatives and lower number of false positives and false negatives. This is proof that the balanced model is better at correctly classifying.

4.

	precision	recall	f1-score	support
0	0.93	0.91	0.92	265
1	1.00	1.00	1.00	104
2	0.91	0.95	0.93	326
3	0.93	0.90	0.91	709
4	0.97	0.95	0.96	372
5	0.96	0.93	0.95	406
6	0.84	0.90	0.87	527
accuracy			0.92	2709
macro avg	0.93	0.93	0.93	2709
weighted avg	0.92	0.92	0.92	2709

-Precision – Ratio of TP to the total number of positive predictions.

-Recall – Ratio of TP to the total number of actual positives.

Micro and macro averages are used to compute the overall performances of a classification model based on performance metrics such as precision, recall etc.

Micro averaging -:

- We calculate the contributions of all classes to compute the metric.
- Gives equal importance to all the sample and is quite used when dealing with imbalanced datasets.

Macro-averaging -:

- We calculate the metric foreach class separately and then take the average over all the classes.
- Often used when all the classes are equally important and balanced.

Section 4 :- Advanced

a) **Penalty** = 'l2'

Accuracy score :-

```
0.9191583610188261
```

Confusion matrix:-

```
array([[240,   0,  19,   0,   0,   2,   4],
[  0, 104,   0,   0,   0,   0,   0],
[  6,   0, 313,   0,   3,   1,   3],
 [  1,   0,   0, 630,   1,  12,  65],
[  1,   0,   9,   2, 355,   0,   5],
[  5,   0,   0,   8,   0, 379,  14],
[  2,   0,   2,  36,  11,   7, 469]], dtype=int64)
```

Evaluation :-
Usually, it is to expected that the balanced, with penalty model that avoids overfitting, gets a more accurate result with more TP. But here we have a confusion matrix that is similar to the balanced non-penalty confusion matrix with slightly fewer TP.