Diploma Thesis

Distinguishing seven different registered varieties of dry beans with similar features in order for seed classification

Abhishek Swain
Diploma, University of Hyderabad

Feb 27, 2022

Contents

Problem Definition

What is the problem about?

Dry bean is the most popular pulse produced in the world. The main problem dry bean producers and marketers face is in ascertaining good seed quality. Lower quality of seeds leads to lower quality of produce. Seed quality is the key to bean cultivation in terms of yield and disease. Manual classification and sorting of bean seeds is a difficult process. Our objective is to use Machine learning techniques to do the automatic classification of seeds.

Why is this problem important to solve?

Ascertaining seed quality is important for producers and marketers. Doing this manually would require a lot of effort and is a difficult process. This is why we try to use machine learning techniques to do the automatic classification of seeds.

Business/Real-World impact of solving this problem?

- Saves hours of manual sorting and classification of seeds.
- We can do it in real-time.

Dataset

Source of dataset

The dataset is downloaded from the UCI Machine learning repository. Link: https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset

Explanation of each feature

There are a total of 16 features - 12 dimensional and 4 shape features

1. Area (A): The area of a bean zone and the number of pixels within its boundaries.

$$A = \sum_{r,c \in R} 1$$

2. Perimeter (P): Bean circumference is defined as the length of its border.
3. Major axis length (L): The distance between the ends of the longest line that can be drawn from a bean.

4. Minor axis length (l): The longest line that can be drawn from the bean while standing perpendicular to the main axis.
5. Aspect ratio (K): Defines the relationship between L and l.
6. Eccentricity (Ec): Eccentricity of the ellipse having the same moments as the region.
7. Convex area (C): Number of pixels in the smallest convex polygon that can contain the area of a bean seed.
8. Equivalent diameter (Ed): The diameter of a circle having the same area as a bean seed area.
9. Extent (Ex): The ratio of the pixels in the bounding box to the bean area.
10. Solidity (S): Also known as convexity. The ratio of the pixels in the convex shell to those found in beans
11. Roundness (R): Roundness is the measure of how closely the shape of an object approaches that of a mathematically perfect circle. Calculated with the following formula:
12. Compactness (CO): Measures the roundness of an object.

The shape features are:

1. *ShapeFactor1(SF1)* = $\frac{L}{A}$
2. *ShapeFactor2(SF2)* = $\frac{l}{A}$
3. *ShapeFactor3(SF3)* = $\frac{A}{\frac{L}{2}*\frac{l}{2}*\pi}$
4. *ShapeFactor4(SF4)* = $\frac{A}{\frac{L}{2}*\frac{l}{2}*\pi}$

Dataset size and challenges

1. The dataset is in the form of an excel sheet with 13,611 rows, each referring to one example of a seed with 16 features.
2. The last column of the sheet is "Class" which has 7 unique classes: ***Barbunya, Bombay, Cali, Dermason, Horoz, Seker and Sira***.
3. The dataset is imbalanced, "BOMBAY" class has only 522 examples whereas "DERMASON" has 3546 examples. So, we need to deal with this imbalance.

Tools to process the data

I will use 'Pandas' to process the data. There's also 'Dask', but Pandas should be good enough to handle 13k rows in the excel sheet

Data Acquisition

1. Data is openly available and collected from the UCI Machine learning repository
2. Other than primary source no more data can be acquired as it has been acquired through a specialized camera system only available with the authors of the paper.

Key Performance Indicator (KPI)

Business Metric Definition

1. We will be using *Confusion Matrix* and using it for calculating per-class *F1-score.*
2. We will average the calculated *F1-score* for all the 7 classes.

Why is the metric used?

Since the dataset is imbalanced, if we simply use a metric like accuracy, it will give us a very skewed idea of the performance of the model. This is why we need metrics that are robust to imbalanced data.

Alternative metrics that can be used?

Accuracy is another metric we can use but that is not going to help us much. We have fewer examples of BOMBAY class which constitutes about 3.8% of the dataset. So, let's say our model doesn't predict the BOMBAY class at all, that would still give us an accuracy close to 96%

Pros and cons of metric used?

F1 - score

Pros :

1. Takes data distribution into account, so it's useful in-case

Cons :

1. Less interpretable. Precision and recall are more interpretable than the f1-score since it measures the type-1 error and type-2 error. However, f1-score measures the trade-off between this two.
2. When positive class is minority class, the score is quite sensitive when there is switching where the ground truth is positive.

We will also using other metrics: Recall, Precision, Sensitivity, Balanced Accuracy, AUC in order to mitigate the cons of F1-score.

Exploratory Data Analysis

Description of data at a glance

We will use this dataframe for further analysisA

brief overview of the dataframe

RangeIndex: 13611 entries, 0 to 13610Data
columns (total 17 columns):

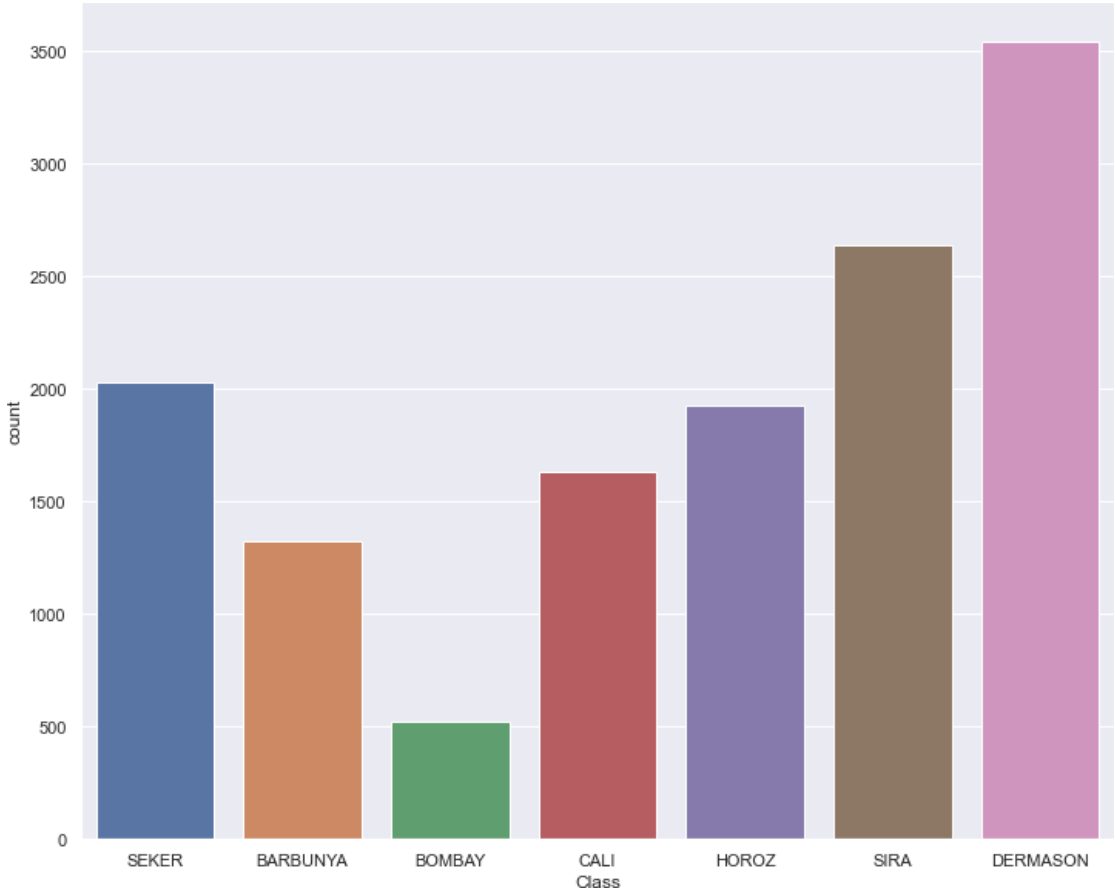
#	Column	Non-Null Count	Dtype
0	Area	13611 non-null	int64
1	Perimeter	13611 non-null	float64
2	MajorAxisLength	13611 non-null	float64
3	MinorAxisLength	13611 non-null	float64
4	AspectRation	13611 non-null	float64
5	Eccentricity	13611 non-null	float64
6	ConvexArea	13611 non-null	int64
7	EquivDiameter	13611 non-null	float64
8	Extent	13611 non-null	float64
9	Solidity	13611 non-null	float64
10	roundness	13611 non-null	float64
11	Compactness	13611 non-null	float64
12	ShapeFactor1	13611 non-null	float64
13	ShapeFactor2	13611 non-null	float64
14	ShapeFactor3	13611 non-null	float64
15	ShapeFactor4	13611 non-null	float64
16	Class	13611 non-null	objectdtypes:

float64(14), int64(2), object(1) memory usage: 1.8+ MB

- We have 16 features, 12 dimensional and 4 shape features
- The Classcolumn contains the Classes
- We have 13,611 rows each corresponding to 16 features per bean
- We have got 5 different classes: 'SEKER', 'BARBUNYA', 'BOMBAY', 'CALI', 'HOROZ','SIRA', 'DERMASON'

Analysisng the Classes

```
_ = sns.countplot(data=df, x='Class')
```

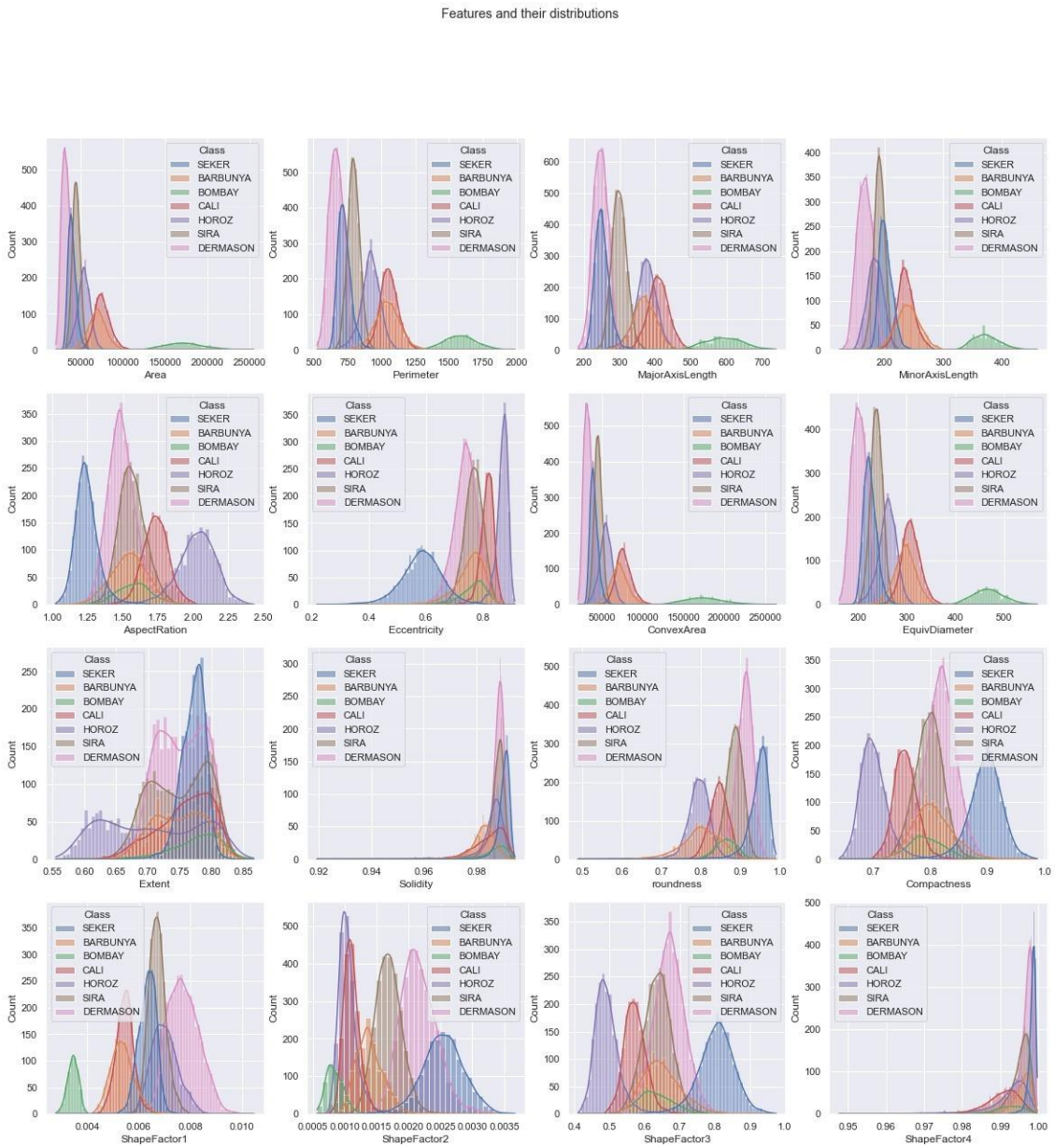


Obvservation

- We have got 5 classes and above are the counts of the classes. As, we can see that the majority class is DERMASON and minority one is BOMBAY. The data is imbalanced as BOMBAYhas only 500 examples where as DERMASONhas 3500 examples.

Analysing the features

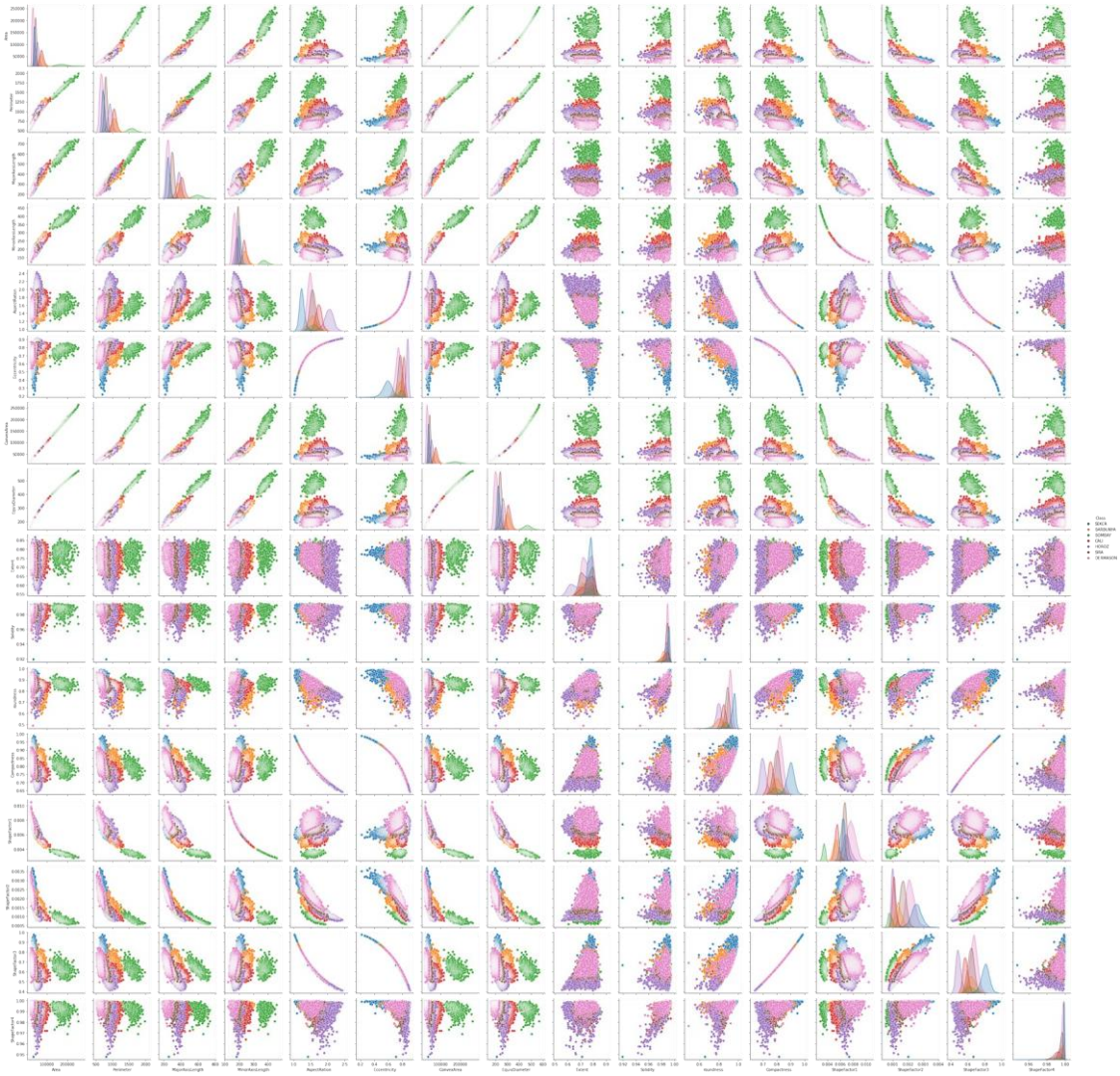
Univariate Analysis



Obvservation

- BOMBAYclass can be differentiated easily using any feature
- The other classes have a lot of overlap and are not easy to distinguish

Bivariate Analysis

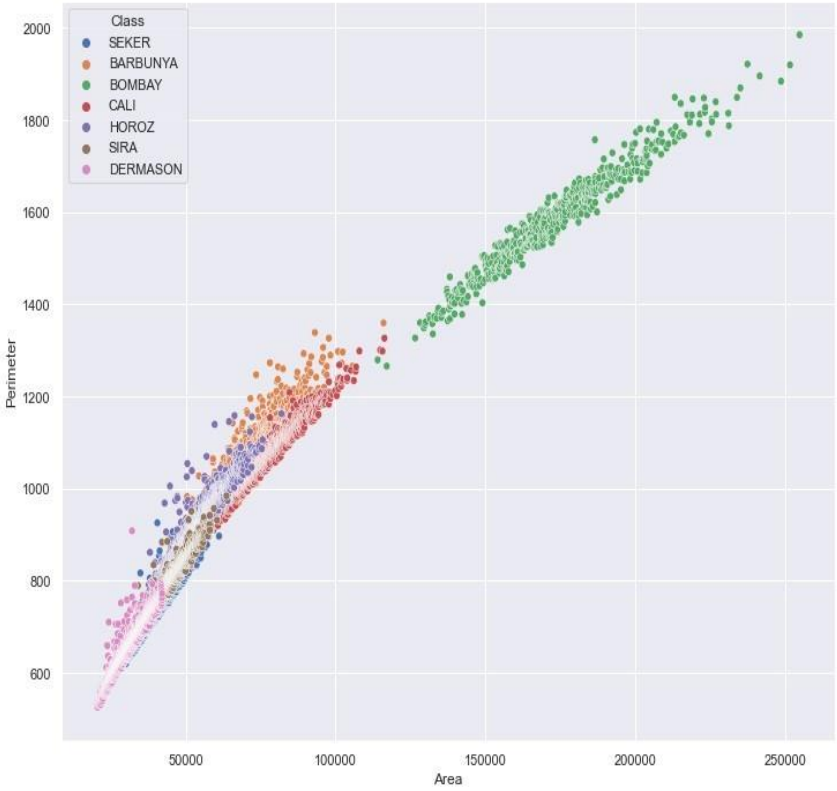


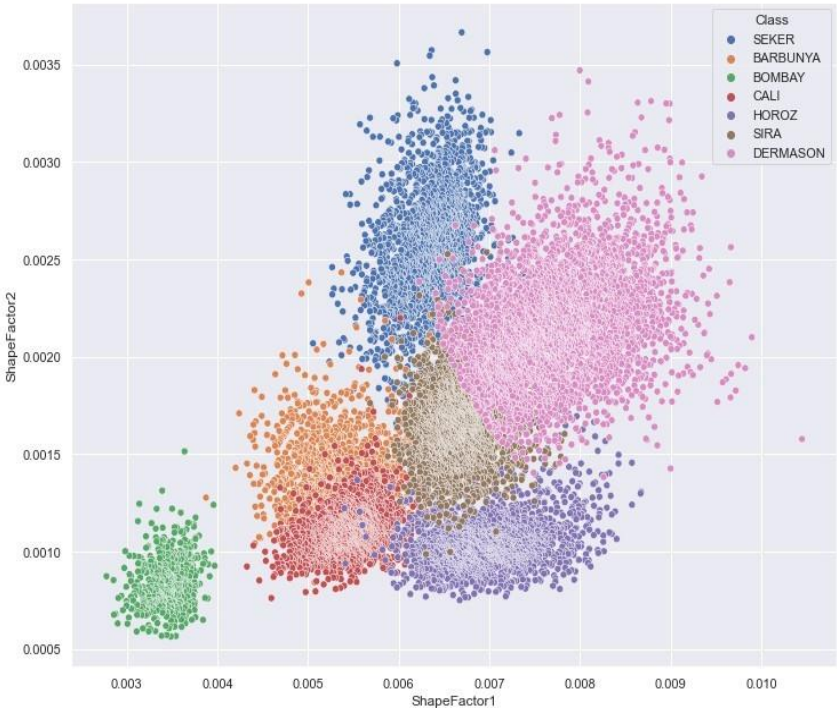
Since there are a lot of features, let's look at the pair plot first and then we can progress

obvservation

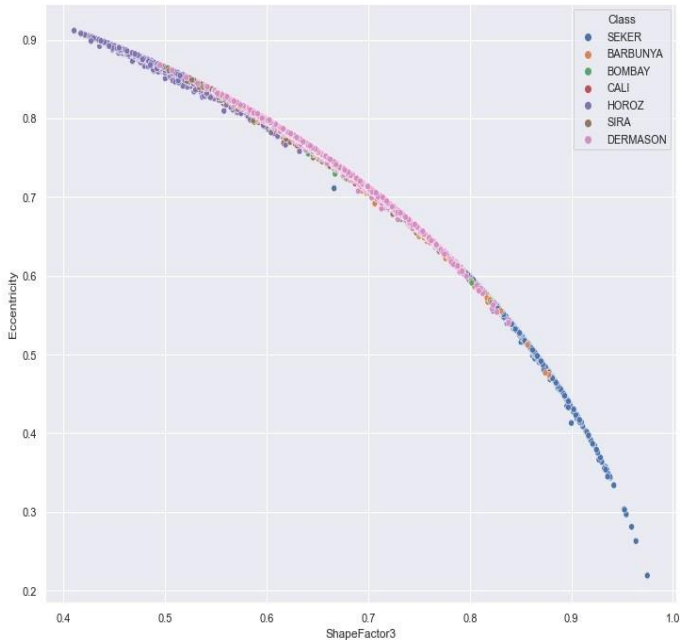
- So, it's kinda cumbersome, but still it gives us some details about it our data
- The green colored points are points belonging to BOMBAYour minority class. It seems any feature is good enough for separating BOMBAY from other classe.
- We can't really say the same for other classes

We can zoom in on some of the plots and see up close for ourselves that BOMBAY is easy to separate


Class
SEKER
BARBUNYA
BOMBAY
CALI
HOROZ
SIRA
DERMASON
Perimeter
Area
2000
1800
1600
1400
1200
1000
800
600
50000
100000
150000
200000
250000


Class
SEKER
BARBUNYA
BOMBAY
CALI
HOROZ
SIRA
DERMASON
ShapeFactor2
ShapeFactor1
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.010

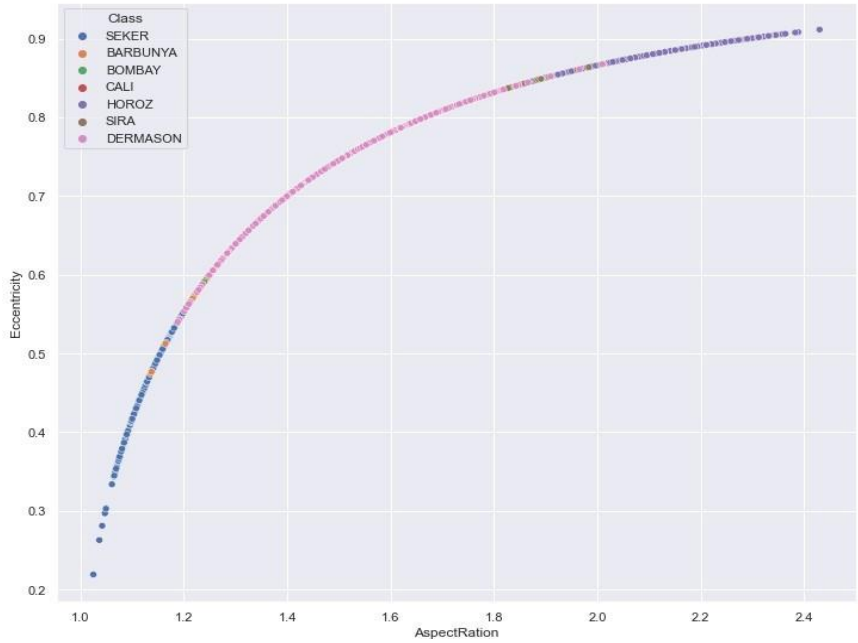
Now, lets zoom in on some features which are highly correlated to each other

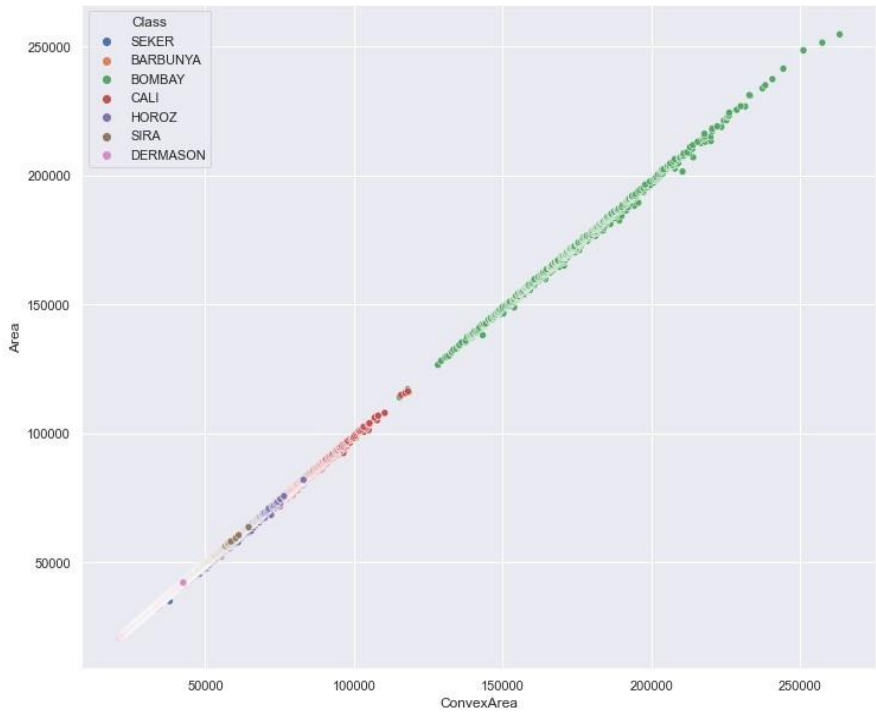


Obvservation

- ShapeFactor3 and Eccentricity are high negative correlation, they seem to be perfectly lining up

Few other features with high negative and positive correlation



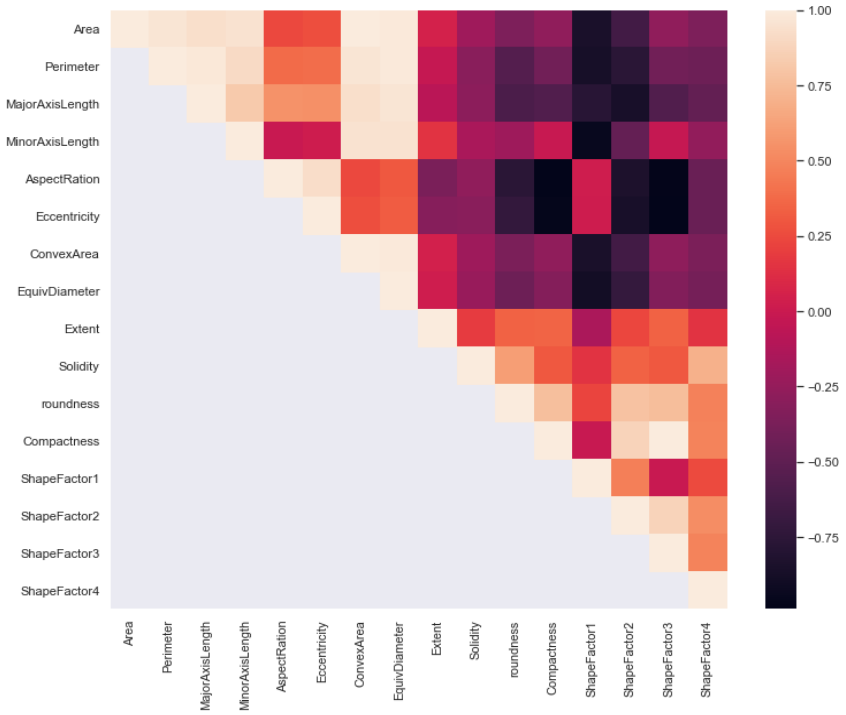


Obvservation

- This is pretty interesting to look at, Area and ConvexArea seem to be the exact same features. Makes sense as ConvexArea approximates Area to the closest convex polygon

Let's Move on to correlation analysis

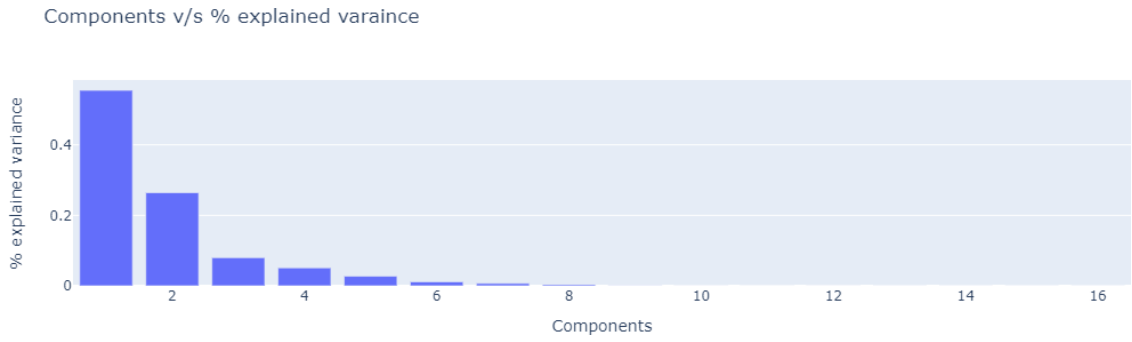
Correlation Analysis



Obvservation

- As, we can see that most of our features are highly correlated either negatively or positively.
- My hypothesis is even if we use very less features, we will still be able to descibe ourdata well.

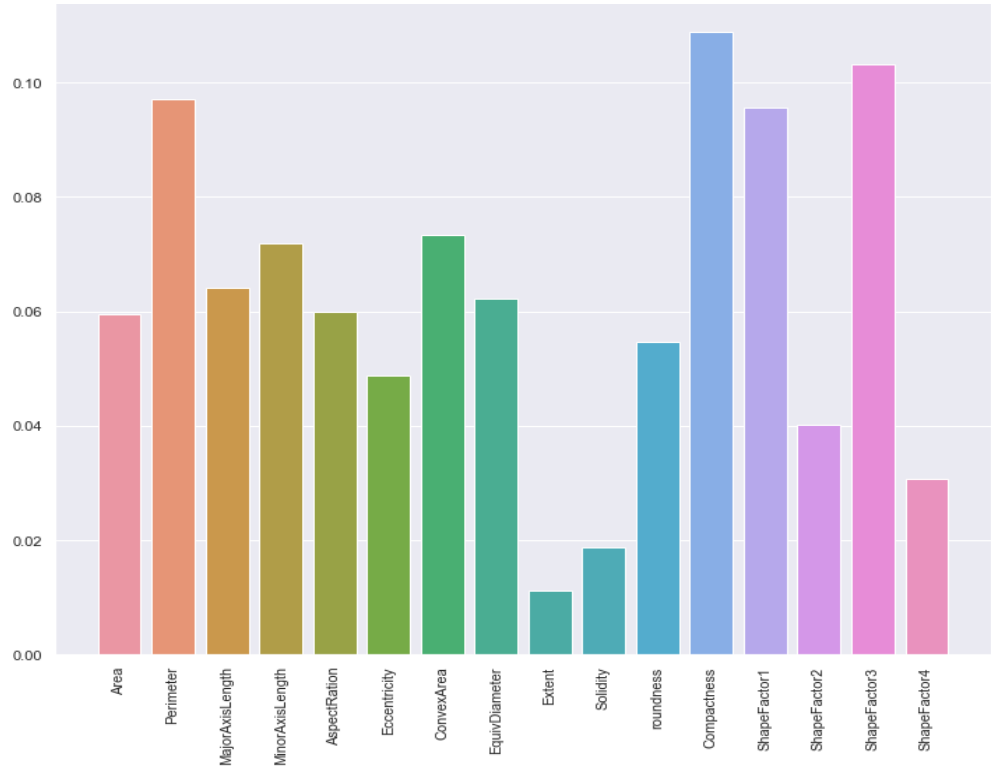
Let's use PCA and see if that holds true. The idea of PCA is simple — reduce the number of variables of a data set, while preserving as much information as possible.



Obvservation

- It's interesting to see that just the first 5 components are good enough to explain 96 % of the data. If we take just 8 out of the 16 components, we can explain the wholedata
- But PCA is like not so good for interpretebility. Nonetheless, it kind of validates my hypothesis

Feature importances



Obvservation

- ShapeFactor3, Compactness, Perimeterhave the highest importances.
- We will use one of them to remove outliers.

Cleaning the data

Dealing with missing values

column	NA count	Null count
Area	0	0
Perimeter	0	0
MajorAxisLength	0	0
MinorAxisLength	0	0
AspectRation	0	0
Eccentricity	0	0
ConvexArea	0	0
EquivDiameter	0	0
Extent	0	0
Solidity	0	0
roundness	0	0
Compactness	0	0
ShapeFactor1	0	0
ShapeFactor2	0	0
ShapeFactor3	0	0

ShapeFactor4	0	0	
Class	0	0	

obvservation

- As you can see, the dataset is fairly complete with no missing or na values. So, we don't need to deal with them

Checking for negative values

Since all the featues are either dimensional or derived from the dimensional features, thhevalues can't be negative. Let's cehck for negative features.

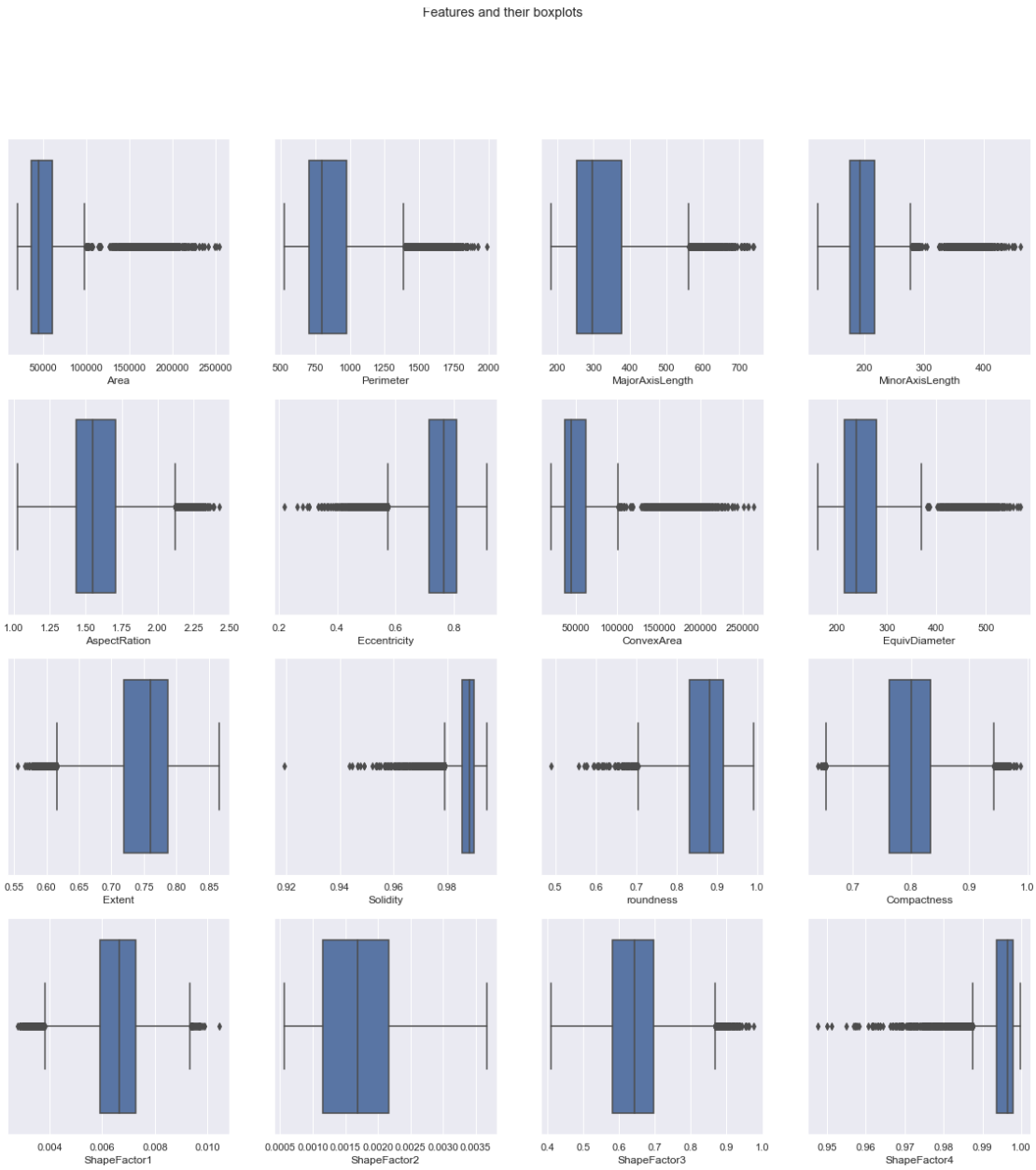
```
Area                0
Perimeter           0
MajorAxisLength     0
MinorAxisLength     0
AspectRation        0
Eccentricity        0
ConvexArea          0
EquivDiameter       0
Extent              0
Solidity            0
roundness           0
Compactness         0
ShapeFactor1        0
ShapeFactor2        0
ShapeFactor3        0
ShapeFactor4        0
dtype: int64
```

Obvservation

- All the columns have positive values which is good as we can now use all rows

Outlier Removal

Let's see the distirbution of the features



Outlier removal using zscore method

column	method	% data retained
roundness	zscore	100.0
Solidity	zscore	100.0
ShapeFactor4	zscore	100.0
Extent	zscore	100.0
Eccentricity	zscore	100.0
ShapeFactor1	zscore	99.993
Compactness	zscore	99.993
ShapeFactor2	zscore	99.963
ShapeFactor3	zscore	99.941
AspectRation	zscore	99.89
MajorAxisLength	zscore	97.678
Perimeter	zscore	97.032
EquivDiameter	zscore	96.584
ConvexArea	zscore	96.451
Area	zscore	96.451
MinorAxisLength	zscore	96.268

Removal of outliers using iqr method

column	method	% data retained
ShapeFactor2	iqr	100.0
roundness	iqr	99.331
Compactness	iqr	99.199
ShapeFactor3	iqr	98.567
Extent	iqr	97.98
MajorAxisLength	iqr	97.215
AspectRation	iqr	96.525
Perimeter	iqr	96.327
EquivDiameter	iqr	96.135
ShapeFactor1	iqr	96.084
ConvexArea	iqr	95.959
Area	iqr	95.952
MinorAxisLength	iqr	95.82
ShapeFactor4	iqr	94.365
Solidity	iqr	94.284
Eccentricity	iqr	93.806

Conclusion

We get a lot of insights from the data:

- The dataset is clean
- Features are highly correlated
- Removal of few features will not impact the performance or interpretability

Feature Selection and Modelling

Feature Selection

Feature selection is the process of reducing the number of input variables when developing apredictive model.

It is desirable to reduce the number of input variables to both reduce the computational cost ofmodelling and, in some cases, to improve the performance of the model

Statistical-based feature selection methods involve evaluating the relationship between each input variable and the target variable using statistics and selecting those input variables that have the strongest relationship with the target variable. These methods can be fast and effective, although thechoice of statistical measures depends on the data type of both the input and output variables.

Variance Thresholding

If the variance is low or close to zero, then a feature is approximately constant and will not improvethe performance of the model. In that case, it should be removed.

Variance will also be very low for a feature if only a handful of observations of that feature differfrom a constant value.

What we can do is set a threshold and drop features with low variance

Anova Test

Analysis of variance (ANOVA) is a statistical technique that is used to check if the means of two or more groups are significantly different from each other. ANOVA checks the impact of one or more factors by comparing the means of different samples.

If we had categorical variables, we would do another test called the $\chi 2$ test. Since we have all numericfeatures, we do the ANOVA test.

Recursive Feature Elimination

Recursive Feature Elimination selects features by recursively considering smaller subsets of featuresby pruning the least important feature at each step. Here models are created iteratively and, in each iteration, it determines the best and worst performing features and this process continues until all thefeatures are explored. Next ranking is given on each feature based on their elimination order. In the worst case, if a dataset contains N number of features RFE will do a greedy search

for $N2$N2 combinations of features.

Feature selection using Random Forest

Feature selection using Random Forest comes under the category of Embedded methods. Embeddedmethods combine the qualities of filter and wrapper methods. They are implemented by algorithms that have their own built-in feature selection methods. Some of the benefits of embedded methods are:

1. They are highly accurate.
2. They generalize better.
3. They are interpretable

Here is a summary of all the feature selection methods and the features selected

Methods	Features Selected
Variance Thresholding (threshold = 1)	['Area', 'Perimeter', 'MajorAxisLength', 'MinorAxisLength', 'AspectRation', 'Eccentricity', 'ConvexArea', 'EquivDiameter', 'Extent', 'roundness', 'Compactness', 'ShapeFactor1', 'ShapeFactor2', 'ShapeFactor3']
ANOVA F-test	['Area', 'Perimeter', 'MajorAxisLength', 'MinorAxisLength', 'ConvexArea', 'EquivDiameter', 'ShapeFactor1', 'ShapeFactor2']
Recursive Feature Elimination (estimator = DecisionTreeClassifier)	['Perimeter', 'MajorAxisLength', 'MinorAxisLength', 'roundness', 'Compactness', 'ShapeFactor1', 'ShapeFactor3', 'ShapeFactor4']
Using RandomForest feature importance	['Perimeter', 'MajorAxisLength', 'MinorAxisLength', 'AspectRation', 'EquivDiameter', 'Compactness', 'ShapeFactor1', 'ShapeFactor3']

Modelling

```
from pycaret.classification import *from
sklearn import metrics
from sklearn.model_selection import train_test_split

import seaborn as sns
from joblib import dump, load
import json
import os

sns.set(rc={"figure.figsize": (10, 8)}, font_scale=1.25)

df = pd.read_csv("./DryBeanDataset/Dry_Bean_Dataset.csv").sample(frac=1).r
eset_index(drop=True)

df.head()
```

Experiment without transformed data

Docs: PyCaret

We setup a pycaret experiment. The parameters are:

- data: df

- target: Class
- normalize: Normalizes all the numeric features using method mentioned using normalize_methodif set to True
- transformation: Applies yeo-johnsontransformation or method mentioned using transform_methodif set to True
- fix_imbalance: Fixes imbalance using SMOTEor method mentioned using imbalance_methodif set to True

```
exp = setup(
        data=df,
        target='Class',
        train_size=0.7,
        experiment_name='baseline_without_transforms',remove_perfect_collinearity=False
)
```

```
<pandas.io.formats.style.Styler at 0x1ebd4f4ff10>
```

Comparing base-line models

Calling the compare_models()is going to fit all classification models for our data

```
%%time
best_model = compare_models()
best_model
```

```
<pandas.io.formats.style.Styler at 0x1ebd4a1bd00>Wall

time: 2min 5s

LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=1
.0,
               importance_type='split', learning_rate=0.1, max_depth=-1,
               min_child_samples=20, min_child_weight=0.001, min_split_gai
n=0.0,
               n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
               random_state=8669, reg_alpha=0.0, reg_lambda=0.0, silent='w
arn',
               subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

The F1 here is the weighted f1 we are using as a metric. So, that's good, we can also pass a custom metric

Obvservation

- Light Gradient Boosting Classifier performs the best among all the baselines, without us doig any transforming or feature selection at a F1 of approx ~ 93

This is pretty good, let's see if we can stretch it further using tuning the model.

Individual Estimators and tuning them

We can see there's not much difference between gradient boosting and the LGBM Classifier. WE will start with LGBM as it's faster to train

```
def plot(estimator, plot_type, dst):res =
    plot_model(
        estimator=estimator,
        plot=plot_type, save=True
    )

    os.rename(res, dst)
    for file in os.listdir("."):
        if file.endswith('.png'):
            os.remove(file)

def clean_params(params):d =
    {}
    for key, value in params.items(): d[key.replace('actual_estimator__',
        "")] = value

    return d

def save(model, tuner=None):
    if tuner is not None:
        dump(
            model,
            filename=f"./ML_models/PC_{model.__class__.__name__}_{tuner.__class__.__
name__}.model"
        )
        with open(
            f'./ML_results/PC_{model.__class__.__name__}_{tuner.__class__.
__name__}_params.json',
            mode='w'
        ) as f:
            json.dump(clean_params(tuner.best_params_), fp=f)

        print(f"Model saved at: ./ML_models/PC_{model.__class__.__name__}_
{tuner.__class__.__name__}.model")
        print(f"Tuner saved at: ./ML_results/PC_{model.__class__.__name__}
_{tuner.__class__.__name__}_params.json")

    else:
        dump(
            model,
            filename=f"./ML_models/PC_{model.__class__.__name__}_baseline.
model"
        )
        print(f"Model saved at: ./ML_models/PC_{model.__class__.__name__}_
baseline.model")
```

Light Gradient Boosting

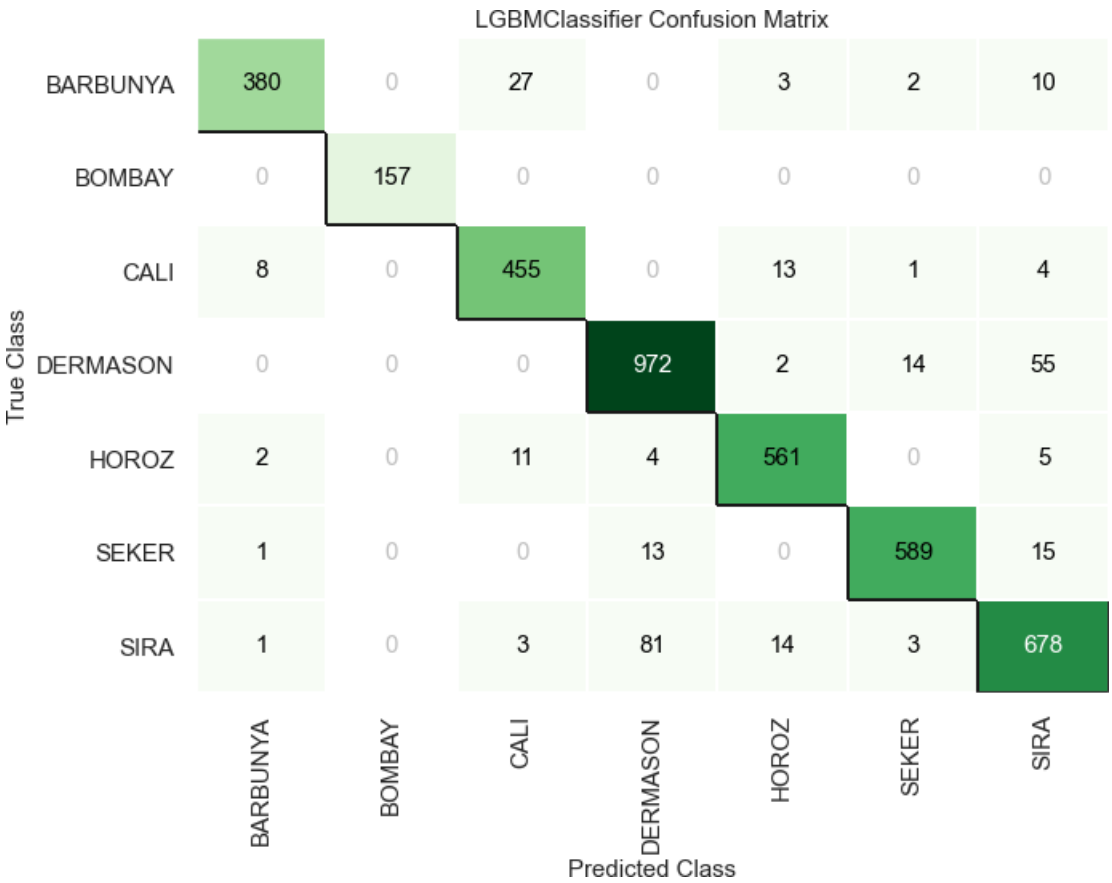
```
%%time
lgbm = create_model('lightgbm')
```

<pandas.io.formats.style.Styler at 0x1ebd4a3a5b0>Wall time:

6.24 s

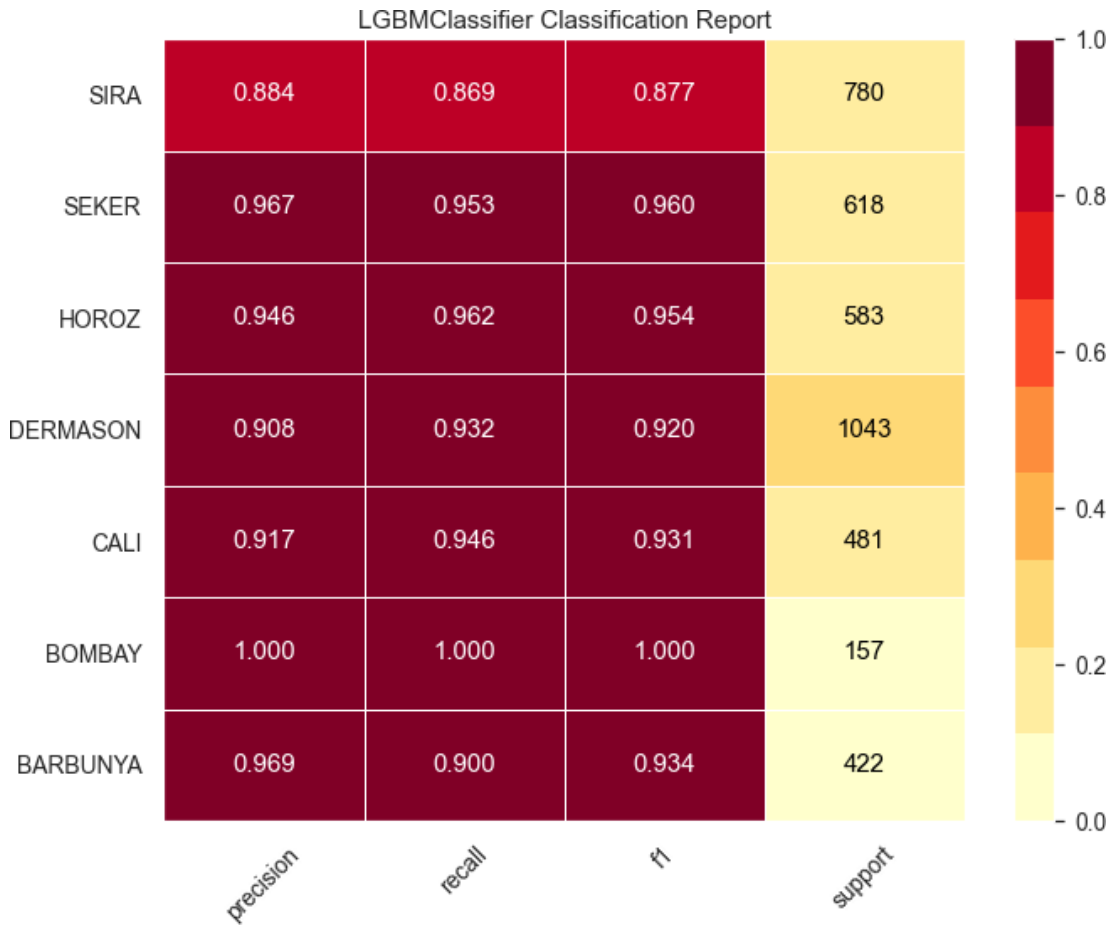
Plotting different plots like confusion matrix and auc is also very easy as simple as 1 line of code. We look at few plots, to asses performance

```
plot_model(lgbm, plot='confusion_matrix')
```



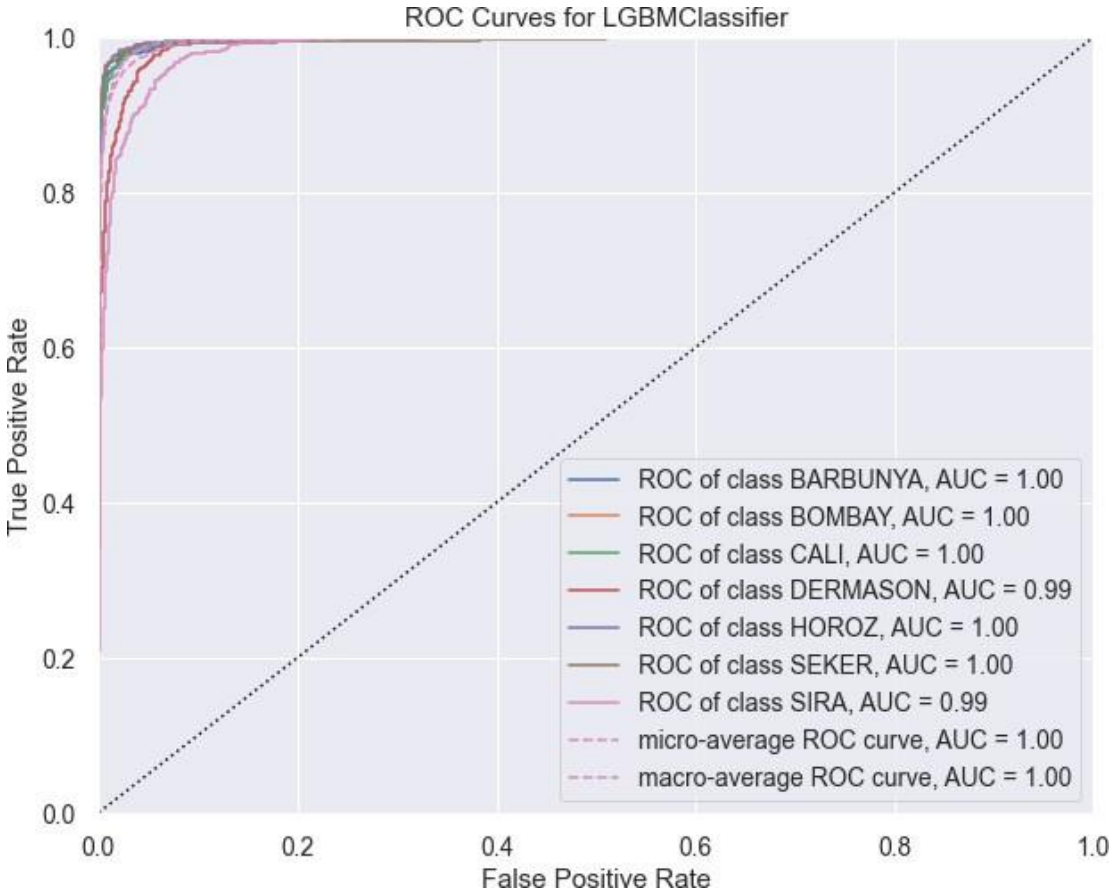

```
# plot(estimator=lgbm, plot_type='confusion_matrix', dst=f"./ML_results/CF
_{lgbm.__class__.__name__}.png")
```

```
plot_model(lgbm, plot='class_report')
```




```
plot(estimator=lgbm, plot_type='class_report', dst=f'./ML_results/ClassReport_{lgbm.__class__name_}.png')
```

```
plot_model(estimator=lgbm, plot='auc')
```




```
plot(lgbm, plot_type='auc', dst=f"./ML_results/AUC_{lgbm.__class__.__name__}.png")
```

```
save(model=lgbm)
```

Model saved at: ./ML_models/PC_LGBMClassifier_baseline.model

`Obvservation`

- The baseline lightgbm performs well with an f1 of approx ~ 93
- Our model seems to be confused between DERMASON and SIRA varieties
- Our precision, recall and f1 for each class is more than 86, which is also a good indication

<u>Tuning The LightGBM</u>

Tuning the LightGBMClassifier. We can do Grid-search, Random-search as these are the good old hyper paramter tuning methods. But there's a more efficient tuning method using Bayesian Hyperparamter tuning. Here's a one line summary of what bayesian searchis:

> Build a probability model of the objective function and use it to select the most promising hyperparameters to evaluate in the true objective function.

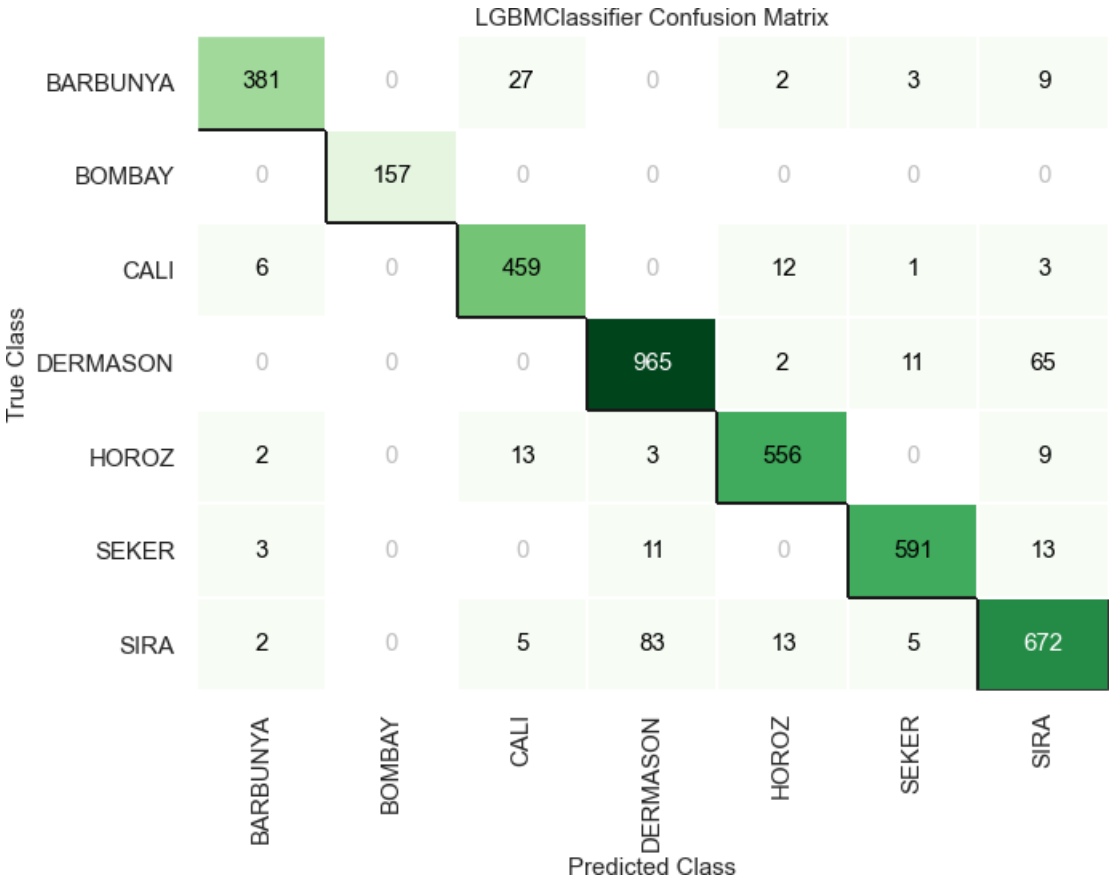
```
%%time
tuned_lgbm, tuner = tune_model(
    estimator=lgbm, search_library="scikit-optimize",
```

```
        n_iter=25,
        optimize='f1', return_tuner=True
)
```

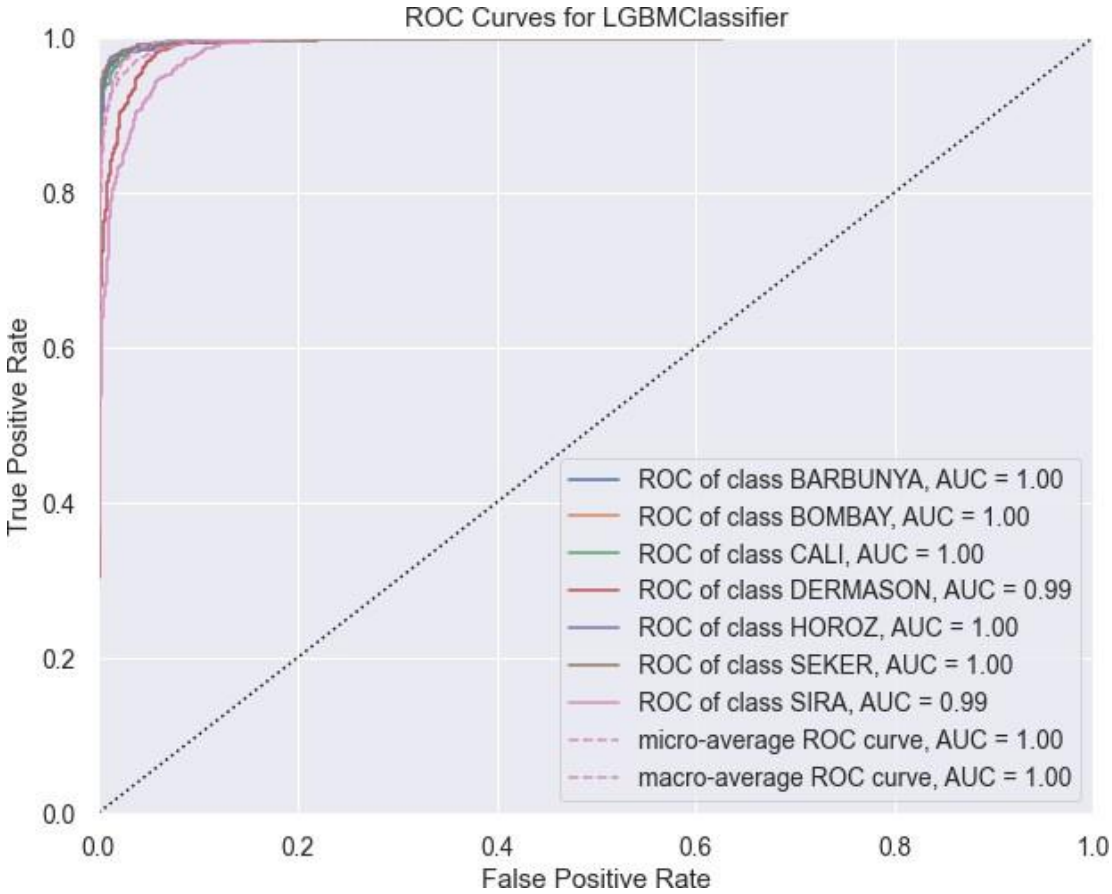
<pandas.io.formats.style.Styler at 0x1ebd6374c70>

Wall time: 2min 17s

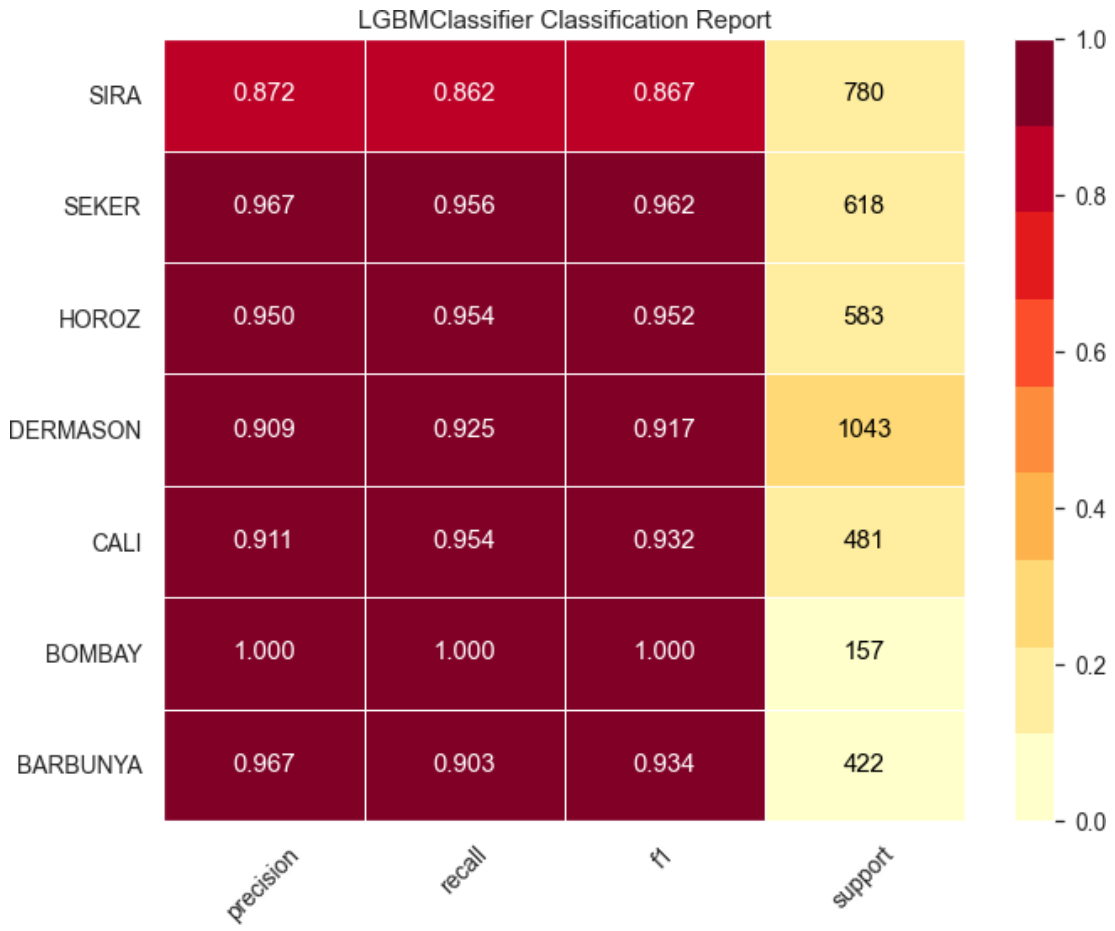
```
plot_model(tuned_lgbm, plot='confusion_matrix')
```




```
plot_model(tuned_lgbm, plot='auc')
```




```
plot_model(tuned_lgbm, plot='class_report')
```




```
plot_model(tuned_lgbm, plot='parameter')
```

	Parameters
boosting_type	gbdt
class_weight	None
colsample_bytree	1.0
importance_type	split
learning_rate	0.07351087677623395
max_depth	-1
min_child_samples	8
min_child_weight	0.001
min_split_gain	0.308079603643812
n_estimators	132
n_jobs	-1
num_leaves	109
objective	None
random_state	8669
reg_alpha	2.878849039397276
reg_lambda	2.7700803937357488e-08
silent	warn
subsample	1.0
subsample_for_bin	200000
subsample_freq	0
bagging_fraction	0.740705467313804
bagging_freq	2
feature_fraction	0.768581357957563

We can access the search space easily too:

```
params = tuner.get_params()
params['search_spaces']
```

```
{'actual_estimator__num_leaves': Integer(low=2, high=256, prior='uniform',
transform='normalize'),
  'actual_estimator__learning_rate': Real(low=1e-06, high=0.5, prior='log-uniform',
transform='normalize'),
  'actual_estimator__n_estimators': Integer(low=10, high=300, prior='uniform',
transform='normalize'),
  'actual_estimator__min_split_gain': Real(low=0, high=1, prior='uniform',
transform='normalize'),
  'actual_estimator__reg_alpha': Real(low=1e-10, high=10, prior='log-uniform',
transform='normalize'),
  'actual_estimator__reg_lambda': Real(low=1e-10, high=10, prior='log-uniform',
transform='normalize'),
  'actual_estimator__feature_fraction': Real(low=0.4, high=1, prior='uniform',
transform='normalize'),
  'actual_estimator__bagging_fraction': Real(low=0.4, high=1, prior='uniform',
transform='normalize'),
  'actual_estimator__bagging_freq': Integer(low=0, high=7, prior='uniform',
transform='normalize'),
  'actual_estimator__min_child_samples': Integer(low=1, high=100, prior='uniform',
transform='normalize')}
```

```
tuner.best_params_
```

```
OrderedDict([('actual_estimator__bagging_fraction', 0.740705467313804),
             ('actual_estimator__bagging_freq', 2),
             ('actual_estimator__feature_fraction', 0.768581357957563),
             ('actual_estimator__learning_rate', 0.07351087677623395),
             ('actual_estimator__min_child_samples', 8),
             ('actual_estimator__min_split_gain', 0.308079603643812),
             ('actual_estimator__n_estimators', 132),
             ('actual_estimator__num_leaves', 109),
             ('actual_estimator__reg_alpha', 2.878849039397276), ('actual_estimator__
             reg_lambda', 2.7700803937357488e-08)])
```

All of the class stuff we wrote in the previous notebook where we were doing custom tuning is now reduced to just a single line of code and we have full control over it. The default search-space provided in pycaret is good enough for tuning, but we also pass a custom grid like we did in our previous notebook

```
Model saved at: ./ML_models/PC_LGBMClassifier_BayesSearchCV.model
Tuner saved at: ./ML_results/PC_LGBMClassifier_BayesSearchCV_params.json
```

We can also let pycaret choose for us if we don't want to use bayesian search

```
tuned_lgbm_auto, tuner_auto = tune_model(
    estimator=lgbm,
    choose_better=True, optimize='f1',
    return_tuner=True
)
```

```
<pandas.io.formats.style.Styler at 0x1ebd4d9bc40>tuner_auto.best_params_

{'actual_estimator_reg_lambda': 2, 'actual_estimator_
 reg_alpha': 1e-07,'actual_estimator__num_leaves': 256,
 'actual_estimator__n_estimators': 70,
 'actual_estimator__min_split_gain': 0.9,
 'actual_estimator__min_child_samples': 91,
 'actual_estimator__learning_rate': 0.15,
 'actual_estimator__feature_fraction': 0.6,
 'actual_estimator__bagging_freq': 4,
 'actual_estimator__bagging_fraction': 1.0}tuner_auto.__class_

sklearn.model_selection._search.RandomizedSearchCV
```

Obvservation

- Setting choose_better=True, it uses RandomSearchCV instead of BayesSearchCV
- There's no drastic difference between the two. Infact, random search is just randomly searching for the parameters.
- So, BayesSearch is better than random search in the sense that it uses a probability distribution rather than it just doing random search, which is more efficient as the probability guides the search

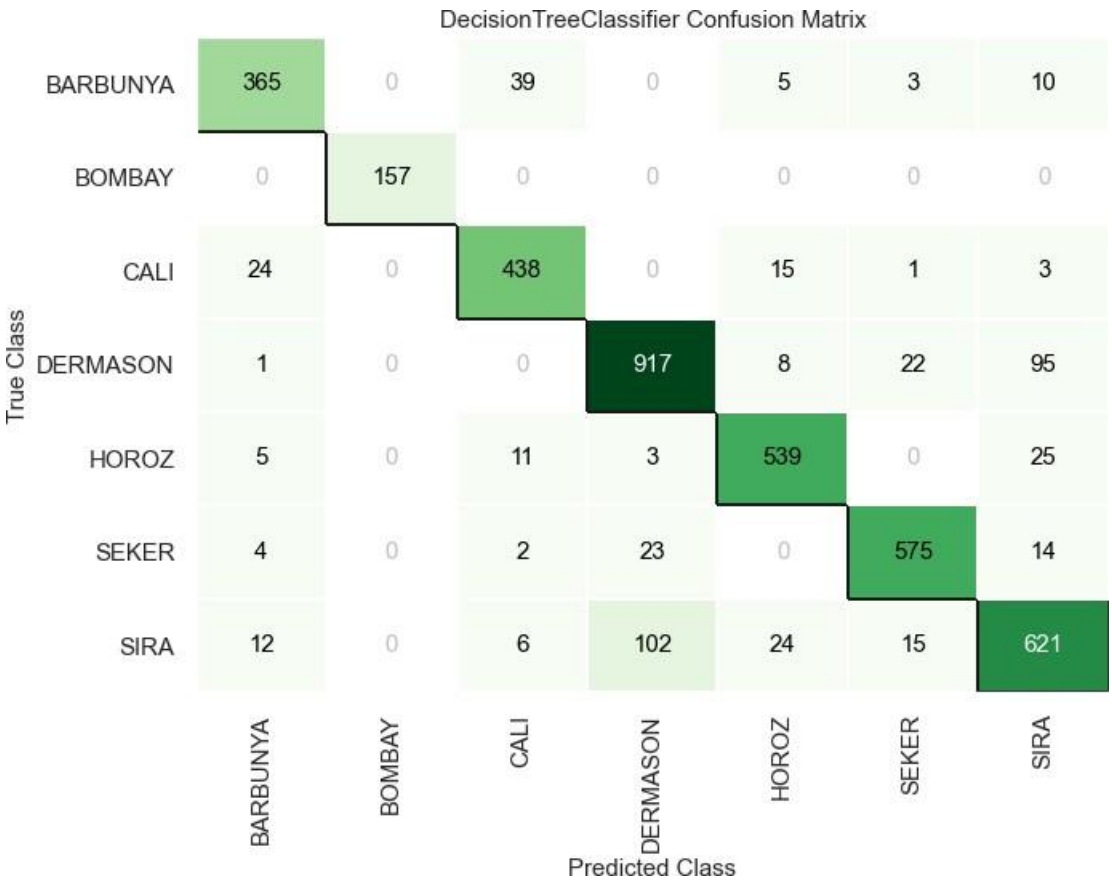
Results - 1

- The highest accuracy score mentioned in the paper which is 93.13 %. In the paper, they get to it through SVM with a polynomial kernel.
- We have reached an accuracy of 92.72 % with a LightGBM, which you can say is fasterto train than the SVM whose time complexity would be Quadratic
- I have used no preprocessing or transformation or fixed the target imbalance, similar to the paper
- I have used all the 16 features

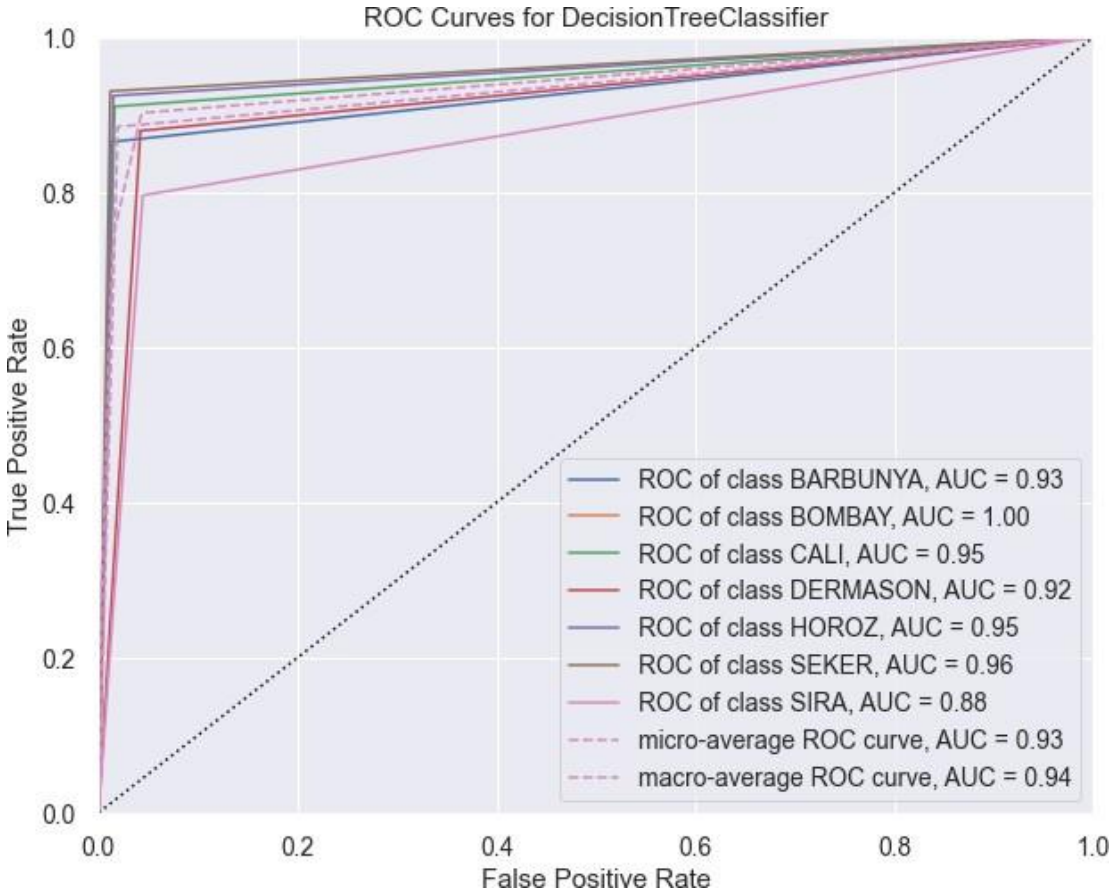
Other models mentioned in the paper

Decision Tree

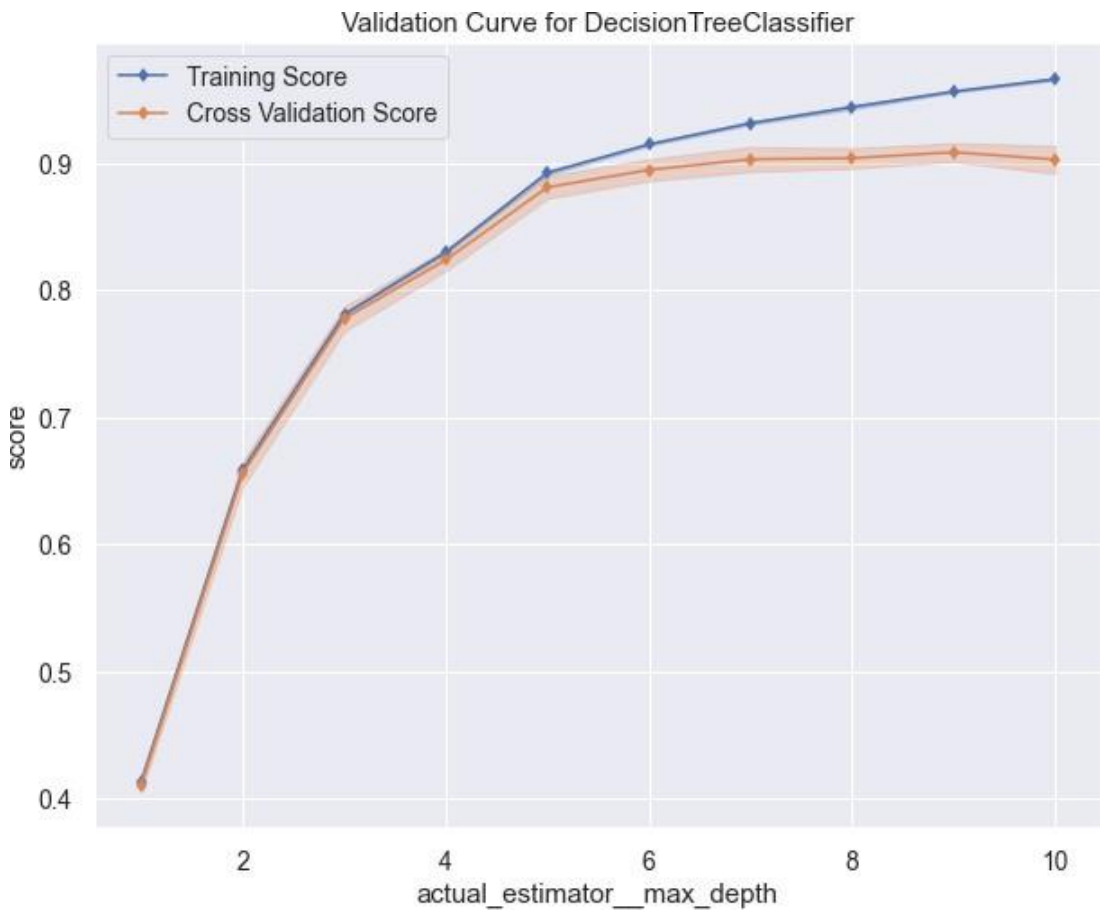
```
%%time
dt = create_model('dt')

<pandas.io.formats.style.Styler at 0x1ebd4d9b640>

Wall time: 879 ms

plot_model(dt, plot='confusion_matrix')
```




```
plot_model(dt, plot='auc')
```




```
plot_model(dt, plot='vc')
```

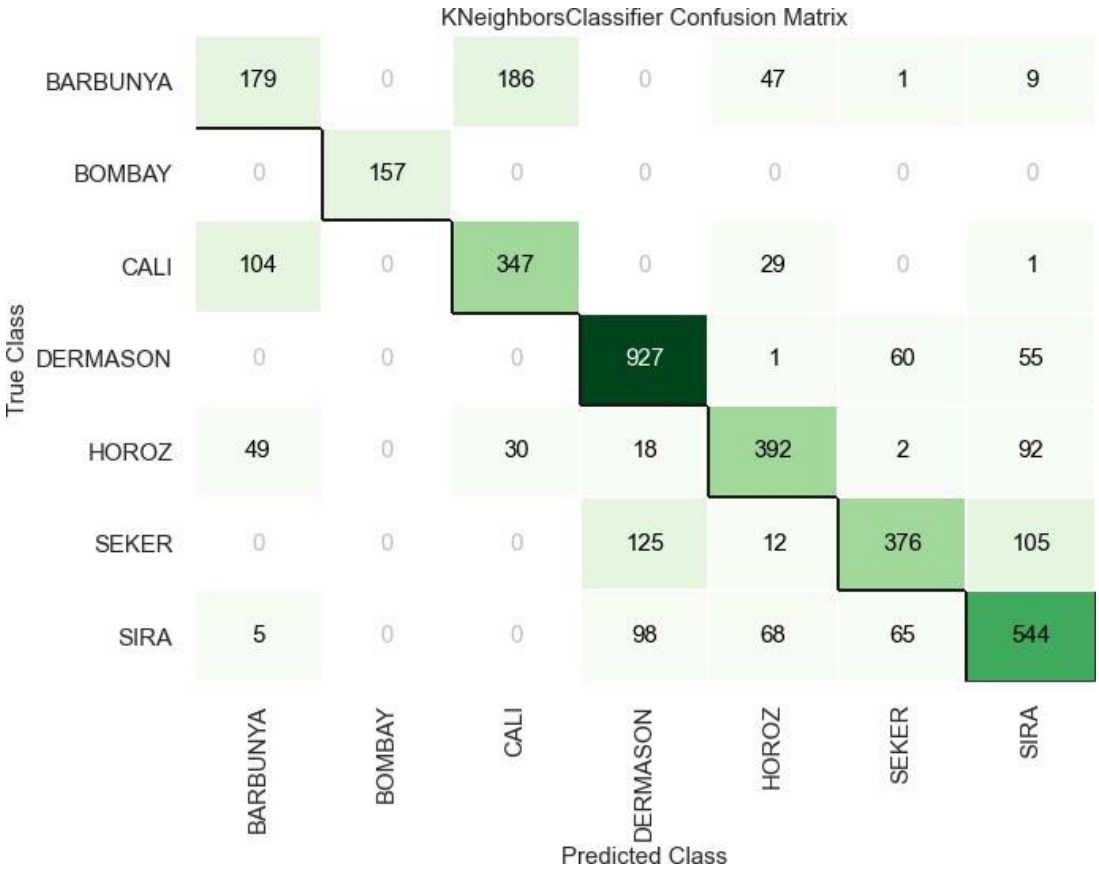


KNN

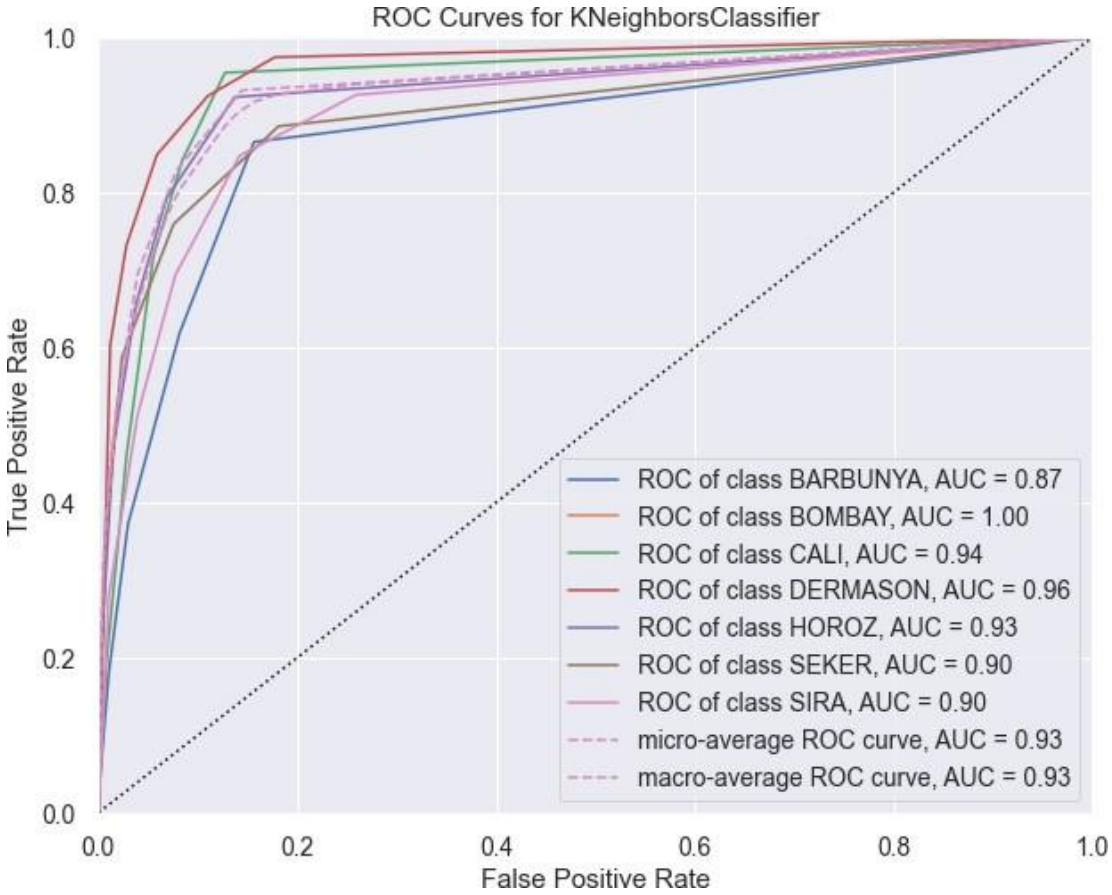
knn = create_model('knn')

<pandas.io.formats.style.Styler at 0x1ebd6147340>

```
plot_model(knn, plot='confusion_matrix')
```



plot_model(knn, plot='auc')

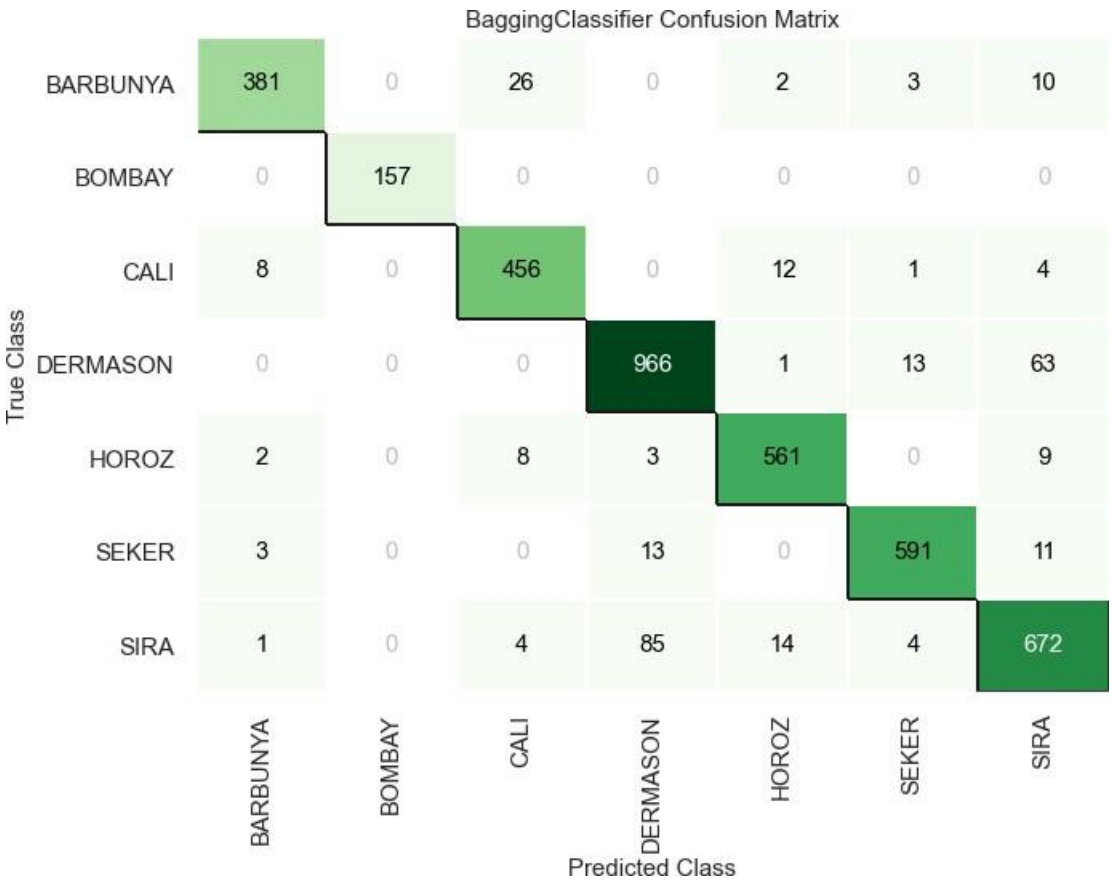


Ensembling

Ensembled Light Gradient Boosting

ensembled_lgbm = ensemble_model(tuned_lgbm, optimize='f1')

<pandas.io.formats.style.Styler at 0x1ebd4d9bac0> plot_model(ensembled_lgbm, plot='confusion_matrix')

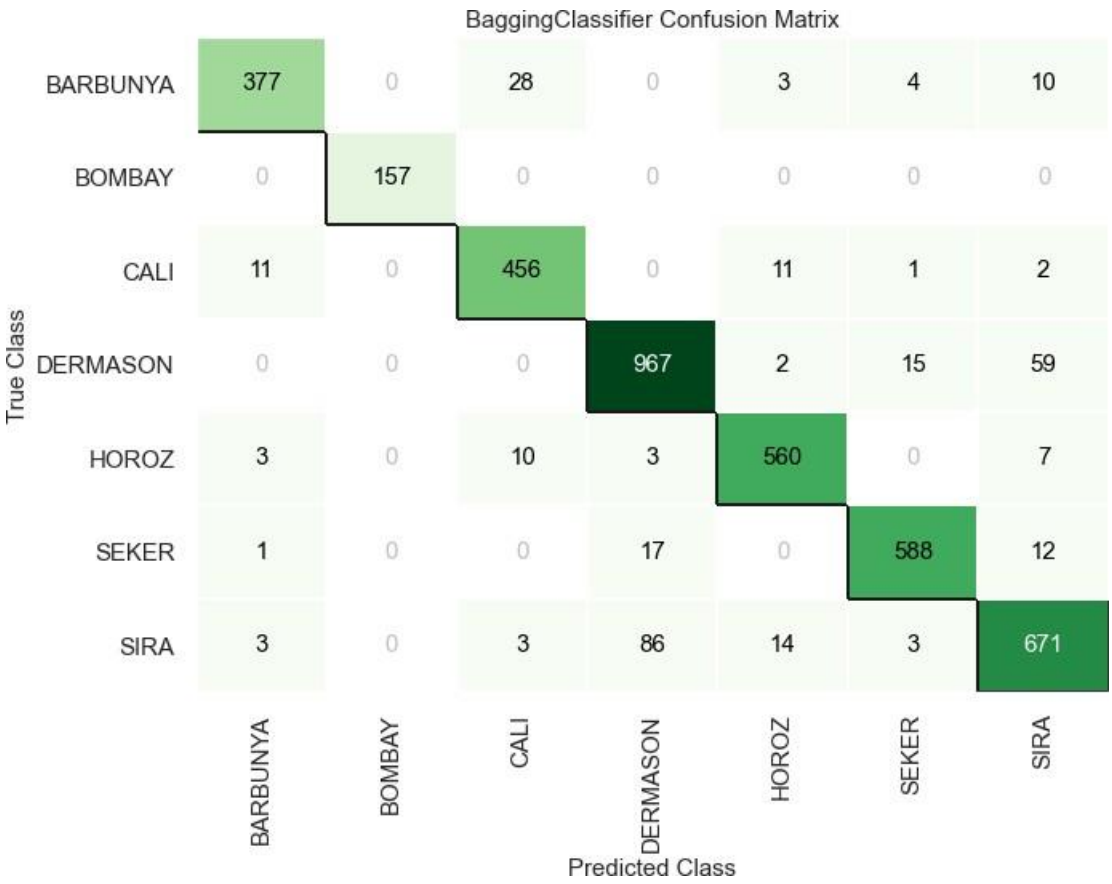


Ensembled Decision Tree

```
ensembled_dt = ensemble_model(dt, n_estimators=100, optimize='f1')
```

```
<pandas.io.formats.style.Styler at 0x1ebd6392910>
```

```
plot_model(ensembled_dt, plot='confusion_matrix')
```



save(model=ensembled_dt)

Model saved at: ./ML_models/PC_BaggingClassifier_baseline.model

Observation

- Both the ensembled LightGBM and Decision tree, do not do well than out tuned LightGBm model.

Blending

We will try blending the tuned light gbm and ensembled decision tree model

blended_lgbm_dt = blend_models(estimator_list=[tuned_lgbm, ensembled_dt], optimize='f1')

tuned_blended_lgbm_dt = tune_model(estimator=blended_lgbm_dt, search_library='scikit-optimize', optimize='f1')

Results - 2

- The highest accuracy score mentioned in the paper which is 93.13 %. In the paper, they get to it through SVM with a polynomial kernel.
- We have reached an accuracy of 92.78 % with a blended model of tuned lightgbm + ensembled decision tree,
- I have used no preprocessing or transformation or fixed the target imbalance, similar to the paper
- I have used all the 16 features

Experiment with transformed data

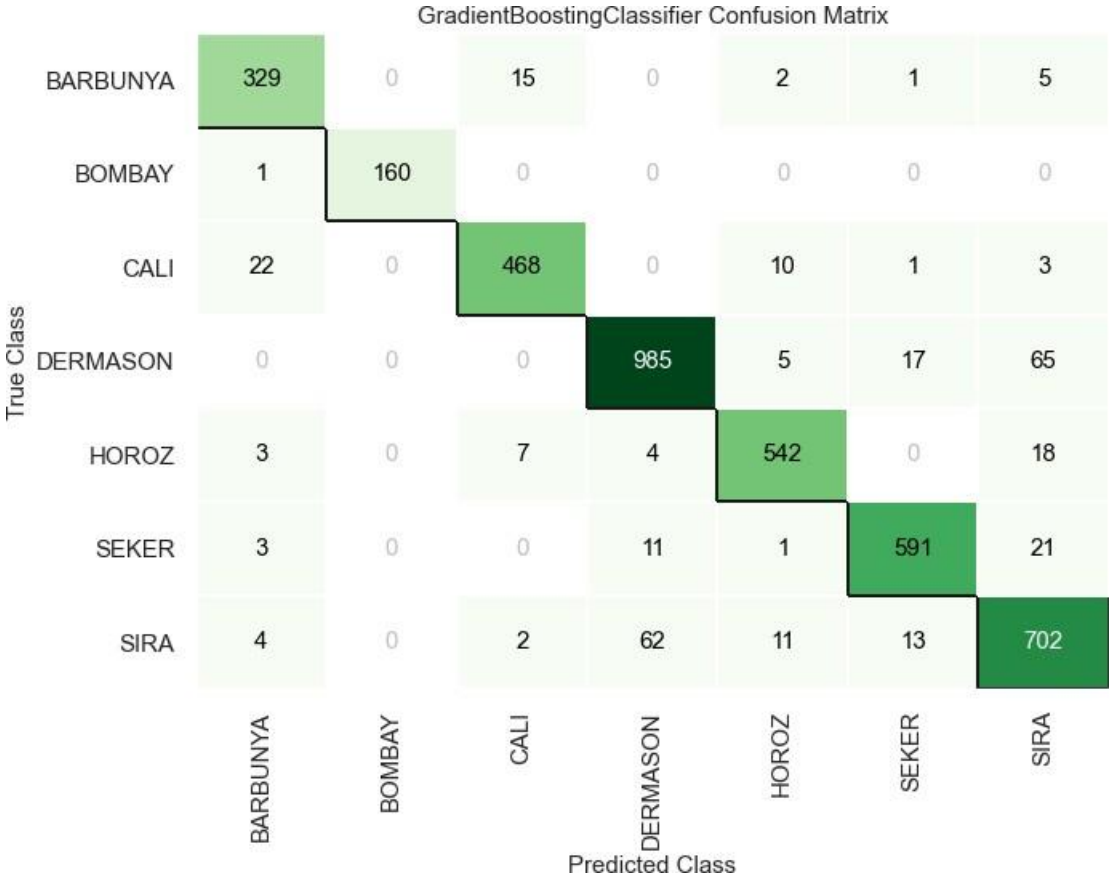
```
exp = setup(
        data=df,
        target='Class',
        train_size=0.7,
        experiment_name='baseline_with_transforms', remove_perfect_collinearity=False,
        fix_imbalance=True, normalize=True,
transformation=True
)
```

```
<pandas.io.formats.style.Styler at 0x1ebd4e6a0d0>
```

```
best_model = compare_models()
```

```
<pandas.io.formats.style.Styler at 0x1ebd639c9a0>
```

```
plot_model(best_model, plot='confusion_matrix')
```




```
%%time
tuned_lgbm_transformed, tuner_lgbm_transformed = tune_model(
        best_model,
        optimize='f1', search_library='scikit-
        optimize',return_tuner=True
)
```

Results - 3

- The highest accuracy score mentioned in the paper which is 93.13 %. In the paper, they get to it through SVM with a polynomial kernel.
- We have reached an accuracy of 92.9 % ~ 93 % with lightgbm after preprocessing the data by normalizing it and fixing the imbalance
- I have used all the 16 features

Conclusion

- The best model found was transformed data + Light Gradient Boosting
- We can also go with simple Light Gradient Boosting as the difference between them is not that significant

Advanced Modelling

We did the basic modelling and found out that tuned Light Gradient Boosting machine performs really well with a F1-score of 0.93. Now we move on to some advanced modelling. We will use a neural network for doing so.

Artificial Neural Network

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm.

How do neural networks work?

Think of each individual node as its own linear regression model, composed of input data, weights, a bias (or threshold), and an output. The formula would look something like this:

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$

$$\text{output} = f(x) = \begin{cases} 1 \text{ if } \sum w_1 x_1 + b \geq 0 \\ 0 \text{ if } \sum w_1 x_1 + b < 0 \end{cases}$$

Once an input layer is determined, weights are assigned. These weights help determine the importance of any given variable, with larger ones contributing more significantly to the output compared to other inputs. All inputs are then multiplied by their respective weights and then summed. Afterward, the output is passed through an activation function, which determines the output. If that output exceeds a given threshold, it "fires" (or activates) the node, passing data to the next layer in the network. This results in the output of one node becoming in the input of the next node. This process of passing data from one layer to the next layer defines this neural network as a feedforward network.

Let's break down what one single node might look like using binary values. We can apply this concept to a more tangible example, like whether you should go surfing (Yes: 1, No: 0). The decision to go or not to go is our predicted outcome, or y-hat. Let's assume that there are three factors influencing your decision-making:

1. Are the waves good? (Yes: 1, No: 0)
2. Is the line-up empty? (Yes: 1, No: 0)
3. Has there been a recent shark attack? (Yes: 0, No: 1)

Then, let's assume the following, giving us the following inputs:

- X1 = 1, since the waves are pumping
- X2 = 0, since the crowds are out
- X3 = 1, since there hasn't been a recent shark attack

Now, we need to assign some weights to determine importance. Larger weights signify that particular variables are of greater importance to the decision or outcome.

- W1 = 5, since large swells don't come around often
- W2 = 2, since you're used to the crowds
- W3 = 4, since you have a fear of sharks

Finally, we'll also assume a threshold value of 3, which would translate to a bias value of –3. With all the various inputs, we can start to plug in values into the formula to get the desired output.

Y-hat = (1*5) + (0*2) + (1*4) – 3 = 6

If we use the activation function from the beginning of this section, we can determine that the output of this node would be 1, since 6 is greater than 0. In this instance, you would go surfing; but if we adjust the weights or the threshold, we can achieve different outcomes from the model. When we observe one decision, like in the above example, we can see how a neural network could make increasingly complex decisions depending on the output of previous decisions or layers.

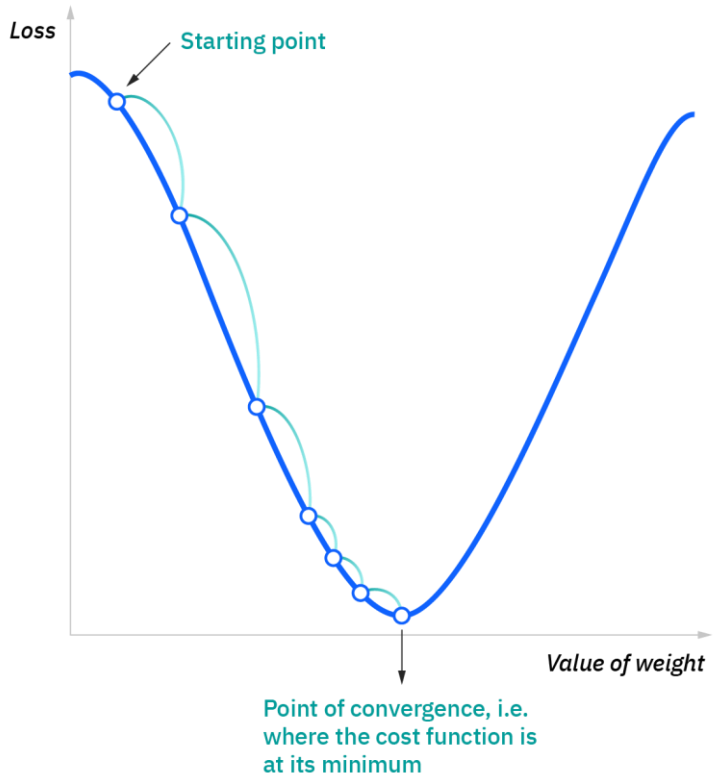
In the example above, we used perceptrons to illustrate some of the mathematics at play here, but neural networks leverage sigmoid neurons, which are distinguished by having values between 0 and 1. Since neural networks behave similarly to decision trees, cascading data from one node to another, having x values between 0 and 1 will reduce the impact of any given change of a single variable on the output of any given node, and subsequently, the output of the neural network.

As we start to think about more practical use cases for neural networks, like image recognition or classification, we'll leverage supervised learning, or labeled datasets, to train the algorithm. As we train the model, we'll want to evaluate its accuracy using a cost (or loss) function. This is also commonly referred to as the mean squared error (MSE). In the equation below,

- *i* represents the index of the sample,
- y-hat is the predicted outcome,
- y is the actual value, and
- *m* is the number of samples.

$$\text{Cost Function} = \text{MSE} = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y} - y)^2$$

Ultimately, the goal is to minimize our cost function to ensure correctness of fit for any given observation. As the model adjusts its weights and bias, it uses the cost function and reinforcement learning to reach the point of convergence, or the local minimum. The process in which the algorithm adjusts its weights is through gradient descent, allowing the model to determine the direction to take to reduce errors (or minimize the cost function). With each training example, the parameters of the model adjust to gradually converge at the minimum.



Most deep neural networks are feedforward, meaning they flow in one direction only, from input to output. However, you can also train your model through backpropagation; that is, move in the opposite direction from output to input. Backpropagation allows us to calculate and attribute the error associated with each neuron, allowing us to adjust and fit the parameters of the model(s) appropriately.

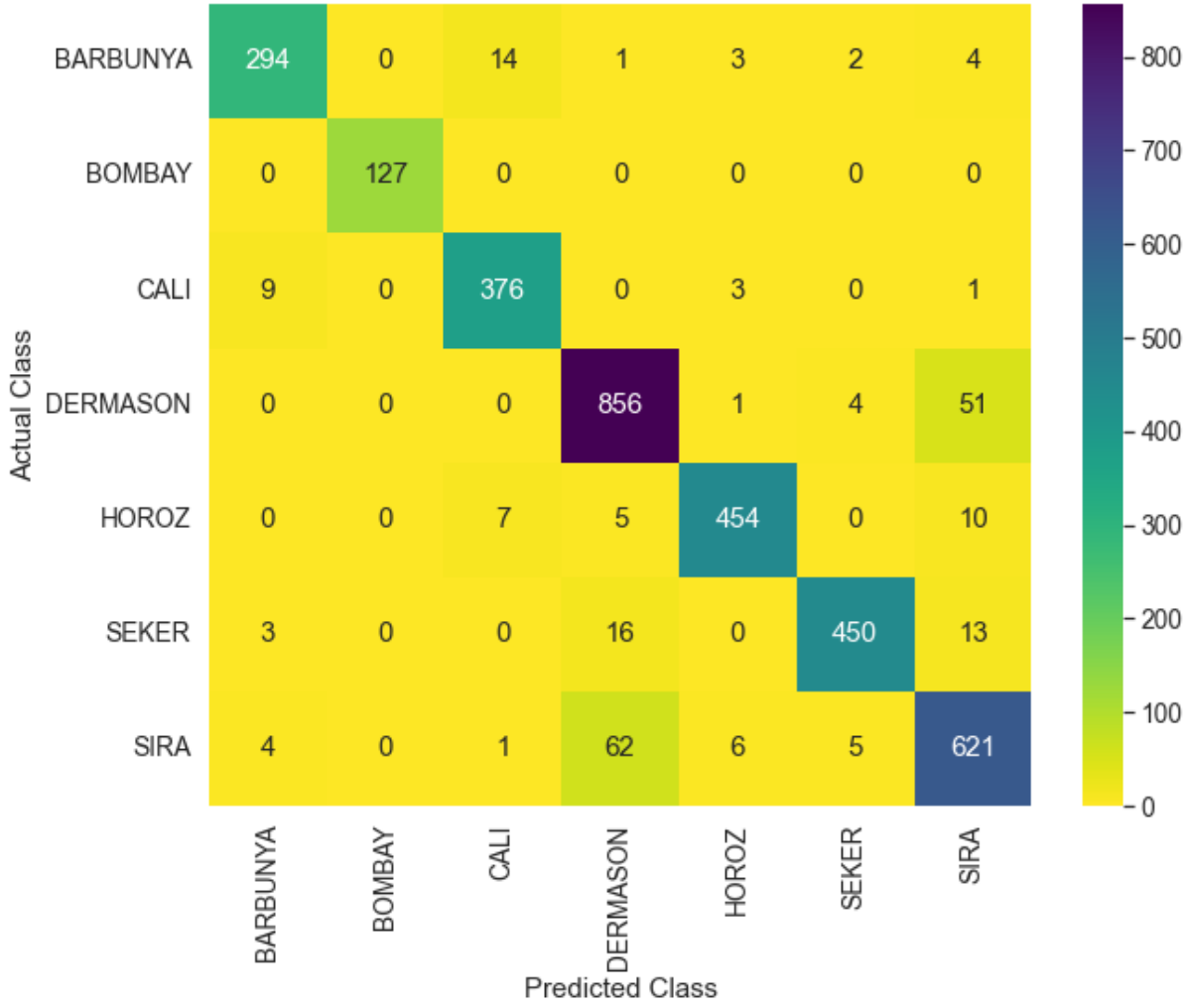
Vanilla Net

So, for our modelling I call our model “Vanilla Net”

1) It’s a 2 layer NN with relu activation.
2) The first hidden layer has 512 nodes
3) The second one has 256 nodes
4) Both the layers use `relu` activation
5) Optimizer: `Adam` with a `lr=3e-4`
6) loss: `SparseCategoricalCrossEntropy(logits=True)`
7) Epochs: 20

Results

With resampling (Oversampling with SMOTE) We even beat the best accuracy and f1-score in the paper with an accuracy of **93.39%** & f1-score of **0.9340**



Deployment

What is Model Deployment?

Deployment is the method by which you integrate a machine learning model into an existing production environment to make practical business decisions based on data. It is one of the last stages in the machine learning life cycle and can be one of the most cumbersome. Often, an organization's IT systems are incompatible with traditional model-building languages, forcing data scientists and programmers to spend valuable time and brainpower rewriting them.

Why is Model Deployment Important?

In order to start using a model for practical decision-making, it needs to be effectively deployed into production. If you cannot reliably get practical insights from your model, then the impact of the model is severely limited.

Model deployment is one of the most difficult processes of gaining value from machine learning. It requires coordination between data scientists, IT teams, software developers, and business professionals to ensure the model works reliably in the organization's production environment. This presents a major challenge because there is often a discrepancy between the programming language in which a machine learning model is written and the languages your production system can understand, and re-coding the model can extend the project timeline by weeks or months.

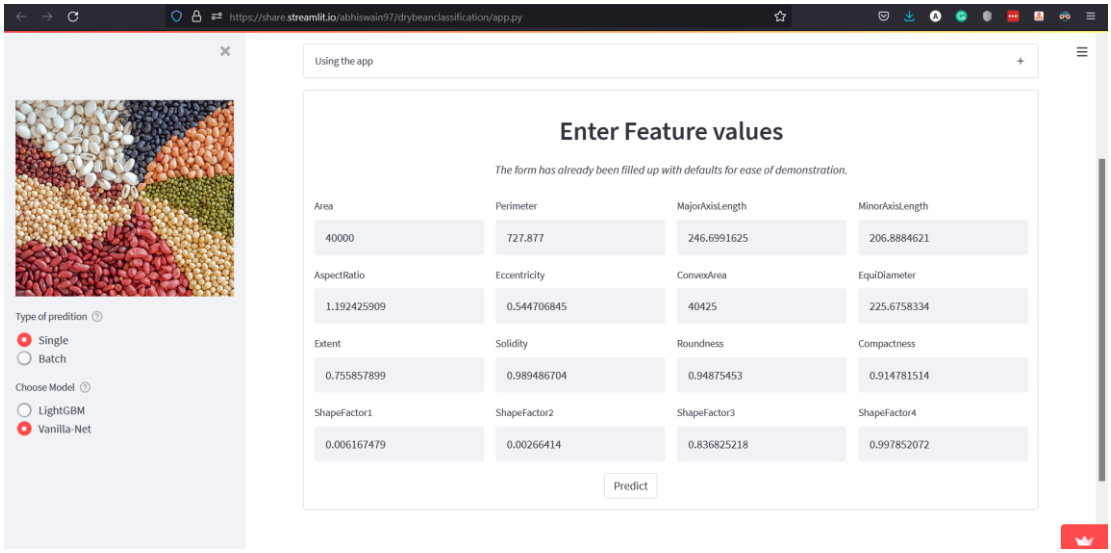
In order to get the most value out of machine learning models, it is important to seamlessly deploy them into production so a business can start using them to make practical decisions.

Deploying our app

The app is deployed at: [Dry bean classifier]

I have created a Streamlit app named “Dry bean classifier”. It has two ways you can make classification.

1. Single prediction

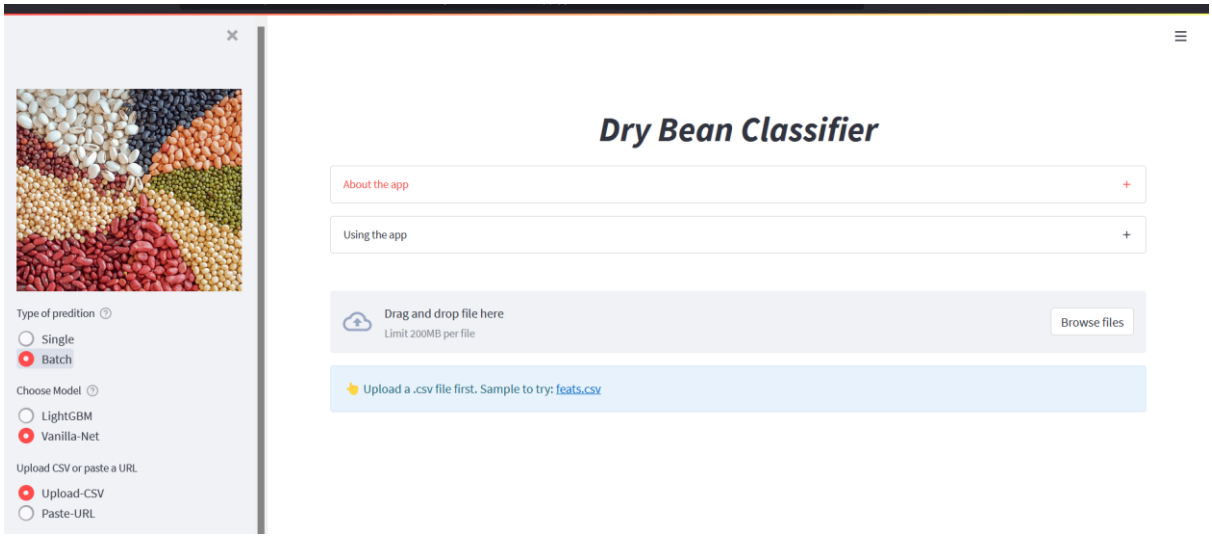


The form has been pre-filled with default values for ease of demonstration.

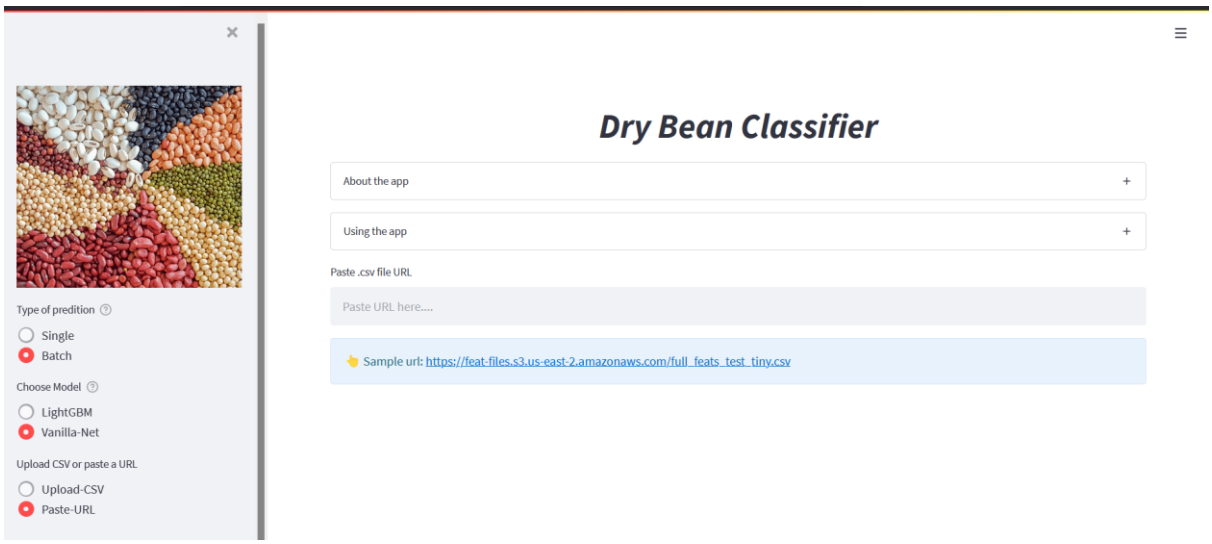
2. Batch prediction using .csv file

You can also use a .csv file or a link to a csv file for doing prediction in bulk

Upload CSV directly

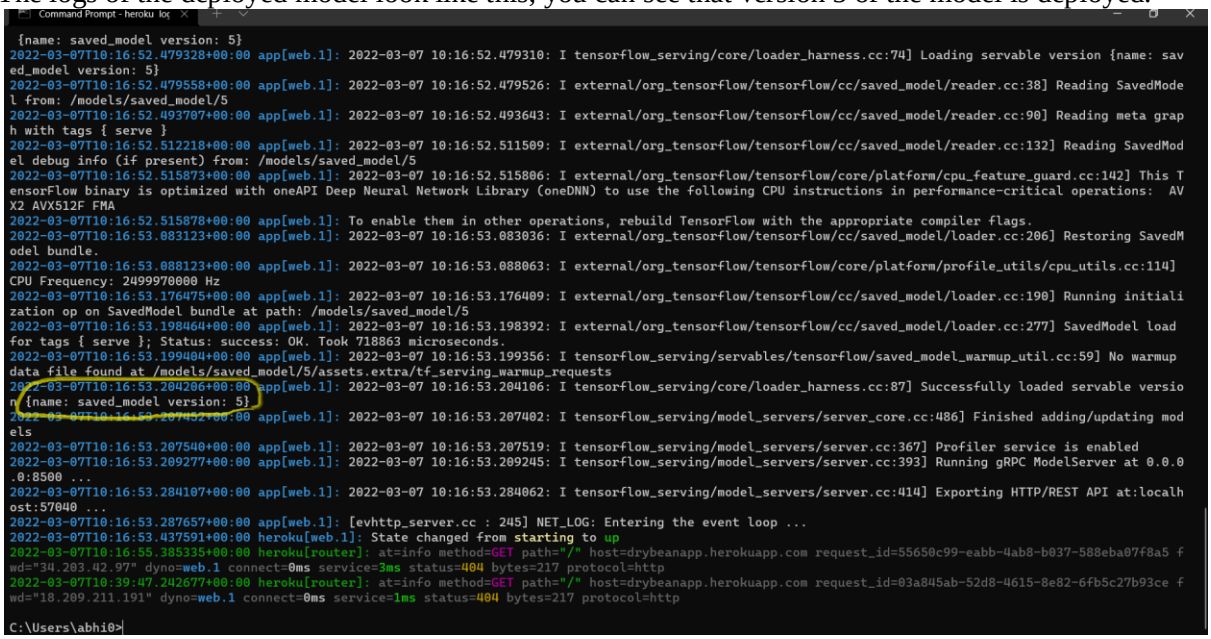


Paste a link to a CSV file



Deploying the Tensorflow model

So, loading the TF model along with the app in Streamlit makes the app slower. So, to solve this I have the Vanilla Net model served at a docker container on Heroku using Tensorflow serving.
The logs of the deployed model look like this, you can see that version 5 of the model is deployed.



```
Command Prompt - heroku logs
 {name: saved_model version: 5}
2022-03-07T10:16:52.479328+00:00 app[web.1]: 2022-03-07 10:16:52.479310: I tensorflow_serving/core/loader_harness.cc:74] Loading servable version {name: sav
ed_model version: 5}
2022-03-07T10:16:52.479558+00:00 app[web.1]: 2022-03-07 10:16:52.479526: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:38] Reading SavedMode
l from: /models/saved_model/5
2022-03-07T10:16:52.493707+00:00 app[web.1]: 2022-03-07 10:16:52.493643: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:90] Reading meta grap
h with tags { serve }
2022-03-07T10:16:52.512218+00:00 app[web.1]: 2022-03-07 10:16:52.511509: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:132] Reading SavedMod
el debug info (if present) from: /models/saved_model/5
2022-03-07T10:16:52.515873+00:00 app[web.1]: 2022-03-07 10:16:52.515806: I external/org_tensorflow/tensorflow/core/platform/cpu_feature_guard.cc:142] This T
ensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AV
X2 AVX512F FMA
2022-03-07T10:16:52.515878+00:00 app[web.1]: To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-03-07T10:16:53.083123+00:00 app[web.1]: 2022-03-07 10:16:53.083036: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:206] Restoring SavedM
odel bundle.
2022-03-07T10:16:53.088123+00:00 app[web.1]: 2022-03-07 10:16:53.088063: I external/org_tensorflow/tensorflow/core/platform/profile_utils/cpu_utils.cc:114]
CPU Frequency: 2499970000 Hz
2022-03-07T10:16:53.176475+00:00 app[web.1]: 2022-03-07 10:16:53.176409: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:190] Running initiali
zation op on SavedModel bundle at path: /models/saved_model/5
2022-03-07T10:16:53.198464+00:00 app[web.1]: 2022-03-07 10:16:53.198392: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:277] SavedModel load
for tags { serve }; Status: success: OK. Took 718863 microseconds.
2022-03-07T10:16:53.199404+00:00 app[web.1]: 2022-03-07 10:16:53.199356: I tensorflow_serving/servables/tensorflow/saved_model_warmup_util.cc:59] No warmup
data file found at /models/saved_model/5/assets.extra/tf_serving_warmup_requests
2022-03-07T10:16:53.204206+00:00 app[web.1]: 2022-03-07 10:16:53.204106: I tensorflow_serving/core/loader_harness.cc:87] Successfully loaded servable versio
n {name: saved_model version: 5}
2022-03-07T10:16:53.207452+00:00 app[web.1]: 2022-03-07 10:16:53.207402: I tensorflow_serving/model_servers/server_core.cc:486] Finished adding/updating mod
els
2022-03-07T10:16:53.207540+00:00 app[web.1]: 2022-03-07 10:16:53.207519: I tensorflow_serving/model_servers/server.cc:367] Profiler service is enabled
2022-03-07T10:16:53.209277+00:00 app[web.1]: 2022-03-07 10:16:53.209245: I tensorflow_serving/model_servers/server.cc:393] Running gRPC ModelServer at 0.0.0
.0:8500 ...
2022-03-07T10:16:53.284107+00:00 app[web.1]: 2022-03-07 10:16:53.284062: I tensorflow_serving/model_servers/server.cc:414] Exporting HTTP/REST API at:localh
ost:57040 ...
2022-03-07T10:16:53.287657+00:00 app[web.1]: [evhttp_server.cc : 245] NET_LOG: Entering the event loop ...
2022-03-07T10:16:53.437591+00:00 heroku[web.1]: State changed from starting to up
2022-03-07T10:16:55.385335+00:00 heroku[router]: at=info method=GET path="/" host=drybeanapp.herokuapp.com request_id=55650c99-eabb-4ab8-b037-588eba07f8a5 f
wd="34.203.42.97" dyno=web.1 connect=0ms service=3ms status=404 bytes=217 protocol=http
2022-03-07T10:39:47.242677+00:00 heroku[router]: at=info method=GET path="/" host=drybeanapp.herokuapp.com request_id=03a845ab-52d8-4615-8e82-6fb5c27b93ce f
wd="18.209.211.191" dyno=web.1 connect=0ms service=1ms status=404 bytes=217 protocol=http

C:\Users\abhi0>
```

So, when you click on predict button the app is just making calls to this endpoint! The interesting thing about it is that all you need to do is just hit this endpoint for the prediction, so you can do it on your own without the app by a simple python script.

Conclusion

In this whole document I have listed out the methods and processes I used to perform end to end dry bean classification.

The app is deployed at: Dry bean classifier

The repository containing code: https://github.com/Abhiswain97/DryBeanClassification