Dry Beans Classification

IART G03

Ana Inês Oliveira de Barros - up201806593
João de Jesus Costa - up201806560
João Lucas Silva Martins - up201806436

Problem specification

Task

Classify between seven different registered varieties of dry beans with similar features, based on the features collected.

The beans can be of any of the following **7 classes:** Seker, Barbunya, Bombay, Cali, Dermosan, Horoz, and Sira.

Experience

A dataset, with **16 different attributes**, containing the information collected about dry beans and their classification.

The dataset doesn't have any missing values, but the population is unbalanced.

Performance

Since the population is significantly **unbalanced**, we'll compare the performance of classifiers using the **F1-Score** and the **training/classification times**.

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness	ShapeFactor1	ShapeFactor2	ShapeFactor3	ShapeFactor4
count	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000
mean	53048.284549	855.283459	320.141867	202.270714	1.583242	0.750895	53768.200206	253.064220	0.749733	0.987143	0.873282	0.799864	0.006564	0.001716	0.643590	0.995063
std	29324.095717	214.289696	85.694186	44.970091	0.246678	0.092002	29774.915817	59.177120	0.049086	0.004660	0.059520	0.061713	0.001128	0.000596	0.098996	0.004366
min	20420.000000	524.736000	183.601165	122.512653	1.024868	0.218951	20684.000000	161.243764	0.555315	0.919246	0.489618	0.640577	0.002778	0.000564	0.410339	0.947687
50%	44652.000000	794.941000	296.883367	192.431733	1.551124	0.764441	45178.000000	238.438026	0.759859	0.988283	0.883157	0.801277	0.006645	0.001694	0.642044	0.996386
max	254616.000000	1985.370000	738.860153	460.198497	2.430306	0.911423	263261.000000	569.374358	0.866195	0.994677	0.990685	0.987303	0.010451	0.003665	0.974767	0.999733

Fig. 1 - Brief description and analysis of the data

Tools & algorithms

Libraries & tools

Programming language: Python 3.9.4

Programming environment: Jupyter Lab

- matplotlib 3.4.1-2
- numpy 1.20.2-1
- pandas 1.2.3-1
- scikit-learn 0.24.1-1
- scipy 1.6.3-1
- seaborn 0.11.1-1
- imbalanced-learn 0.7.0-1

Classifiers used

1. Decision trees
2. K-nearest neighbors
3. Support vector
4. Naive bayes
5. Random forest

All work was can be found in the submitted notebook

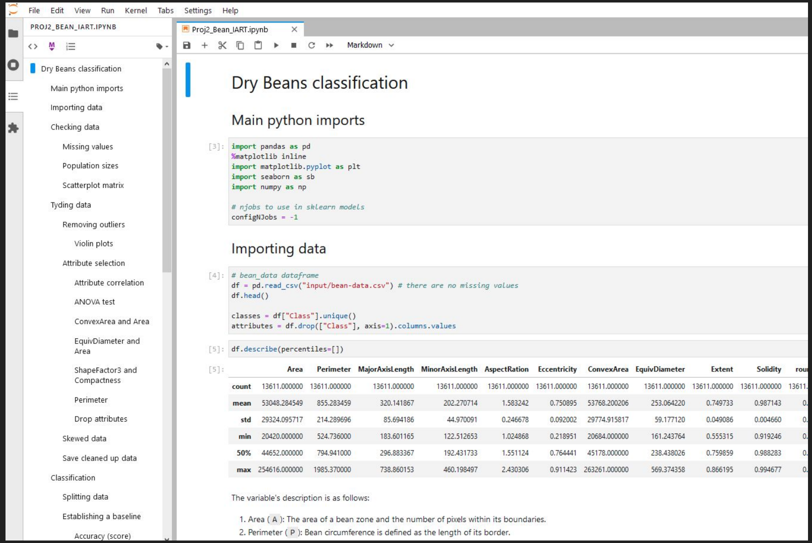

Figure 2 - Snip of the jupyter notebook.

Data Analysis

Missing Values

```
df.isnull().sum()

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
Class              0
dtype: int64
```

Fig. 3 - Number of missing values for each attribute

Data Imbalance

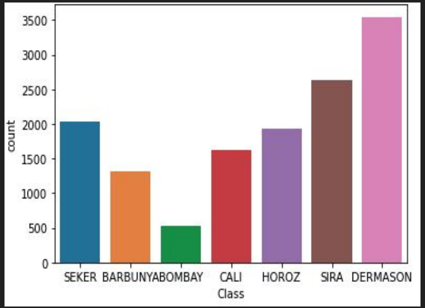

Fig. 4 - Imbalance of bean classes in the data.

Outliers in data

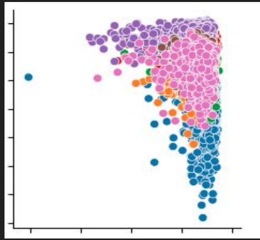

Fig. 5 - Eccentricity by Solidity

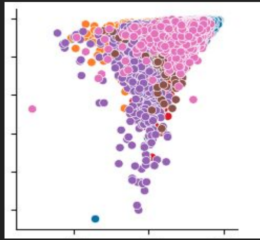

Fig. 6 - Shapefactor4 by Roundness

Feature Correlation

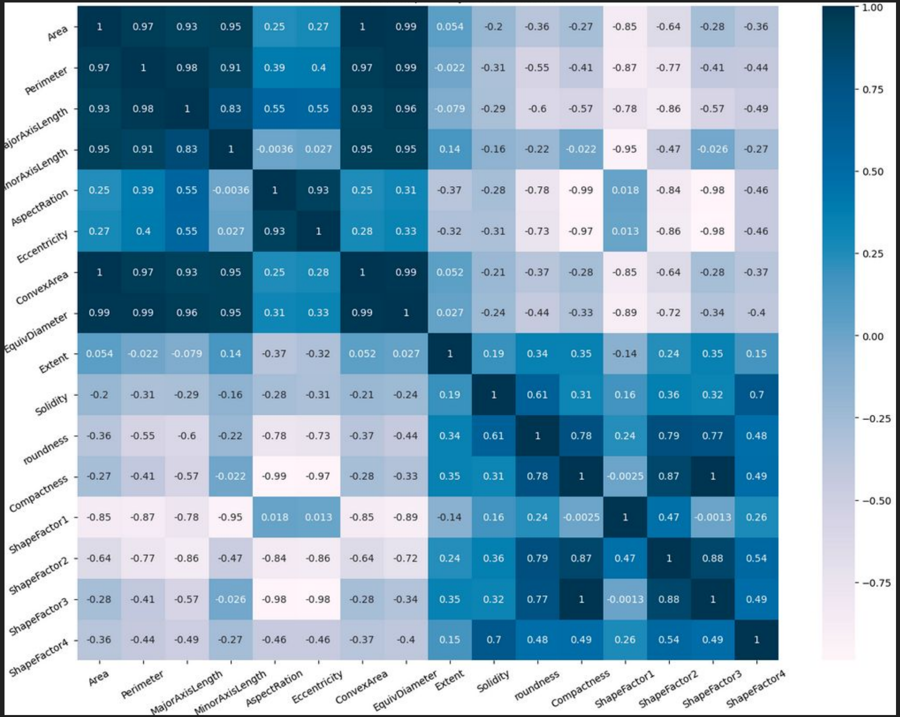

Fig. 7 - Correlation heatmap of all attributes

Data preprocessing

	Features	Score
0	Area	31363.979943
6	ConvexArea	31346.833288
7	EquivDiameter	27432.577030
1	Perimeter	26634.975526
2	MajorAxisLength	23987.837768
3	MinorAxisLength	23586.590304
13	ShapeFactor2	14241.057150
12	ShapeFactor1	12510.092268
4	AspectRation	11846.422220
11	Compactness	11662.234353
14	ShapeFactor3	11374.795676
5	Eccentricity	9596.592513
10	roundness	6941.210935
15	ShapeFactor4	1275.522452
9	Solidity	661.225771
8	Extent	419.975510

Fig. 8 - ANOVA Test of all features

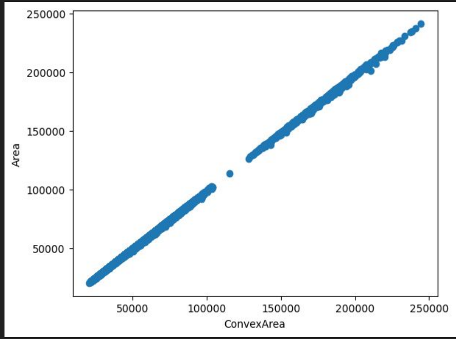

Fig. 9 - Linear correlation between Area and ConvexArea.

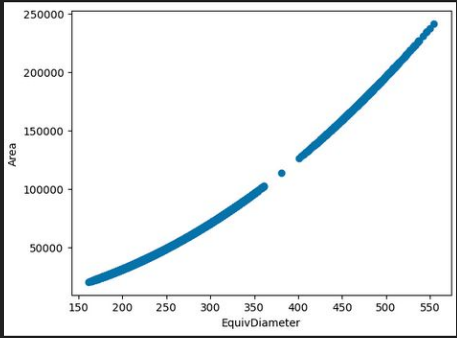

Fig. 10 - Correlation between Area and EquivDiameter.

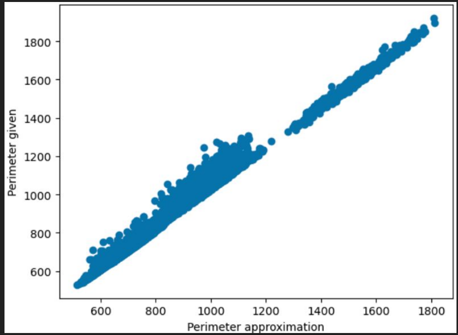

Fig. 11 - Comparison between the perimeter and our approximation.

$$p = 2\pi\sqrt{\frac{a^2+b^2}{2}}$$

$$p = \pi(3(a+b) - \sqrt{(3a+b)(a+3b)})$$

$$h = \frac{(a-b)^2}{(a+b)^2}$$

$$p = \pi(a+b)(1 + \frac{3h}{10+\sqrt{4-3h}})$$

Fig. 12 - Formulas of an ellipsis.

$$area = \pi * (\frac{diameter}{2})^2$$

Fig. 14 - Correlation formula between area and diameter.

$$p = \pi(a+b)$$

Fig. 13 - Correlation formula between perimeter(p), MajorAxisLengt(a)h and MinorAxisLength(b) (assuming h=0).

Data preprocessing

```python
perimeterRatio = approxPerimeter / df["Perimeter"]
sb.displot(perimeterRatio);
```

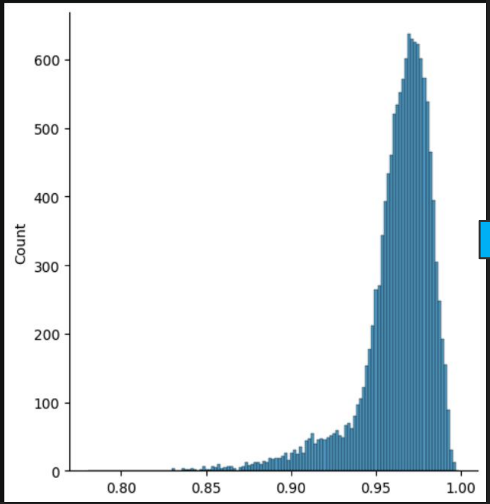

Fig. 14 - Perimeter Ratio before normalization (Heavily skewed graph).

```python
df["perimeterRatio"] = np.log(perimeterRatio)
sb.displot(df["perimeterRatio"]);
```

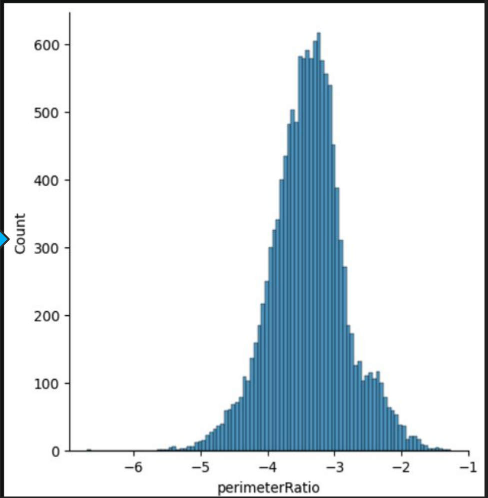

Fig. 15 - Perimeter Ratio after normalization.

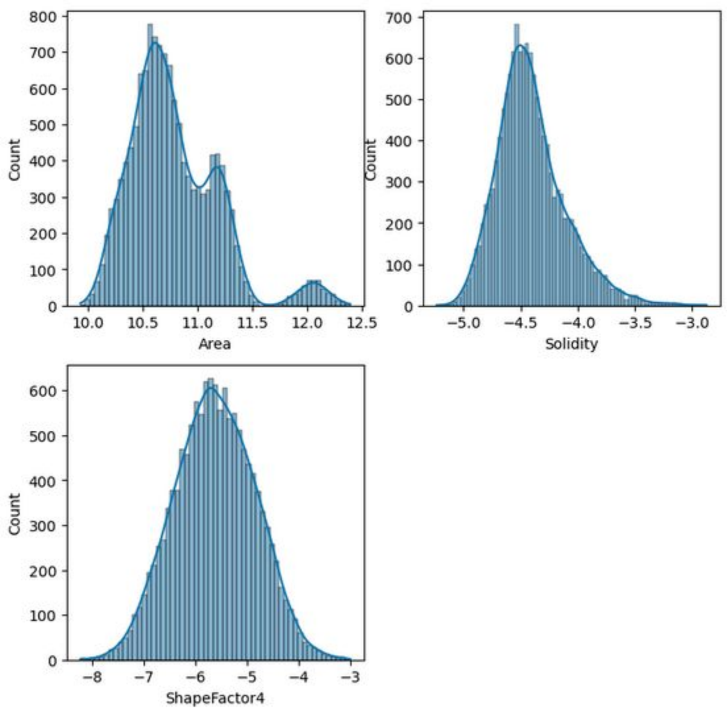

Fig. 16 - Area, Solidity and ShapeFactor after normalization (to reduce skewness).

Classification

Data Splitting

```
from sklearn.model_selection import train_test_split

test_size = 0.30

(X_train, X_test, y_train, y_test) = train_test_split(X, y, random_state=0)
```

Fig. 21 - How we are splitting our data.

Parameter tuning (done for all classifiers)

```
from sklearn.tree import DecisionTreeClassifier

dt_param_grid = {"criterion": ["gini", "entropy"],
                 "splitter": ["best", "random"],
                 "max_depth": [8, 9, 10],
                 "max_features": np.arange(5, len(attributes) + 1, 1),
                 "class_weight": [None, "balanced"],
                 "random_state": [0]}

dt_classifier = grid_search(DecisionTreeClassifier(), dt_param_grid)
```

Fig. 22 - Example of parameter tuning for the DecisonTree classifier using Grid Search.

F1-Score per classifier

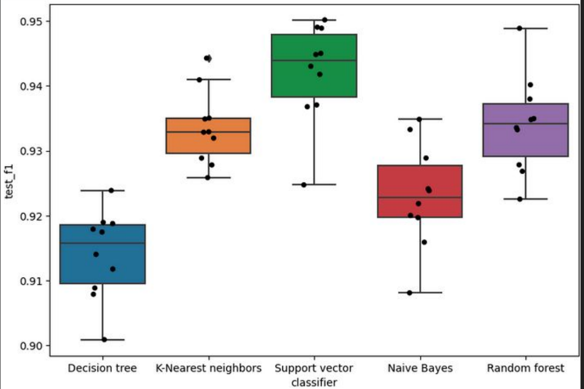

Fig. 17 - F1-Score of each classifier.

Time per classifier

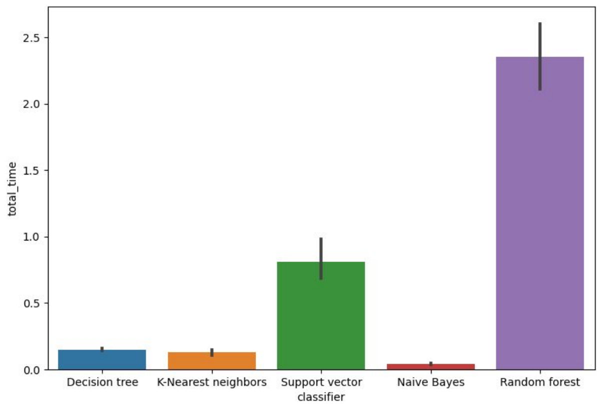

Fig. 18 - Time that each classifier takes to fit and score once.

Oversampling

Since our data is unbalanced we applied the SMOTE oversampling technique

```
from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline, make_pipeline
from imblearn.over_sampling import SMOTE

def grid_search_oversample(clf, parameter_grid, param_prefix):
    imba_pipeline = make_pipeline(SMOTE(random_state=0, n_jobs=configNJobs), clf)
    prefixed_param_grid = {}
    for key in parameter_grid:
        prefixed_param_grid[param_prefix + key] = parameter_grid[key]
    return grid_search(imba_pipeline, prefixed_param_grid)
```

Fig. 19 - Pipeline including oversampling using SMOTE

Support vector classifier

```
sv_classifier_os = grid_search_oversample(SVC(), sv_param_grid, "svc__")
```

```
Best score: 0.9404039309850933
Best parameters: {'svc__C': 1, 'svc__gamma': 'auto', 'svc__kernel': 'rbf', 'svc__random_state': 0}
```

Fig. 20 - Oversampling results for support vector (done for all classifiers)

	accuracy	precision	recall	f1	oversampled	classifier
0	0.942926	0.943116	0.942926	0.942971	no	Support vector
1	0.941724	0.942167	0.941724	0.941825	yes	Support vector
0	0.937218	0.937452	0.937218	0.937245	no	Random forest
0	0.936618	0.937121	0.936618	0.936684	no	K-Nearest neighbors
1	0.935416	0.935763	0.935416	0.935470	yes	Random forest
1	0.933914	0.934541	0.933914	0.934031	yes	K-Nearest neighbors
1	0.926404	0.927340	0.926404	0.926623	yes	Decision tree
0	0.918294	0.919726	0.918294	0.918422	no	Naive Bayes
1	0.917693	0.919496	0.917693	0.917866	yes	Naive Bayes
0	0.916491	0.917387	0.916491	0.916781	no	Decision tree

Fig. 21 - Oversampled results compared to non-oversampled results

Results and Conclusions

- Oversampling produced negligible results (in most cases).
- Support Vector had the best results.
- F1-Score differences between the multiple classifiers isn't very significant.
- All classifiers produced similar results: **F1-Score between 91%-95%**.
- Although it performs the best, the Support vector classifier is quite slow (the **2nd slowest**).
- **The Random forest** and the **K-Nearest neighbors** classifiers perform similarly to the **Support vector** classifier.
- If time is a relevant constraint, **K-Nearest neighbors** should be considered, because it runs several times faster.

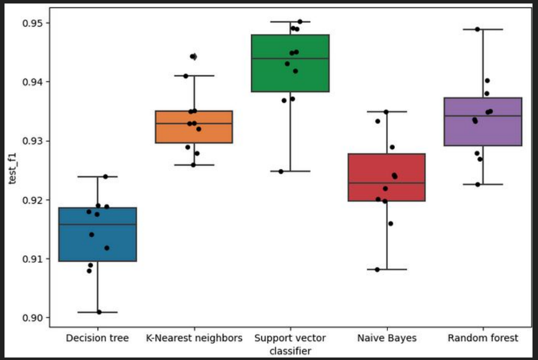

Fig. 22 - F1-Score of each classifier

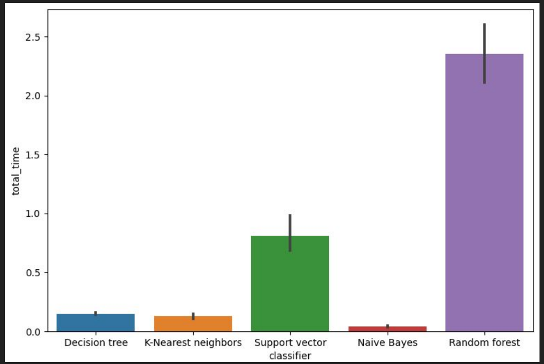

Fig. 23 - Time that each classifier takes to fit and score once.

Related work and bibliographic search

- Previous solutions of the same problem
  - https://github.com/NaitikJ/DryBean--Dataset
  - https://github.com/HimankSehgal/DSGRecruitmentTask_DryBeanDataset
- Data Analysis and Machine Learning Projects
  - https://github.com/rhiever/Data-Analysis-and-Machine-Learning-Projects
- Performance metrics to classification problems
  - https://www.kaggle.com/usengecoder/performance-metrics-for-classification-problems
- Feature Selection Techniques
  - https://pierpaolo28.github.io/blog/blog27/
  - https://www.kaggle.com/rxsraghavagrawal/feature-selection-techniques
  - https://www.kaggle.com/prashant111/comprehensive-guide-on-feature-selection
- Select k best: feature selection example in python
  - https://www.datatechnotes.com/2021/02/seleckbest-feature-selection-example-in-python.html
- Remove outliers in python
  - https://www.statology.org/remove-outliers-python
- Oversampling
  - https://kiwidamien.github.io/how-to-do-cross-validation-when-upsampling-data.html