OSDA Big Homework

Ivan Efimenkov

December 2024

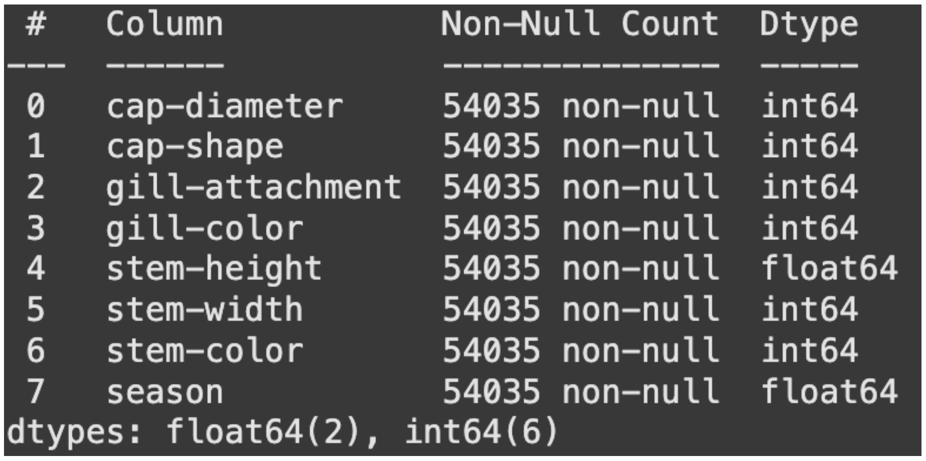
1 Link

Github code

`https://github.com/codered-epidemy/OSDA`

2 Data description

The dataset is taken from Kaggle (link). It was preprocessed by Prisha Sawhney. Original dataset is taken from UCL Library (link).
This data shows different features of mushrooms. Using this features we need to predict whether it is edible or not.
In original dataset from UCL there is 20 features. Kaggle dataset consists of 54035 rows with 8 features (all numerical). Target feature:
1 - mushroom is poisonous
0 - mushroom is edible

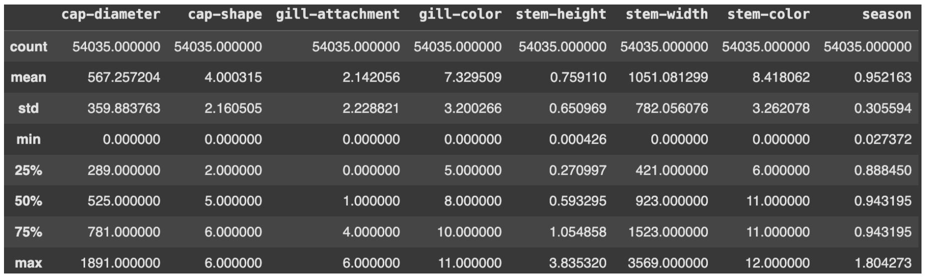


```
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   cap-diameter     54035 non-null  int64  
 1   cap-shape        54035 non-null  int64  
 2   gill-attachment  54035 non-null  int64  
 3   gill-color       54035 non-null  int64  
 4   stem-height      54035 non-null  float64
 5   stem-width       54035 non-null  int64  
 6   stem-color       54035 non-null  int64  
 7   season           54035 non-null  float64
dtypes: float64(2), int64(6)
```

Features and their types

3 EDA

I checked NA values, get info about data (using .info()) and also used describe method.

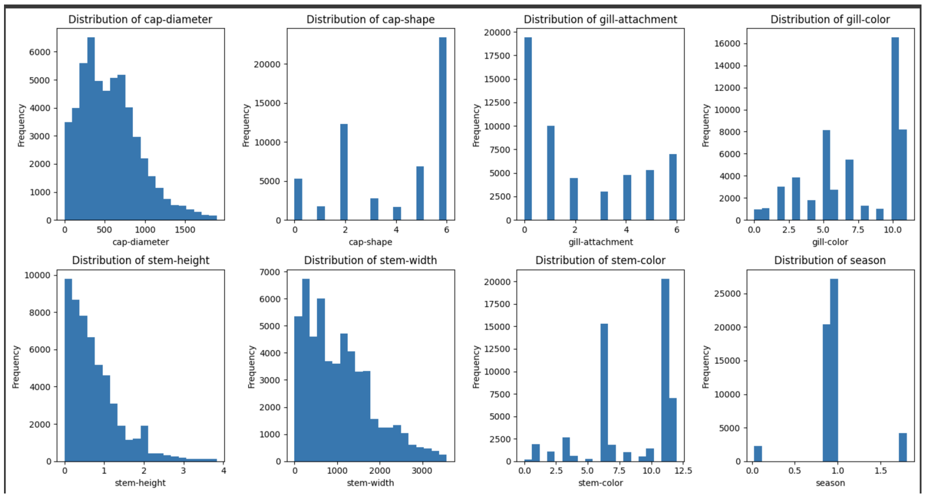


	cap-diameter	cap-shape	gill-attachment	gill-color	stem-height	stem-width	stem-color	season
count	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000	54035.000000
mean	567.257204	4.000315	2.142056	7.329509	0.759110	1051.081299	8.418062	0.952163
std	359.883763	2.160505	2.228821	3.200266	0.650969	782.056076	3.262078	0.305594
min	0.000000	0.000000	0.000000	0.000000	0.000426	0.000000	0.000000	0.027372
25%	289.000000	2.000000	0.000000	5.000000	0.270997	421.000000	6.000000	0.888450
50%	525.000000	5.000000	1.000000	8.000000	0.593295	923.000000	11.000000	0.943195
75%	781.000000	6.000000	4.000000	10.000000	1.054858	1523.000000	11.000000	0.943195
max	1891.000000	6.000000	6.000000	11.000000	3.835320	3569.000000	12.000000	1.804273

Description of features

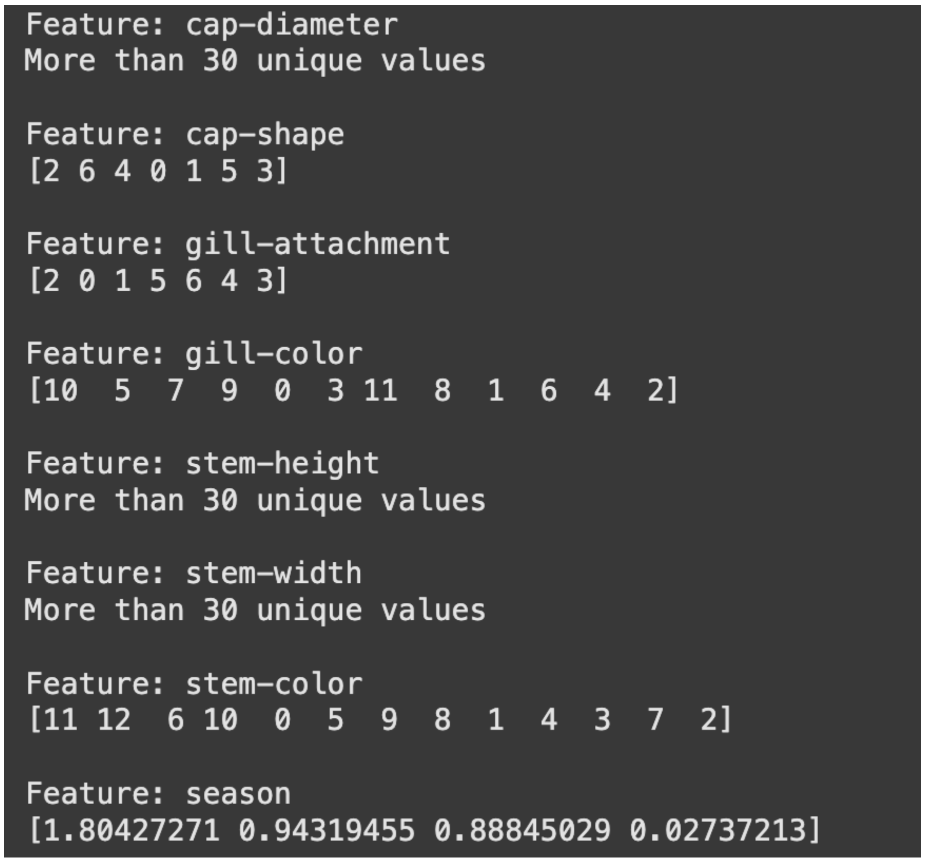
4 Visual DA

Here I used code from Alan's gitlab and get this distributions



Description of features

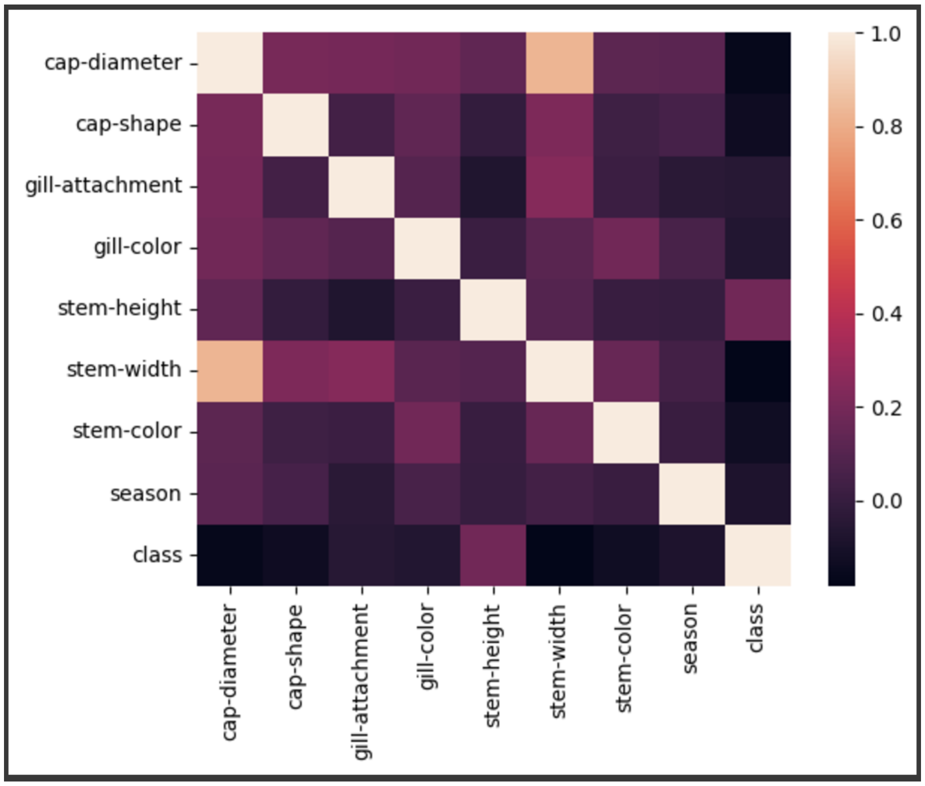
From this distributions I decided to use nominal scaling for cap-shape, gill-attachment, gill-color, stem-color, season. I also counted unique values in different features to check.



```
Feature: cap-diameter
More than 30 unique values

Feature: cap-shape
[2 6 4 0 1 5 3]

Feature: gill-attachment
[2 0 1 5 6 4 3]

Feature: gill-color
[10  5  7  9  0  3 11  8  1  6  4  2]

Feature: stem-height
More than 30 unique values

Feature: stem-width
More than 30 unique values

Feature: stem-color
[11 12  6 10  0  5  9  8  1  4  3  7  2]

Feature: season
[1.80427271 0.94319455 0.88845029 0.02737213]
```

Unique values in different features

For other features I decided to make 5 bins, divide values into them and use inter-ordinal scaling.
After that I checked correlation matrix:



Correlation matrix

Here we can see that most features do not correlate significantly with the target variable. Nevertheless, I would consider all the features.

5 Binarization

As I already mentioned, next step is binarization. For this, I applied inter-ordinal scaling, where we apply rule ”greater than or equal and less than or equal” to some threshold. I also used nominal scaling, because the amount of unique values was not so huge, so it would be appropriate for our methods to count it (taking into account time they spend on evaluations). For inter-ordinal, I took **min** and **max** values of features, divided into 5 bins and applied these bin rules to each object.
For nominal scaling I applied One Hot Encoding.
After binarization I got 81 columns (features).

6 Metrics evaluation

For evaluation of the performance of the models I used:

- **Accuracy** (measures overall correctness of the model):

$$\frac{TP+TN}{TP+FP+FN+TN}$$

- **Precision** (measures how many of the predicted positive cases are actually positive):

$$\frac{TP}{TP+FP}$$

- **Recall** (measures the ability of the model to identify all relevant positive cases):

$$\frac{TP}{TP+FN}$$

- **F1-score** (balances precision and recall, providing a single metric when both are important.):

$$2 * \frac{Precision * Recall}{Precision + Recall}$$

- **True positive** (indicates how many actual positives were correctly identified)
- **False positive** (indicates errors where the model predicted positive but the actual label was negative)
- **False negative** (indicates errors where the model missed positive cases)
- **True negative** (indicates how well the model identifies actual negatives)
- **True Negative Rate** (measures the ability of the model to correctly identify negatives):

$$\frac{TN}{TN+FP}$$

- **False positive rate** (measures how often the model falsely identifies negatives as positives):

$$\frac{FP}{TN+FP}$$

- **False discovery rate** (measures the trustworthiness of positive predictions):

$$\frac{FP}{TP+FP}$$

I also measured time of fitting and predicting.

7 Data split

I decided to split data into to equal parts and for this apply all models (Data split section in code). I suggest such variant because original data is not strongly imbalanced.

8 LazyFCA

LazyFCA is a hybrid machine learning method that combines the principles of Formal Concept Analysis with lazy learning techniques. It is designed for classification tasks where predictions are made based on the relationships between a test sample and subsets of training data, without building a full model beforehand.
LazyFCA avoids creating an exhaustive concept lattice (a hallmark of traditional FCA) and instead focuses on dynamically finding relevant patterns during the classification of each test sample. This approach leverages concepts (defined by shared attributes in the data) to classify samples in a transparent and interpretable way.
I modified Alan's code so it works faster now. The intersections (&) between the sample and X_train subsets are computed in bulk, improving speed. Also I added verbose parameter to class.
Results:

Metric	**Value**
Accuracy	0.497436
Precision	0.497436
Recall	1.00000
F1	0.664384
TP	97
FP	98
FN	0
TN	0
TNR	0
FPR	1.000000
FDR	0.502564
Training time	0.000005
Prediciton time	0.490130

9 ModifiedLazyFCA

ModifiedLazyFCA incorporates weighted voting. This modification assigns weights to the classifiers based on their similarity to the sample being classified.

Definition of weight voting
Each classifier (positive or negative) contributes to the decision based on how

many features it shares with the sample. A higher number of overlapping features results in a higher weight, meaning that classifier is more relevant to the current sample.

How is weight computed?
For each classifier (positive or negative): Count the number of matching features with the test sample. Divide this count by the total number of features in the sample. This gives a normalized weight between 0 and 1, indicating the relevance of the classifier.
Results:

Metric	**Value**
Accuracy	0.656410
Precision	0.727273
Recall	0.494845
F1	0.588957
TP	48
FP	18
FN	49
TN	80
TNR	0.816327
FPR	0.183673
FDR	0.272727
Training time	0.000626
Prediciton time	0.437869

Difference between LazyFCA realizations
ModifiedLazyClassifierFCA achieves better overall performance by balancing precision and recall, leading to significantly improved accuracy and interpretability.

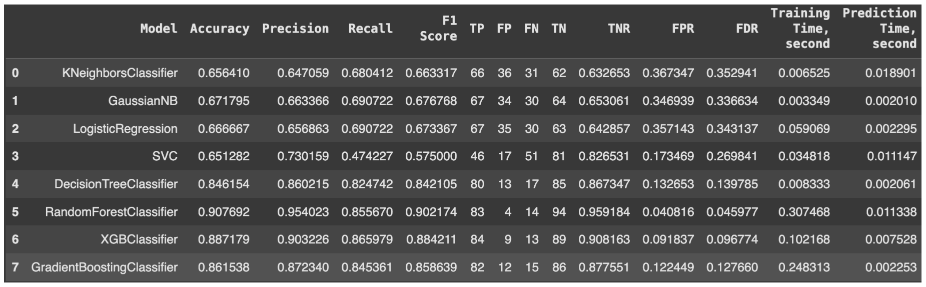
10 Comparison with other models

List of applied models:

- KNN
- Naive Bayes
- Logistic Regression
- SVM
- Decision Tree
- Random Forest
- XGBoost

- GBC

First I applied models to unbinarized data:

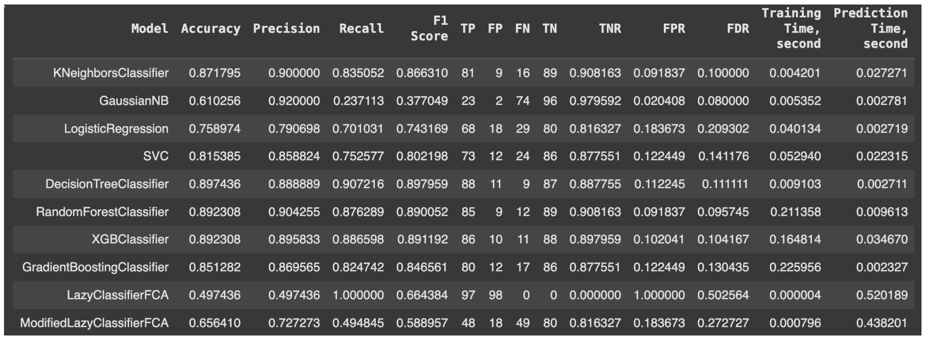


	Model	Accuracy	Precision	Recall	F1 Score	TP	FP	FN	TN	TNR	FPR	FDR	Training Time, second	Prediction Time, second
0	KNeighborsClassifier	0.656410	0.647059	0.680412	0.663317	66	36	31	62	0.632653	0.367347	0.352941	0.006525	0.018901
1	GaussianNB	0.671795	0.663366	0.690722	0.676768	67	34	30	64	0.653061	0.346939	0.336634	0.003349	0.002010
2	LogisticRegression	0.666667	0.656863	0.690722	0.673367	67	35	30	63	0.642857	0.357143	0.343137	0.059069	0.002295
3	SVC	0.651282	0.730159	0.474227	0.575000	46	17	51	81	0.826531	0.173469	0.269841	0.034818	0.011147
4	DecisionTreeClassifier	0.846154	0.860215	0.824742	0.842105	80	13	17	85	0.867347	0.132653	0.139785	0.008333	0.002061
5	RandomForestClassifier	0.907692	0.954023	0.855670	0.902174	83	4	14	94	0.959184	0.040816	0.045977	0.307468	0.011338
6	XGBClassifier	0.887179	0.903226	0.865979	0.884211	84	9	13	89	0.908163	0.091837	0.096774	0.102168	0.007528
7	GradientBoostingClassifier	0.861538	0.872340	0.845361	0.858639	82	12	15	86	0.877551	0.122449	0.127660	0.248313	0.002253

Metrics evaluation for **unbinarized** data

Here we can see that Random Forest is the best model.

Now let's apply same models to binarized data (including LazyFCA):



Model	Accuracy	Precision	Recall	F1 Score	TP	FP	FN	TN	TNR	FPR	FDR	Training Time, second	Prediction Time, second
KNeighborsClassifier	0.871795	0.900000	0.835052	0.866310	81	9	16	89	0.908163	0.091837	0.100000	0.004201	0.027271
GaussianNB	0.610256	0.920000	0.237113	0.377049	23	2	74	96	0.979592	0.020408	0.080000	0.005352	0.002781
LogisticRegression	0.758974	0.790698	0.701031	0.743169	68	18	29	80	0.816327	0.183673	0.209302	0.040134	0.002719
SVC	0.815385	0.858824	0.752577	0.802198	73	12	24	86	0.877551	0.122449	0.141176	0.052940	0.022315
DecisionTreeClassifier	0.897436	0.888889	0.907216	0.897959	88	11	9	87	0.887755	0.112245	0.111111	0.009103	0.002711
RandomForestClassifier	0.892308	0.904255	0.876289	0.890052	85	9	12	89	0.908163	0.091837	0.095745	0.211358	0.009613
XGBClassifier	0.892308	0.895833	0.886598	0.891192	86	10	11	88	0.897959	0.102041	0.104167	0.164814	0.034670
GradientBoostingClassifier	0.851282	0.869565	0.824742	0.846561	80	12	17	86	0.877551	0.122449	0.130435	0.225956	0.002327
LazyClassifierFCA	0.497436	0.497436	1.000000	0.664384	97	98	0	0	0.000000	1.000000	0.502564	0.000004	0.520189
ModifiedLazyClassifierFCA	0.656410	0.727273	0.494845	0.588957	48	18	49	80	0.816327	0.183673	0.272727	0.000796	0.438201

Metrics evaluation for **binarized** data

11 Conclusion

If we take a glance at the pictures above, we can find that state-of-the-art methods overcome the LazyFCA classificator (even after implementing modification). I suppose some other modification could improve LazyFCA efficiency and performance significantly. For this dataset, the best model is RandomForestClassifier (even without hyperparameter tuning).