Urban Artem Sergeevich

Lazy FCA Project

Github repository link: https://github.com/dlkeyxq/Lazy-FCA

Original dataset link (taken from Kaggle): https://www.kaggle.com/datasets/mssmartypants/rice-type-classification

Dataset Description

This is a set of data created for rice classification. There are 2 classes: Jasmine - 1, Gonen - 0.

There are total 18185 rows and 12 columns in dataset, they are following:

- id
- Area
- MajorAxisLength
- MinorAxisLength
- Eccentricity
- ConvexArea
- EquivDiameter
- Extent
- Perimeter
- Roundness
- AspectRation
- Class

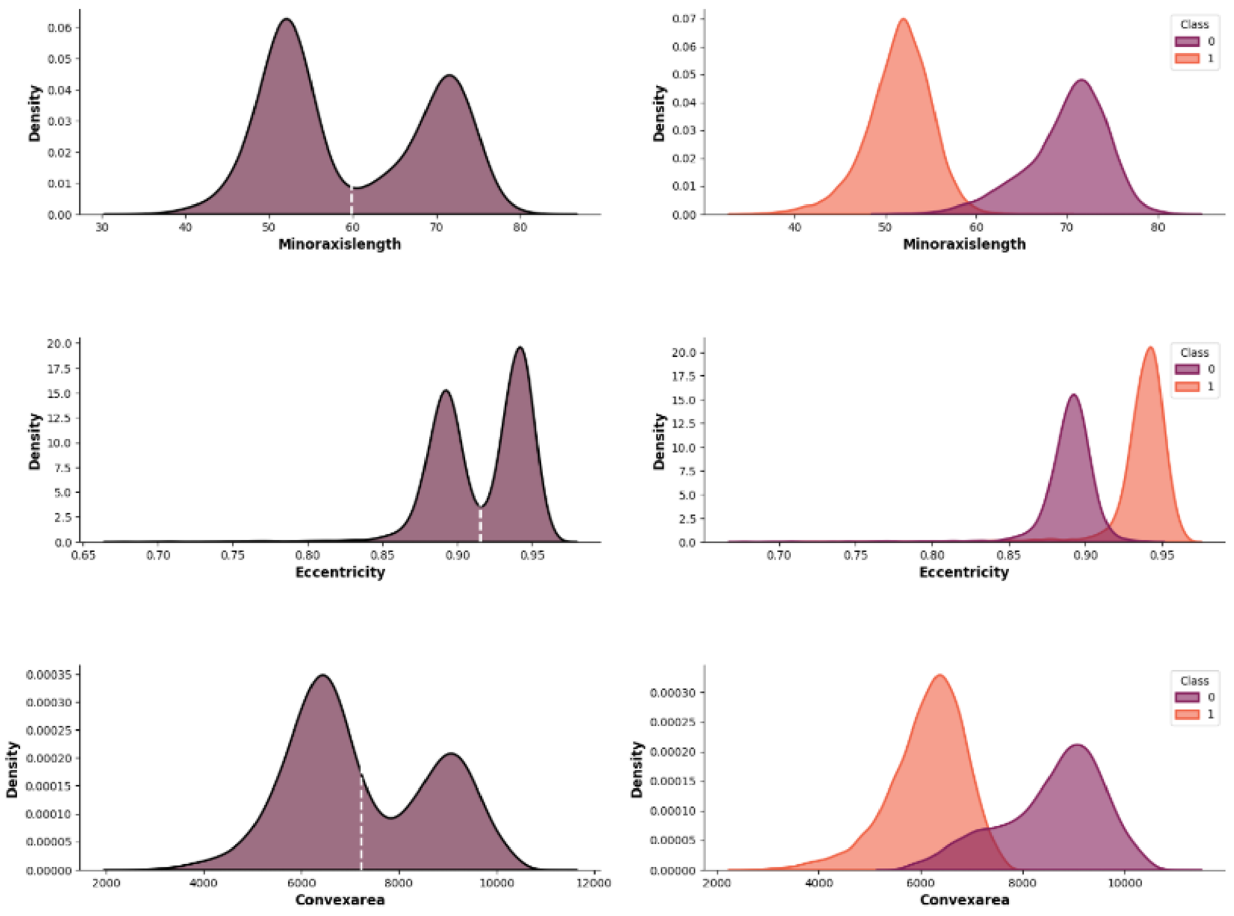
All the columns (except id and Class) contains numerical information about shape features and other mathematical parameters of grains of ice.

What steps were done

1. EDA

On this step I've analyzed the dataset:

- found null values (there were no null values),
- dropped id column (this does not contain any useful information for learning since every row has different id),
- looked into distribution of all features,
- examined correlation between different features and target: it was found that there is high positive correlation between Eccentricity, AspectRation and Class; high negative correlation between Area, MinotAxisLength, ConvexArea, EquivDiameter, Roundness and Class.



2. Data Scaling (binarization)

On this step I've chosen inter-ordinal scaling for every feature (since every feature is numerical) and the link between the target and features was not clear.

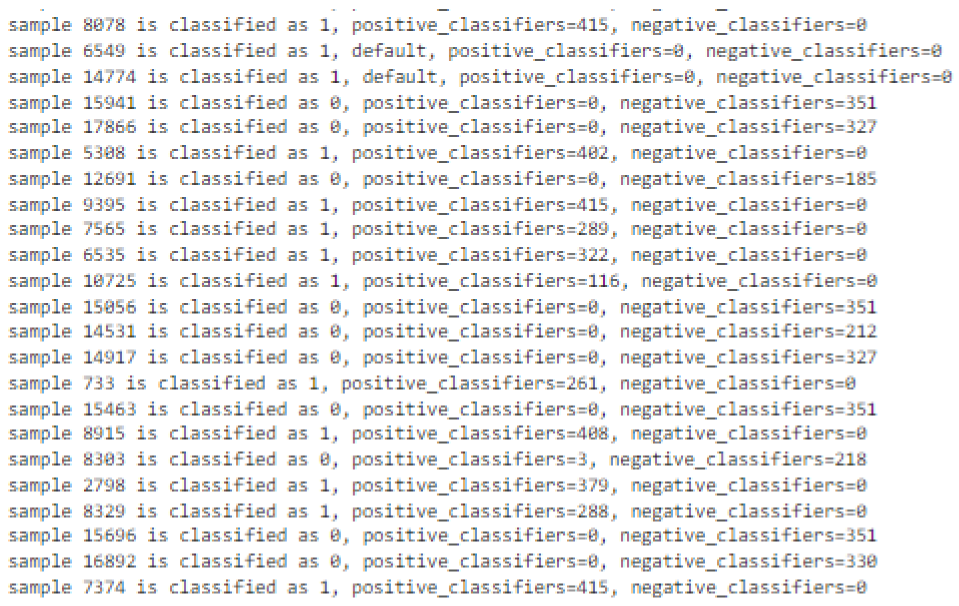
For each feature I checked distribution plots, min and max values and determined in-between values for thresholds

After binarization was complete, there were 106 features in binarized dataset.

3. Lazy FCA application

On this step I've selected only 6% of data that was in original dataset because Lazy FCA is quite slow algorithm and with only 6% of data it took about 15 minutes for model to make predictions, with 100% of data it will be more than a day for model to make predictions.

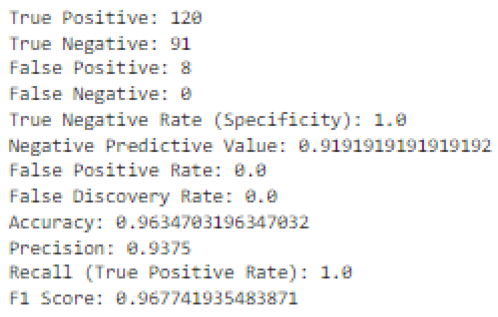
As a Lazy FCA algorithm I've selected original lazy FCA baseline.



```
sample 8078 is classified as 1, positive_classifiers=415, negative_classifiers=0
sample 6549 is classified as 1, default, positive_classifiers=0, negative_classifiers=0
sample 14774 is classified as 1, default, positive_classifiers=0, negative_classifiers=0
sample 15941 is classified as 0, positive_classifiers=0, negative_classifiers=351
sample 17866 is classified as 0, positive_classifiers=0, negative_classifiers=327
sample 5308 is classified as 1, positive_classifiers=402, negative_classifiers=0
sample 12691 is classified as 0, positive_classifiers=0, negative_classifiers=185
sample 9395 is classified as 1, positive_classifiers=415, negative_classifiers=0
sample 7565 is classified as 1, positive_classifiers=289, negative_classifiers=0
sample 6535 is classified as 1, positive_classifiers=322, negative_classifiers=0
sample 10725 is classified as 1, positive_classifiers=116, negative_classifiers=0
sample 15056 is classified as 0, positive_classifiers=0, negative_classifiers=351
sample 14531 is classified as 0, positive_classifiers=0, negative_classifiers=212
sample 14917 is classified as 0, positive_classifiers=0, negative_classifiers=327
sample 733 is classified as 1, positive_classifiers=261, negative_classifiers=0
sample 15463 is classified as 0, positive_classifiers=0, negative_classifiers=351
sample 8915 is classified as 1, positive_classifiers=408, negative_classifiers=0
sample 8303 is classified as 0, positive_classifiers=3, negative_classifiers=218
sample 2798 is classified as 1, positive_classifiers=379, negative_classifiers=0
sample 8329 is classified as 1, positive_classifiers=288, negative_classifiers=0
sample 15696 is classified as 0, positive_classifiers=0, negative_classifiers=351
sample 16892 is classified as 0, positive_classifiers=0, negative_classifiers=330
sample 7374 is classified as 1, positive_classifiers=415, negative_classifiers=0
```

As we can see, major number of samples has great amount of positive (negative) classifiers and 0 negarive (positive) classifiers. That means that model making its predictions with high degree of certainty.

Then I've evaluated the model using next quality metrics:

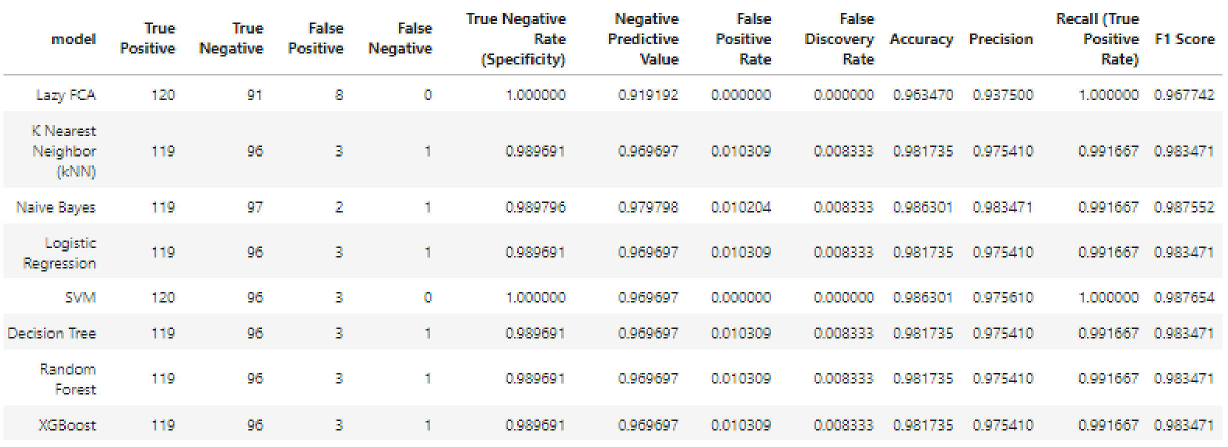


```
True Positive: 120
True Negative: 91
False Positive: 8
False Negative: 0
True Negative Rate (Specificity): 1.0
Negative Predictive Value: 0.9191919191919192
False Positive Rate: 0.0
False Discovery Rate: 0.0
Accuracy: 0.9634703196347032
Precision: 0.9375
Recall (True Positive Rate): 1.0
F1 Score: 0.967741935483871
```

Overall, model performance is quite good, because all the metrics are > 0.9 (except FP, FN, FPR, FDR, but that's mainly because FP and FN = 0).

4. Comparing with other algorithms

On this step I've fitted different ML models (KNN, Naive Bayes, Logistic Regression, SVM, Decision Tree, Random Forest, XGBoost) to compare performance with Lazy FCA model.



model	True Positive	True Negative	False Positive	False Negative	True Negative Rate (Specificity)	Negative Predictive Value	False Positive Rate	False Discovery Rate	Accuracy	Precision	Recall (True Positive Rate)	F1 Score
Lazy FCA	120	91	8	0	1.000000	0.919192	0.000000	0.000000	0.963470	0.937500	1.000000	0.967742
K Nearest Neighbor (kNN)	119	96	3	1	0.989691	0.969697	0.010309	0.008333	0.981735	0.975410	0.991667	0.983471
Naive Bayes	119	97	2	1	0.989796	0.979798	0.010204	0.008333	0.986301	0.983471	0.991667	0.987552
Logistic Regression	119	96	3	1	0.989691	0.969697	0.010309	0.008333	0.981735	0.975410	0.991667	0.983471
SVM	120	96	3	0	1.000000	0.969697	0.000000	0.000000	0.986301	0.975610	1.000000	0.987654
Decision Tree	119	96	3	1	0.989691	0.969697	0.010309	0.008333	0.981735	0.975410	0.991667	0.983471
Random Forest	119	96	3	1	0.989691	0.969697	0.010309	0.008333	0.981735	0.975410	0.991667	0.983471
XGBoost	119	96	3	1	0.989691	0.969697	0.010309	0.008333	0.981735	0.975410	0.991667	0.983471

Unfortunately, Lazy FCA algorithm performed a little bit worse than other models (it has lower accuracy, NPV, Precision, f1_score) taking into account major metrics, but it had the best TNR, FPR, FDR and Recall.

Nevertheless, the difference in metrics is not that high and that means that Lazy FCA is good algorithm for binary classification.

However, the biggest minus of this algorithm is it’s complexity, because for each test row we are checking the dependencies with all train data and it has at least power-law complexity and that is why it is performing very slow with big amount of data.