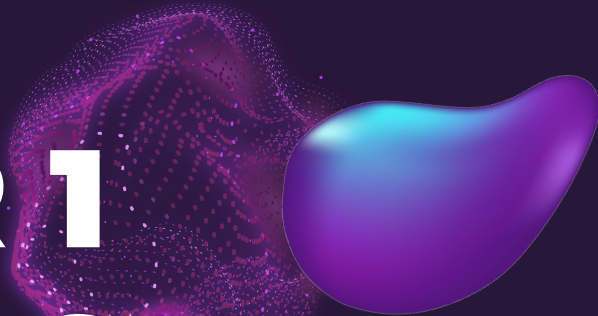


CHALLENGER 1^x CLASSIFICATION

(IML)

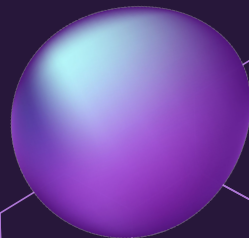
ZUHA AQIB 26106

IBA, Fall 2024



x

x



CONTENTS OF THIS TEMPLATE

x

x

You can delete this slide when you're done editing the presentation

<u>FONTS</u>	To view this template correctly in PowerPoint, download and install the fonts we used
<u>USED AND ALTERNATIVE RESOURCES</u>	An assortment of graphic resources that are suitable for use in this presentation
<u>THANKS SLIDE</u>	You must keep it so that proper credits for our design are given
<u>COLORS</u>	All the colors used in this presentation
<u>ICONS AND INFOGRAPHIC RESOURCES</u>	These can be used in the template, and their size and color can be edited
EDITABLE PRESENTATION THEME	You can edit the master slides easily. For more info, click <u>here</u>

x

For more info:

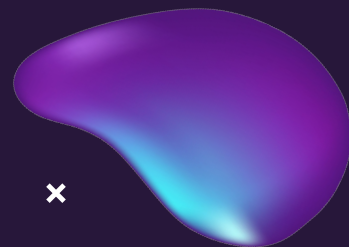
SLIDESGO | BLOG | FAQs

You can visit our sister projects:

FREEPIK | FLATICON | STORYSET | WEPIK | VIDEVO



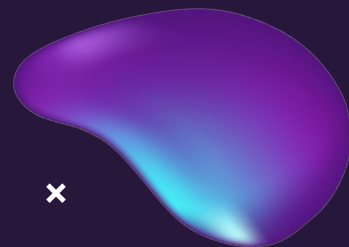
TABLE OF CONTENTS



01	ABOUT THE CHALLENGE	0.96407
02	DECISION TREES	0.91215
03	GAUSSIAN NAIVE BAYES	You can describe the topic of the section here
04	CATEGORICAL NAIVE BAYES	You can describe the topic of the section here
05	K-NEAREST NEIGHBOURS	You can describe the topic of the section here



TABLE OF CONTENTS



06 **RANDOM FOREST**

You can describe the
topic of the section here

07 **GRADIENT BOOSTING**

You can describe the
topic of the section here

08 **ADAPTIVE BOOSTING**

You can describe the
topic of the section here

09 **LIGHT GBM**

You can describe the
topic of the section here

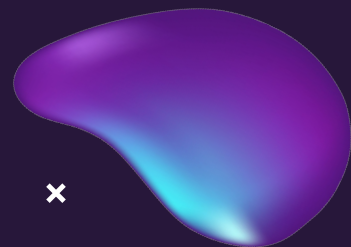
10 **XGBOOST**

You can describe the
topic of the section here



x

TABLE OF CONTENTS



x

11

CAT BOOST

You can describe the
topic of the section here

13

**EXT. RANDOMIZED
TREE**

You can describe the
topic of the section here

14

VOTING

You can describe the
topic of the section here

15

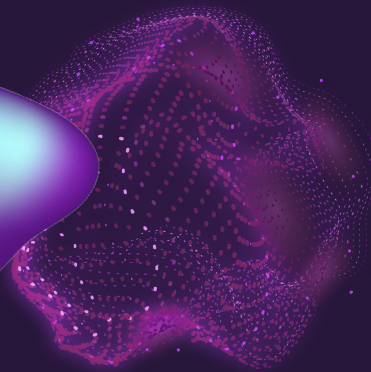
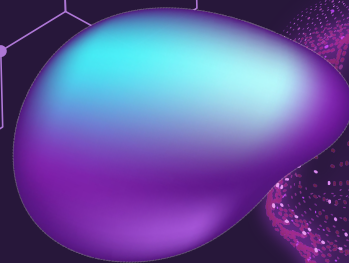
STACKING

You can describe the
topic of the section here

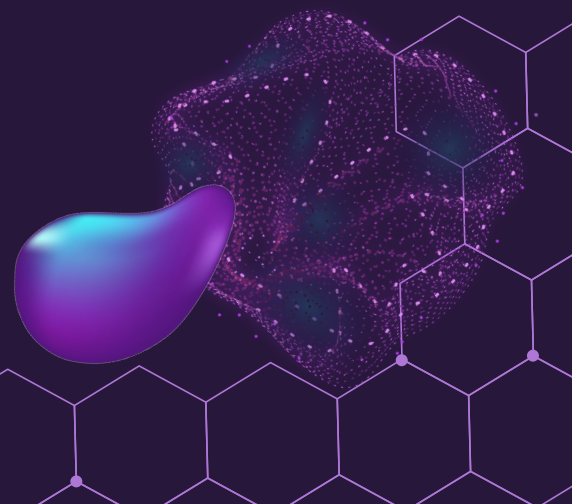
ABOUT THE CHALLENGE



+



x





MY PERFORMANCE

Total Number of Submissions on Kaggle: 210

Highest Accuracy: 0.96407

Position on Leaderboard: 7th

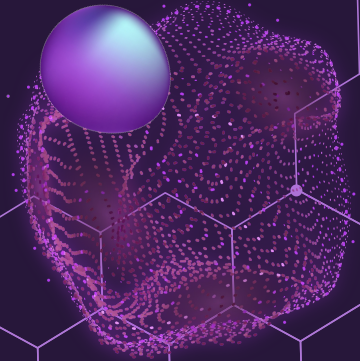
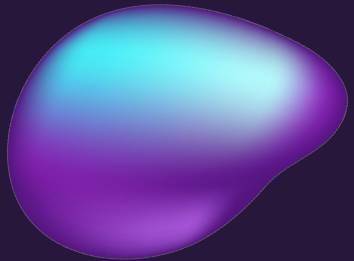
Submitted 10 every day for 21 days



GitHub Repo:



<https://github.com/z-aqib/machine-learning-algorithms.git>



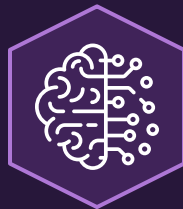
The GitHub Repository contains:

x



Daily Cases

Every entry done with its CSV and notebook in the form of commits



Algo Analysis

Tabular form analysis of each algorithm with rigorous testing of hyperparameters





x

CSVs, Notebooks

CSV and Jupyter Notebook of all 220 submissions in commit form

+

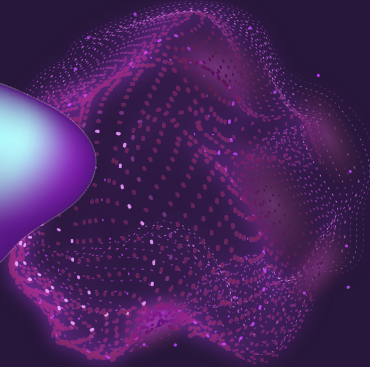
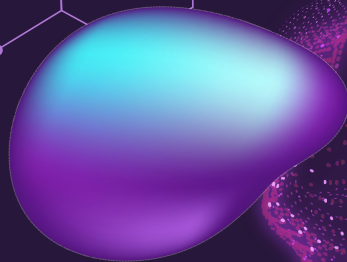


 z-aqib	added cases	8e29d30 · 12 minutes ago	 378 Commits
 .vscode	vscode update - dunno this file	last week	
 challenger1	added cases	12 minutes ago	

The repository will be public on 11th November 2024, at 12:00 AM

0

x

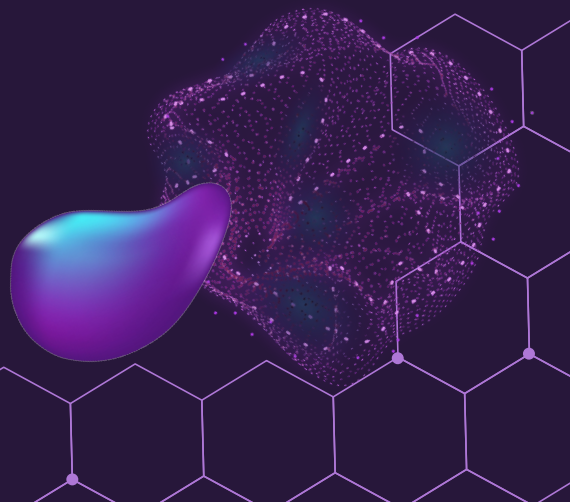


x

2 DECISION TREES



+



total test cases done: 34
started accuracy: 0.72913
highest accuracy achieved: 0.91215
best parameters: (case 159)

- scaler = maxabs
- imputer = knn=7
- feature selection = none
- bagging : 50 estimators
- data splitting = holdout, 70-30 ratio
- criteria: entropy
- maximum depth tree: 5
- minimum samples split: 15
- maximum features used: 60
- minimum samples per leaf: 80

- minmax and maxabs is the best scalers
- normalizer is the worst scaler
- simple and knn=7 perform best, knn=3 performs a bit lesser
- depth of tree is good at 5, 6, while 4, 8 leads to underfit/overfit
- too less features used like 10 and 50 is bad
- too many samples on a leaf like 100 result in underfit. 80 is the breakpoint. smaller then 80 result in overfit
- cross fold performs best at k=10
- entropy is better then gini
- PCA did not perform well, even at best case
- bagging improves the algorithm! did not have enough submissions to analyse the best case bagging, but highest was found!
- algorithm feature importance is okay, it did have higher accuracy but it was because of bagging. maybe the features were too less/more, but overall accuracy decreased

x

Analyzing Scalers and Imputers

Scalers / Imputers	SimpleImputer	KNN = 3	KNN = 5	KNN = 7	Average	New Average
MinMaxScaler	0.89330	0.89142	0.87791	0.88134	0.885605	0.883556667
StandardScaler	0.77620	0.80450	-	-	0.79035	-
MaxAbsScaler	0.88454	0.88155	0.84670	0.88389	0.884215	0.870713333
RobustScaler	0.88517	-	0.87303	-	0.87910	-
Normalizer	0.64969	-	-	0.65308	0.651385	-

the best among all 5 scalers is MinMaxScaler and MaxAbsScaler. the third best is RobustScaler, after that StandardScaler is lower significantly and NormalizerScaler is very very low. Hence we shall be alternating between MinMaxScaler and MaxAbsScaler as they are only different in the third decimal point.

out of KNN and SimpleImputers, we can see that both are good however simple imputer performs better on average. thus we will work with both. in the next 4 cases, let's test knn=3, 5, 7 for MinMaxScaler and MaxAbsScaler to find the best KNN going forward.

x

x

0

x

GAUSSIAN 3 NAIVEBAYES

x

+

total test cases done: 34

total submissions: 18

starting accuracy: 0.83725 (case 31)

highest accuracy achieved: 0.87413 (case 45)

highest parameters:

- imputer = simple
- scaler = minmax
- forward selection with 15 rows

analysis:

- forward selection is best at 15
- knn=7 and simple imputer have no difference in performance with naivebayes
- simple, minmax worked best with NB
- NB performed better with more features then lesser
- row removal does not work in naivebayes
- bagging didnt work so well
- naive bayes does not have algorithm feature importance
- PCA does not perform well

Analyzing Forward Feature Selection

we have run naivebayes multiple times with forward selection, lets analyse its accuracies (while keeping all other parameters like imputer and scaler constant):

case number	feature selector	no. of features	kaggle accuracy
34	forward	5	0.82386
49	forward	13	0.87353
50	forward	14	0.87271
45	forward	15	0.87413
48	forward	17	0.87278
35b	forward	20	0.87148
42	forward	25	0.86875
41a	forward	30	0.86669

we can see that forward=14 might be an outlier, but forward=15 (case = 45) is the best case.

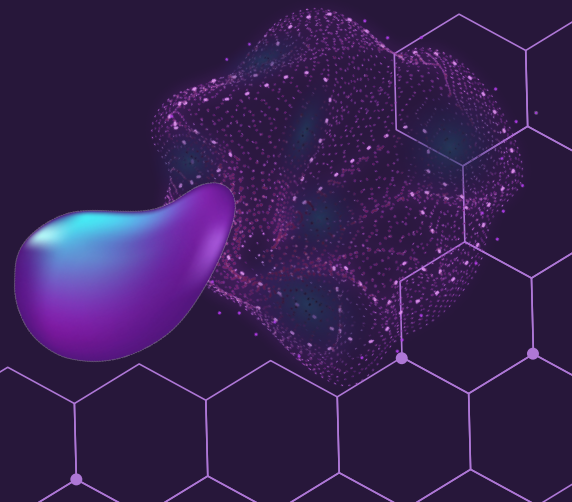
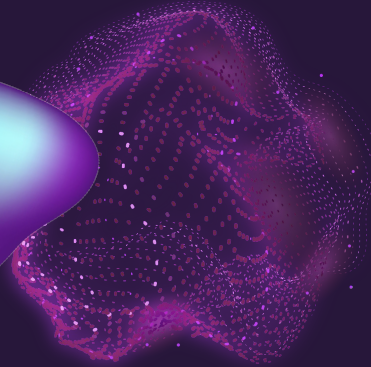
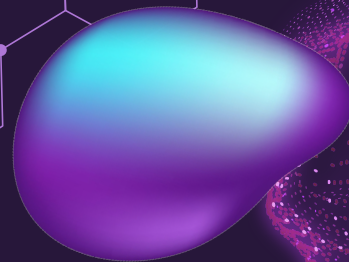
04

CATEGORICAL

NAIVEBAYES

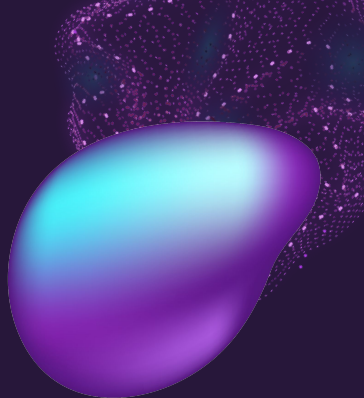


+



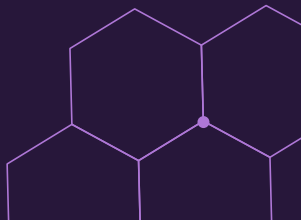


x



x

x

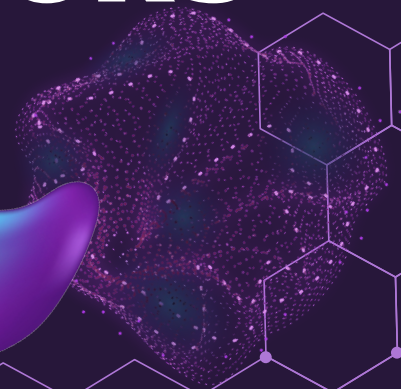
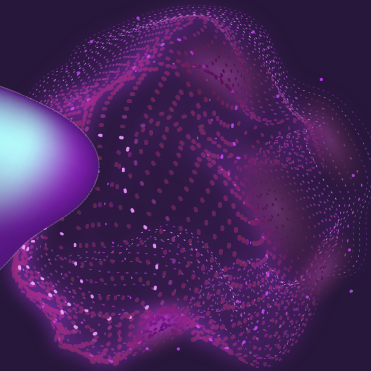
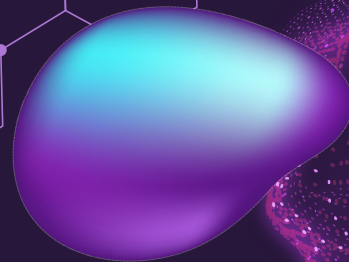


05

K-NEAREST NEIGHBOURS



+



total tries: 21

total submissions: 18

started accuracy: 0.53003

highest accuracy: 0.85212 (case 88)

highest case parameters:

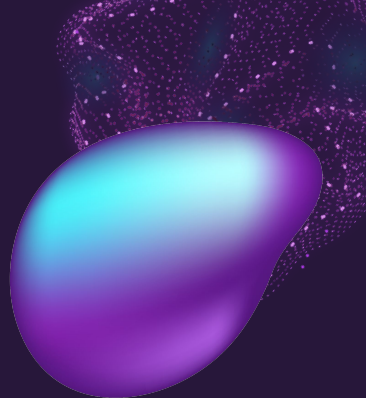
- scaler = minmax
- imputer = knn=3
- feature selector = kbest, at k = 5
- k-nearest neighbours at k = 1500

analysis:

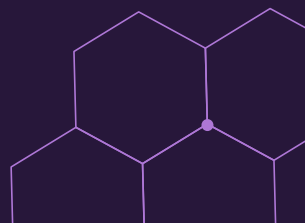
- kbest works at lower number of features
- knn=3, knn=7, simple imputers have no difference on accuracy
- forward selector + k-nearest is very time taking, even after 17 hours it didnt work. at smaller forward selection and smaller k-nearest-neighbours, it runs after 2 hours but accuracy is too low due to less features
- performs best on k-nearest neighbours = 1500. the more the k, the higher the accuracy
- PCA does not perform well

Analyzing K in K-nearest neighbours

case number	K=3	K=5	K=7	K=9	K=11	K=300	K=500	K=1000	K=1500	K=2000
43	-	0.53003	-	-	-	-	-	-	-	-
46	-	-	0.54796	-	-	-	-	-	-	-
47	-	-	0.57056	-	-	-	-	-	-	-
52	0.55037	-	-	-	-	-	-	-	-	-
53	-	-	-	-	0.59883	-	-	-	-	-
54	-	-	-	-	0.60509	-	-	-	-	-
55	-	-	-	-	0.61709	-	-	-	-	-
56	-	-	-	-	0.61709	-	-	-	-	-
57	-	-	-	-	0.62622	-	-	-	-	-
59	-	-	-	-	0.62207	-	-	-	-	-
63	-	-	-	0.63158	-	-	-	-	-	-
64	-	-	0.61114	-	-	-	-	-	-	-
79	-	-	-	-	-	0.81121	-	-	-	-
82	-	-	-	-	-	-	0.82533	-	-	-
85	-	-	-	-	-	-	-	0.83911	-	-
88	-	-	-	-	-	-	-	-	0.85212	-
86b	-	-	-	-	-	-	-	-	-	0.82641



×



BEFORE USING LARGE K:

- we can have mixed analysis to this table. $k=9$ was only tested once but it had the highest, although $k=11$ has a continuous record of being high
- $k=3$ was only tested once because a lower k means very few variables are taken into consideration whereas the dataset is very large.

AFTER USING LARGE K:

- we see that there is a linear relationship or exponential relationship between k -nearest neighbours and accuracy. the more the k , the better the accuracy - but there is a breakpoint where increasing k decreases the accuracy. this breakpoint was found between $k=1500$ and $k=2000$ where $k=1500$ has a very high accuracy but $k=2000$ has low.
- mistakenly, odd k 's should have been used. this was realized after all submissions were made.

Analyzing KBest Feature Selection

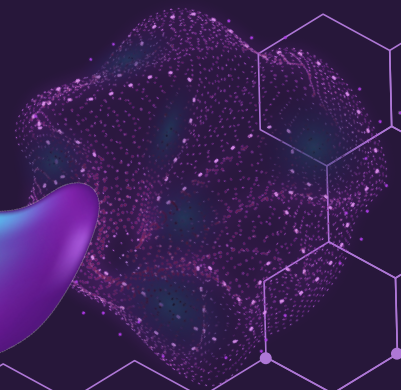
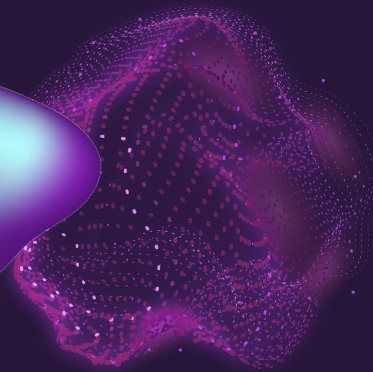
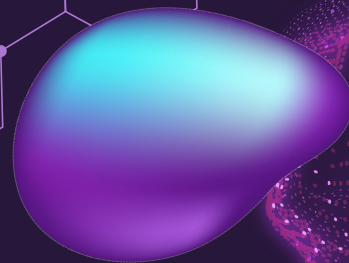
case number	algo used	kbest features	kaggle accuracy
59	k nearest neighbours, 11	3	0.62207
57	k nearest neighbours, 11	5	0.62622
55	k nearest neighbours, 11	10	0.61709
54	k nearest neighbours, 11	15	0.60509
53	k nearest neighbours, 11	20	0.59883
47	k nearest neighbours, 7	30	0.57056

kbest works better with lower number of features. as according to this table, kbest=5 is the best breakpoint.

06 RANDOM FOREST



+



×

total tests: 13
total submissions: 12
started accuracy: 0.90507
highest accuracy: 0.93581 (case 184)
highest parameters:

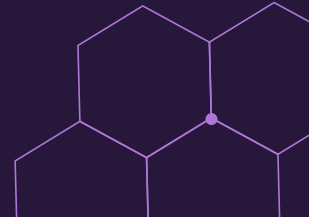
- imputer: knn=7
- scaler: maxabs
- max depth: 11
- estimators: 400
- criterion: entropy
- min samples split: 15
- max features: 60
- min samples leaf: 80
- algorithm feature importance: 20 features

×

×



analysis:

- higher depth of trees allows greater accuracy while lower depth moves to underfitting
 - gini underperforms while entropy performs way better
 - kbest feature selection doesnt work well with random forest (wasnt very tested heavily to say this)
 - algorithm feature importance improves the accuracy
 - smaller number of features extracted and selected is better
 - grid search takes alot of time in random forest
 - kbest does give good accuracy but not the best.
 - PCA did not perform well
- 

Analyzing Depth of Trees

case number	max_depth	kaggle accuracy
51, 58	5	0.91554, 0.91889
60	6	0.91309
62	7	0.92693
66	8	0.93079
125, 133	9	0.92154, 0.92877
44, 68	10	0.90507, 0.93256
78, 83, 118	11	0.93452, 0.92633, 0.93546

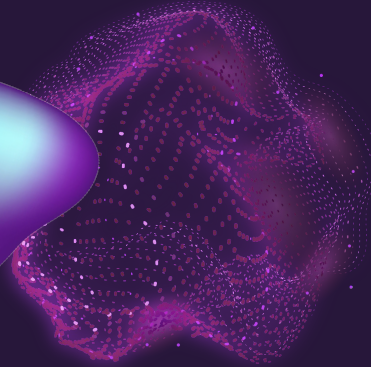
from here we can see that best accuracy is on depth=10 and depth=11 and depth=8 to some extent. thus, the larger the tree, the more the near the breakpoint. we have not found the breakpoint yet.


07

GRADIENT BOOSTING



+





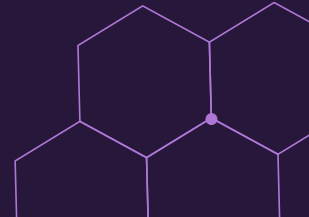
total tests: 12

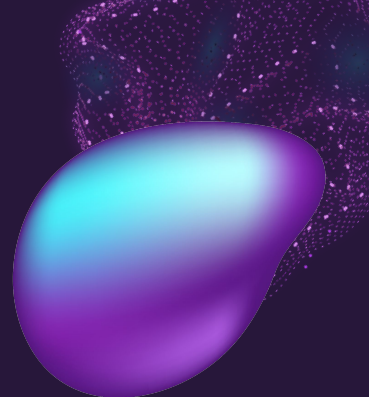
total submissions: 10

started accuracy: 0.88298

highest accuracy: 0.90485 (case 155)

highest case parameters:

- imputer: simple
 - scaler: minmax
 - max depth: 3
 - estimators: 200
 - criterion: friedman_mse
 - bagging of 10 estimators
 - no feature selection
- 



x

x



analysis:

- each submission took AT LEAST three hours and more, some submissions took exceptional time like 12+ hours.
- boosting is itself a very slow algorithm. using any forward or backward feature selector takes more than 12 hours and still doesnt even run on train data let alone full data and prediction
- the max_depth breakpoint was 6. too high depth leads to overfitting thus low accuracy
- kbest could not be rigourously tested as each submission took over 3 hours.
- bagging does work but it takes extremely long, **over 24 hours of running** and laptop use. its not very feasible. accuracy improved by 2 percent but very inefficient. better to use 10 estimators instead of 50, whereas 50 gives better results
- friedman_mse criterion is much better than squared_error
- uptil now, more estimators means better accuracy
- algorithm feature importance wasnt so good, maybe it woud have been better with bagging but overall didnt perform so good
- PCA did not perform well



x

Analyzing depth

case number	depth	accuracy
116, 122, 131	3	0.84769, 0.88551, 0.90057
61c, 65, 89, 102	6	0.88298, 0.88297, 0.85929, 0.90158
80	8	0.83659
67	10	0.79753

according to this table, higher depth doesnt have much power, but diffenret values such as bagging and other parameters matter. on average, smaller depths performed better.

x

08 ADAPTIVE BOOSTING



+



total tests: 17

total submissions: 15

starting accuracy: 0.94475

highest accuracy: 0.94966 (case 76) highest case parameters:

- imputer: simple
- scaler: minmax
- n estimators: 170
- no feature selector, no bagging

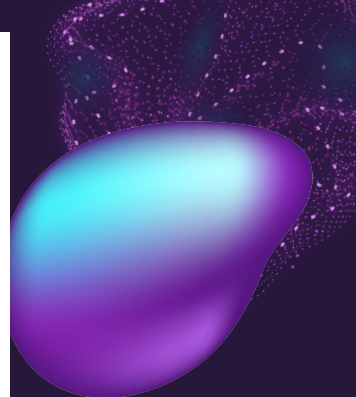
analysis:

- best estimator value is 170
- too many estimators and too less estimators can be wrong
- the default learning rate of 0.5 performs best
- grid search is very slow on adaboost and takes alot of time. especially with bagging
- bagging of 50 estimators is too slow
- greater estimators == lower learning rate
- bagging on best params didnt do so well and underperformed the model
- PCA does not perform well
- algorithm feature importance performs well, but not the best. with the same parameters and 78 features, the model performed better in 4 decimal places.
- kbest did not perform so well and decrease the accuracy by 0.05%

Analyzing Estimators with AdaBoost

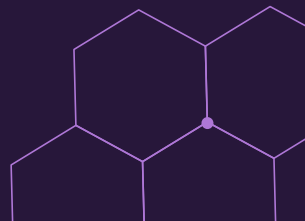
case number	n estimators	kaggle accuracy
71	50	0.93853
72	75	0.94053
69	100	0.94475
73	110	0.94521
74	150	0.94780
75	160	0.94948
76, 87, 90, 114	170	0.94966, 0.93369, 0.94896, 0.93386
81, 84	175	0.94949, 0.93301
77	180	0.93516
70	200	0.93379
142	300	0.86318

from this table we can see that roughly 170 estimators is the breakpoint with the highest accuracy. lesser than 100 is too less estimators and more than 200 is too many estimators. in our search to find the breakpoint, we tested 10 different estimator values and found 170 as the best. after more testing we can also see that too high estimators is bad.



×

×



Analyzing Learning Rate with AdaBoost

case number	learning rate	kaggle accuracy
142	0.005	0.86318 --estimators=300
84	0.1	0.93301
76, 114	default=0.5	0.94966, 0.93386
90	0.6	0.94896
87	0.75	0.93369

from this we can analyse that learning rate is best at default of 0.5, even though l.rate is good at 0.6, however best is at 0.5. having a too high or too low learning rate depreciates the accuracy performance.

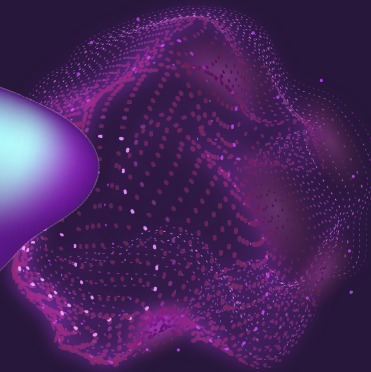
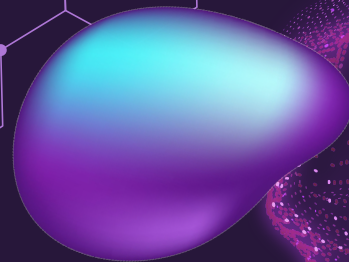
this table analyses with constant estimators of 170.

09

LIGHT GBM



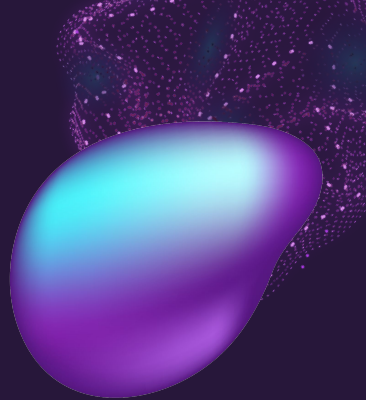
+

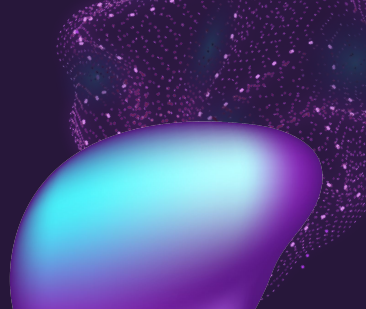


x

total tests: 25
total submissions: 21
starting accuracy: 0.75561
highest accuracy: 0.95323 (case 126c)
highest parameters:

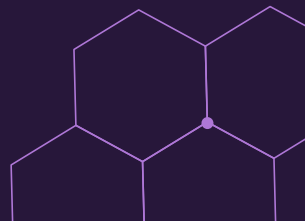
- imputer: simple
- scaler: maxabs
- max depth: 3
- estimators: 1000
- learning rate: 0.01
- bagging estimators: 50
- algorithm feature importance: 20 features





analysis:

- relationship found between number of estimators and learning rate. less estimators == high learning rate. more estimators == low learning rate
- 20 features are the breakpoint, too less decreases accuracy whether it be algorithm feature importance or kbest feature selector, and too many is also bad
- too much depth can be long and ineffective, smaller depth is better
- default min child weight is better, 20 has a higher roc however accuracy decreased
- increasing estimators without altering learning rate makes no difference
- PCA didnt perform well and ruined the accuracy



Analyzing Estimators and Learning Rate

case number	estimators	learning rate	accuracy
94	100	0.5	0.37172
91, 95, 96, 97	100	0.9	0.75561, 0.77939, 0.77699, 0.49841
98	200	0.1	0.87767
99, 100, 101, 103	300	0.01	0.94165, 0.94321, 0.94351, 0.94395
104b	300	0.05	0.94948
92	400	0.9	0.50533
93	400	0.5	0.50408
111, 119, 123c, 126c, 128, 132	1000	0.01	0.95106, 0.95087, 0.95070, 0.95323, 0.94903, 0.95242
136	3000	0.01	0.94173

based on this table, we can see that if estimators are small ≤ 100 , then a higher learning rate is preferred however if the learning rate is high ≥ 300 , $= 1000$, then a lower learning rate is preferred. too much high requires too much low, otherwise accuracy decreases.





x

Analyzing Algorithm Feature Importance

case number	features	accuracy
128	15	0.94903
126c, 136	20	0.95323, 0.94173
132	25	0.95242
123c	35	0.95070

according to this table, 20, 25, 35 are good accuracies however 20 is the highest. it does not perform well on lower number of features.

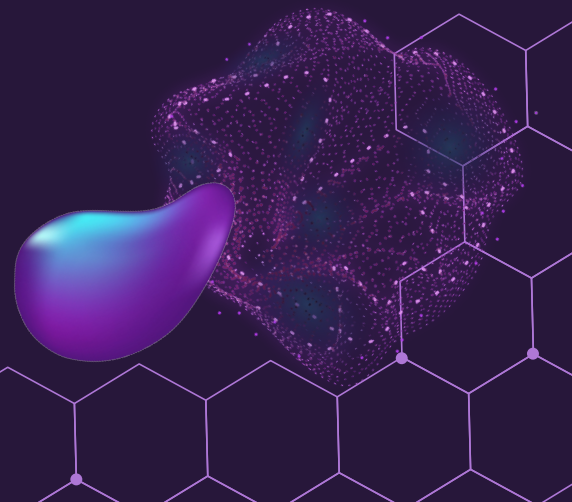
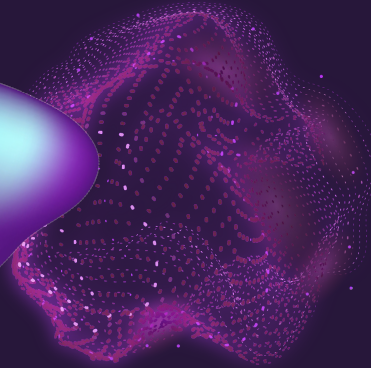
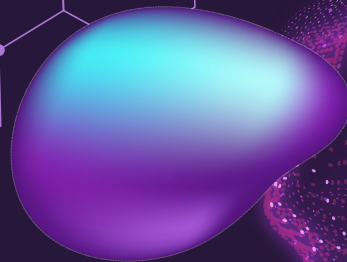


10 XG BOOST

x

x

+



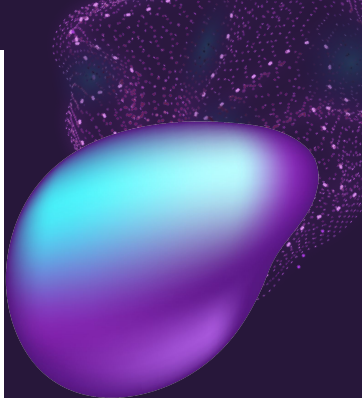
total tries: 22

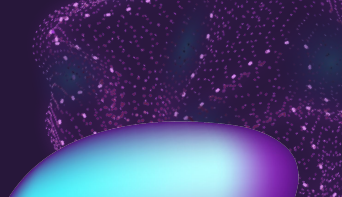
total submissions: 21

starting accuracy: 0.95474

highest accuracy: 0.95979 (case 138) highest parameters:

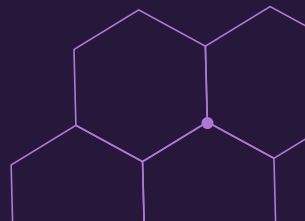
- imputer: simple
- scaler: maxabs
- no parameters in xgboost brackets
- bagging of 100 estimators
- algorithm feature importance, 35 features





analysis:

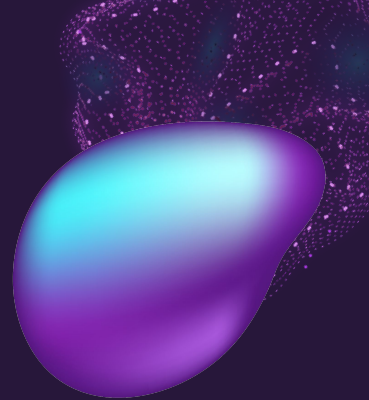
- after singular grids the best params were 100 estimators, 0.2 learning rate, 2 max depth but still accuracy is low
- after multiple grids the best params were 200 estimators, 0.1 learning rate, 2 max depth but still accuracy is low
- best accuracy was achieved at "None" parameters, when brackets are empty
- best algorithm feature importance is at 35 features. 40 is good as well, but 30 has alot of deterioration
- bagging was best at 100. i tested all bagging from 1 to 100 and the lowest roc was at bagging=3 and highest roc at bagging=4 but still accuracy was low on both. bagging=99 was an outlier, but bagging=100 and bagging=150 were highest.
- PCA does not work well



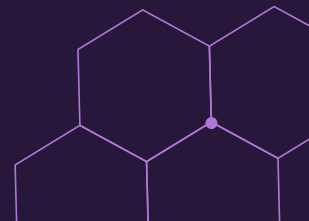
Analyzing Algorithm Feature Importance

case number	features	accuracy
129	30	0.94984
124	35	0.95846
130	36	0.95522
127	40	0.95371

35 features was breakpoint, more were good but less were bad



x



Analyzing Bagging

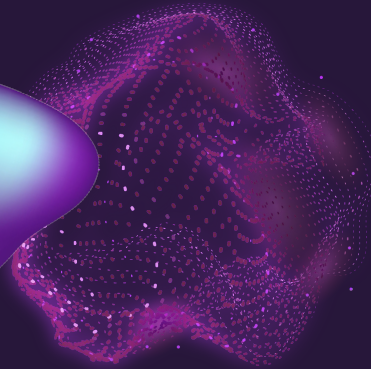
case number	bagging estimators	accuracy
145	3	0.94978
141	4	0.94413
135	50	0.94971
140	99	0.94394
138	100	0.95979
139	150	0.95586

bagging=4 is probably an outlier, as well as bagging=99, but bagging=100 is the best. too low baggers can be bad and too many baggers is very time consuming and ineffective.

11 CAT BOOST



+



no. of tries: 13

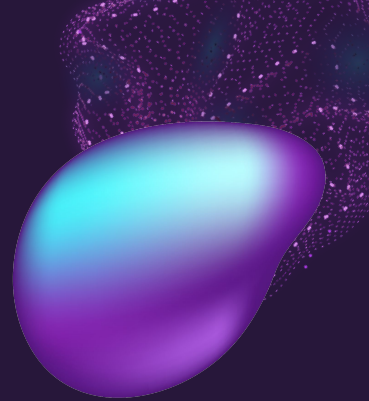
no. of submissions: 11

starting accuracy: 0.93798

highest accuracy: 0.95270 (case 144)

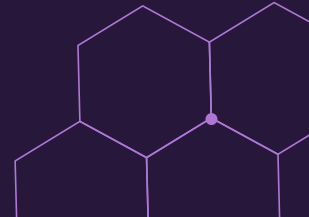
highest case parameters:

- imputer: simple
- scaler: maxabs
- no of estimators: 2000
- depth: 1
- learning rate: 0.1
- bagging, 50 estimators
- algorithm feature importance of 14 features





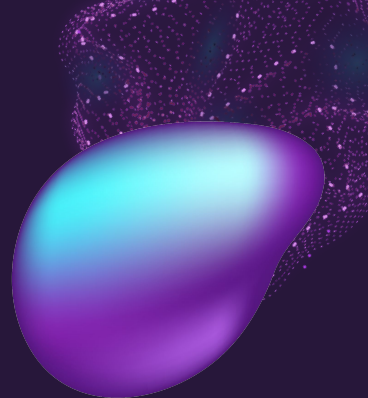
analyse:

- bagging improves the performance on average
 - higher depth is not good. smaller depth works better
 - default iterations are 1000, smaller iterations underperform, 2000 is the best. 2200, 2500, 3000 were also checked but they had lower roc in grid
 - the more the iterations, the relatively smaller learning rate
 - feature importance is AMAZING. accuracy immediately went into 95s
 - the smaller number of features, the better. breakpoint of best features number was found at 14
 - USES ALOT OF MEMORY/RAM. cannot be done with other boosting algorithms or the laptop terminates the process due to >95% usage of RAM
- 

Analyze Algorithm Feature Importance

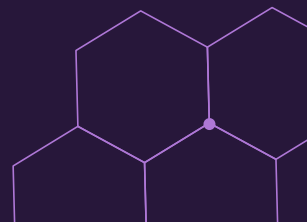
case number	features	accuracy
147	13	0.94950
144	14	0.95270
142	15	0.95191
137	20	0.95006

hence we can see the breakpoint at features=14, which has the highest accuracy



x

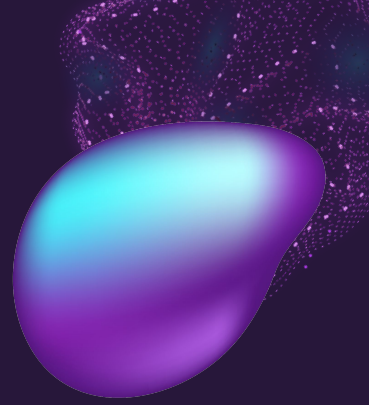
x



Analyze Bagging Estimators

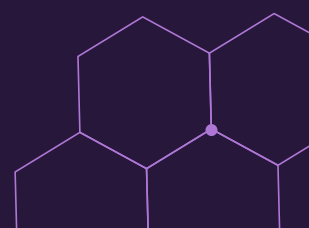
case number	estimators	accuracy
144	50	0.95270
157	75	0.95106
152	100	0.95263

best bagging is seen at 100 estimators, while 75 may be an outlier.



×

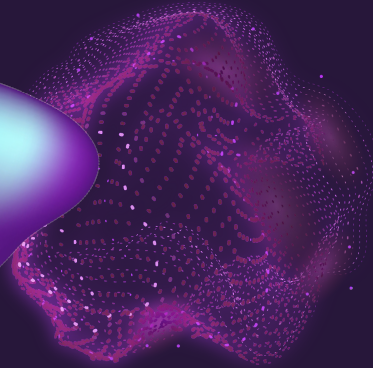
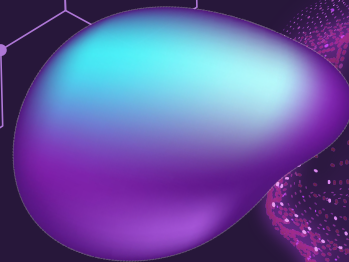
×



13 EXTREMELY RANDOMIZED TREE (ERT)



+



x

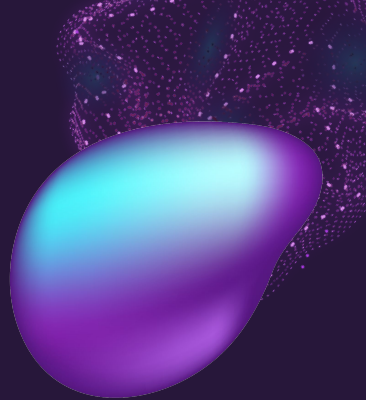
total tries: 8

total submissions: 8

starting accuracy: 0.91809

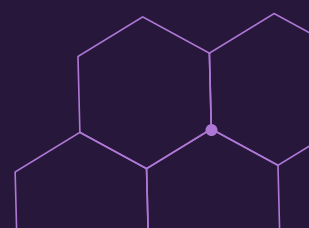
highest accuracy: 0.92081 (case 160) highest case parameters:

- imputer: simple
- scaler: maxabs
- no feature importance or selection
- n_estimators = 800
- bootstrap is default True
- bagging is of 50 estimators



x

x



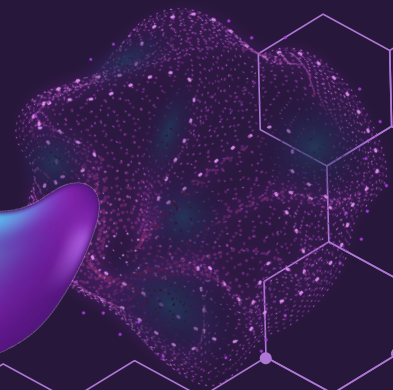
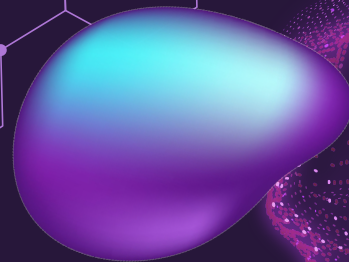
analysis:

- doesnot have a difference with bootstrap=True and boostrap=False
- estimators matter alot - the lower they are the worse
- bagging improves results but on average is very slow and requires alot of time
- performs better on bagging=50 instead of bagging=100
- more estimators improves accuracy
- kbest did not perform well and decreased the accuracy by 5%
- algorithm feature importance aso didnt perform well
- PCA did not perform well

14 VOTING CLASSIFIER



+



analysis:

- the more the models the better
- better with bagged lgbm
- better with xgb algo feature importances

total tries: 10

total submissions: 10

starting accuracy: 0.95250

highest accuracy: 0.96178 (case 201)

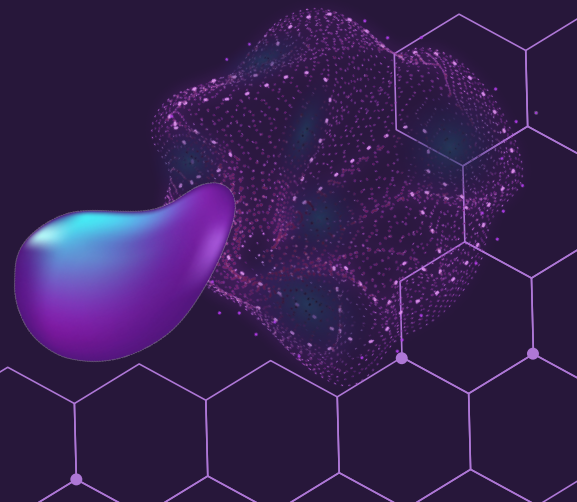
highest case parameters:

- model = lgb.LGBMClassifier(learning_rate=0.02, max_depth=2, n_estimators=3500)
- model_2 = xgb.XGBClassifier(max_depth=5, n_estimators=250, learning_rate=0.1, eval_metric='auc', random_state=42)
- model_2 = featureImportance(model_2, 45)
- model_1 = BaggingClassifier(estimator=model, n_estimators=50, max_features=0.8, max_samples=0.8, bootstrap=True, random_state=42, verbose = 2)
- model = VotingClassifier(estimators=[('one', model), ('two', model_1)], voting='soft', verbose=True)
- simple imputer
- minmax scaler

15 STACKING



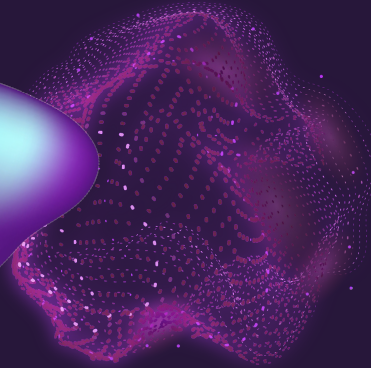
+



x



x

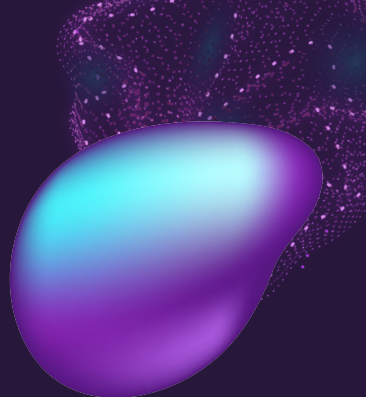




```
total submissions: 9
total tries: 12
highest: 0.95582 (case 209)
- xgb = xgb.XGBClassifier(n_estimators=500, learning_rate=0.05, max_depth=3, subsample=1.0, device = 'cuda') #---40 features
- LGBM = lgb.LGBMClassifier(learning_rate=0.02, max_depth=2, n_estimators=3500 , device = 'gpu')
- Ada = AdaBoostClassifier(n_estimators = 170)
- RF1 = RandomForestClassifier(criterion="entropy", max_depth=13, min_samples_leaf=60, min_samples_split=15, n_estimators=400)
- DT1 = DecisionTreeClassifier(criterion="entropy", max_depth=5, min_samples_leaf=70, min_samples_split=18)
- classifier = StackingClassifier(estimators=[('xgb', xgb),('LGBM',LGBM),('Ada', Ada) , ('RF1',RF1), ('DT1',DT1)],final_estimator
= xgb)
- model_4 = featureImportance( xgb.XGBClassifier(), 45 )
```



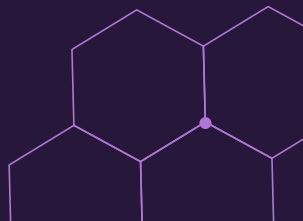
x



analysis:

- bagged stacking worked better, evaluated models better
- random forest stacking is too lengthy and long
- stacking did not perform as well as the models did. voting classifier however performed much better

x





THANK YOU

ZUHA AQIB 26106

IBA, Fall 2024

