In [22]:

```
import numpy as np
import pandas as pd
import scipy.stats as sps
import warnings

from sklearn.model_selection import train_test_split
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score

import xgboost as xgb

warnings.filterwarnings("ignore")
```

Подбор гиперпараметров для XGBoost

Рассмотрим ключевые гиперпараметры модели XGBoost:

- num_round - количество итераций бустинга.
- max_depth - максимальная длина построенного дерева. Если с увеличением данного параметра модель не дает более хорошего качества, то стоит задуматься о feature engineering.
- subsample - доля данных, которые подаются для обучения очередного дерева.
- colsample_bytree - доля признаков, которые используются при построении очередного дерева. Подмножество признаков выбирается по одному разу для каждого дерева.
- colsample_bylevel - доля признаков, которые используются при построении очередного уровня дерева (новый уровень глубины). Подмножество признаков выбирается по одному разу для каждого уровня.
- lambda - коэффициент l_2-регуляризации.
- alpha - коэффициент l_1-регуляризации.
- eta - вес, с которым добавляется предсказание нового построенного дерева.

1. Подбор параметров по сетке

Применим XGBoost Classifier к задаче классификации мобильных телефонов по цене (https://www.kaggle.com/iabhishekofficial/mobile-price-classification)

In [23]:

```
data = pd.read_csv("./mobile-price-classification/train.csv")
```

In [24]:

```
X_train, X_test, y_train, y_test = train_test_split(
    data.drop(['price_range'], axis=1, inplace=False),
    data['price_range'], test_size=0.2,
    shuffle=True, stratify=data['price_range']
)
```

Мы уже познакомились с техникой `GridSearch`. Ее достоинство заключается в том, что мы делаем *полный* перебор по пространству поиска, но это же и является ее недостатком. Когда пространство поиска имеет слишком большую размерность, предпочтительнее использовать `RandomizedSearch`. Суть данной техники похожа на полный перебор по сетке с разницей лишь в том, что генерируется подмножество из всех возможных комбинаций значений гиперпараметров и рассматриваются только модели, соответствующие этим комбинациям.

Посмотрим на значения гиперпараметров по умлочанию:

In [25]:

```
model = xgb.XGBClassifier()
model.get_params()
```

Out[25]:

```
{'objective': 'binary:logistic',
 'base_score': None,
 'booster': None,
 'colsample_bylevel': None,
 'colsample_bynode': None,
 'colsample_bytree': None,
 'gamma': None,
 'gpu_id': None,
 'importance_type': 'gain',
 'interaction_constraints': None,
 'learning_rate': None,
 'max_delta_step': None,
 'max_depth': None,
 'min_child_weight': None,
 'missing': nan,
 'monotone_constraints': None,
 'n_estimators': 100,
 'n_jobs': None,
 'num_parallel_tree': None,
 'random_state': None,
 'reg_alpha': None,
 'reg_lambda': None,
 'scale_pos_weight': None,
 'subsample': None,
 'tree_method': None,
 'validate_parameters': False,
 'verbosity': None}
```

Задаем пространство поиска. В отличие от `GridSearch` можно задавать распределения:

In [26]:

```
parameters_grid = {
    'num_round' : [10, 50, 100, 1000, 1500, 2000],
    'max_depth' : range(1, 15, 2),
    'subsample' : [0.8, 0.9, 1.0],
    'colsample_bytree' : [0.7, 0.8, 0.9, 1.0],
    'colsample_bylevel' : [0.7, 0.8, 0.9, 1.0] ,
    'lambda' : sps.expon(0),
    'alpha' : sps.expon(0),
    'eta' : [0.001, 0.1, 1, 10]
}
```

In [27]:

```
# задаем стратегию кросс-валидации
ss = StratifiedKFold(n_splits=3)

# определяем поиск по сетке
gs = RandomizedSearchCV(
    # модель для обучения, в нашем случае XGBoostClassifier
    estimator=model,
    # количество итераций поиска
    n_iter=200,
    # сетка значений гиперпараметров
    param_distributions=parameters_grid,
    # метрика качества, берем accuracy
    scoring='accuracy',
    # GridSearch отлично параллелится, указываем количество параллельных джоб
    n_jobs=-1,
    # стратегия кросс-валидации
    cv=ss,
     # сообщения с логами обучения: больше значение - больше сообщений
    verbose=10,
    # значение, присваиваемое scorer в случае ошибки при обучении
    error_score='raise'
)
```

Выполняем поиск по сетке

In [28]:

```
%%time
# выполняем поиск по сетке
gs.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent wor
kers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.3s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    1.3s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    1.9s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    2.7s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.6s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:    4.2s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    5.6s
[Parallel(n_jobs=-1)]: Done  69 tasks      | elapsed:    7.0s
[Parallel(n_jobs=-1)]: Done  82 tasks      | elapsed:    8.5s
[Parallel(n_jobs=-1)]: Done  97 tasks      | elapsed:   10.0s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:   11.2s
[Parallel(n_jobs=-1)]: Done 129 tasks      | elapsed:   13.1s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   15.0s
[Parallel(n_jobs=-1)]: Done 165 tasks      | elapsed:   15.8s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   17.8s
[Parallel(n_jobs=-1)]: Done 205 tasks      | elapsed:   18.9s
[Parallel(n_jobs=-1)]: Done 226 tasks      | elapsed:   22.0s
[Parallel(n_jobs=-1)]: Done 249 tasks      | elapsed:   24.2s
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed:   26.7s
[Parallel(n_jobs=-1)]: Done 297 tasks      | elapsed:   28.9s
[Parallel(n_jobs=-1)]: Done 322 tasks      | elapsed:   30.7s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   33.0s
[Parallel(n_jobs=-1)]: Done 376 tasks      | elapsed:   36.0s
[Parallel(n_jobs=-1)]: Done 405 tasks      | elapsed:   38.6s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   41.6s
[Parallel(n_jobs=-1)]: Done 465 tasks      | elapsed:   43.5s
[Parallel(n_jobs=-1)]: Done 496 tasks      | elapsed:   46.3s
[Parallel(n_jobs=-1)]: Done 529 tasks      | elapsed:   48.8s
[Parallel(n_jobs=-1)]: Done 562 tasks      | elapsed:   52.6s
[Parallel(n_jobs=-1)]: Done 600 out of 600 | elapsed:   57.4s finished

CPU times: user 4.29 s, sys: 298 ms, total: 4.59 s
Wall time: 57.7 s
```

Out[28]:

```
RandomizedSearchCV(cv=StratifiedKFold(n_splits=3, random_state=None, s
huffle=False),
                   error_score='raise',
                   estimator=XGBClassifier(base_score=None, booster=No
ne,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamm
a=None,
                                           gpu_id=None, importance_typ
e='gain',
                                           interaction_constraints=Non
e,
                                           learning_rate=None,
                                           max_delta_step=None, max_de
pth=None...
```

```
                                          'colsample_bylevel': [0.7, 0.
8, 0.9,
                                                               1.0],
                                          'colsample_bytree': [0.7, 0.8,
0.9,
                                                              1.0],
                                          'eta': [0.001, 0.1, 1, 10],
                                          'lambda': <scipy.stats._distn_
infrastructure.rv_frozen object at 0x1c1e3069d0>,
                                          'max_depth': range(1, 15, 2),
                                          'num_round': [10, 50, 100, 100
0, 1500,
                                                        2000],
                                          'subsample': [0.8, 0.9, 1.0]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=T
rue,
                   return_train_score=False, scoring='accuracy', verbo
se=10)
```

Найденные оптимальные гиперпараметры:

In [29]:

```
1  gs.best_params_
```

Out[29]:

```
{'alpha': 0.4097610592073989,
 'colsample_bylevel': 0.8,
 'colsample_bytree': 1.0,
 'eta': 1,
 'lambda': 0.42655405988358847,
 'max_depth': 1,
 'num_round': 1500,
 'subsample': 1.0}
```

Оптимальное значение метрики:

In [30]:

```
1  gs.best_score_
```

Out[30]:

```
0.9075
```

Оцениваем качество на тестовой выборке и сравниваем с дефолтной моделью.

In [31]:

```
best_model = gs.best_estimator_
best_model.fit(X_train, y_train)
y_pred_best = best_model.predict(X_test)
accuracy_score(y_test, y_pred_best)
```

Out[31]:

```
0.9225
```

In [32]:

```
default_model = xgb.XGBClassifier()
default_model.fit(X_train, y_train)
y_pred_default = default_model.predict(X_test)
accuracy_score(y_test, y_pred_default)
```

Out[32]:

```
0.8975
```

При помощи случайного поиска гиперпараметров по сетке мы получили более качественную модель.

2. Эвристический подход к настройке гиперпараметров

Для работы с большими датасетами поиск по сетке не представляется возможным, поэтому будем действовать по следующему принципу. Для других бустингов (`LightGBM` , `CatBoost` и etc.) рекомендуется действовать аналогично:

1) Фиксируем параметр `learning_rate` . Обычно берут `0.1` , но для разных задач он устанавливается от `0.05` до `0.3` . Определяем оптимальное количество деревьев для **данного `learning_rate`**

2) Настраиваем параметры дерева: `max_depth` , `min_child_weight` , `gamma` , `subsample` , `colsample_bytree` для фиксированном количестве деревьев и `learning_rate` .

3) Настраиваем параметры регуляризации: `lambda` и `alpha` , что поможет нам уменьшить сложность модели и повысить ее производительность

4) Понижаем `learning_rate` и повторяем пункты 1-3 до сходимости. Возможно, при понижении `learning_rate` не удасться добиться лучшего качества.

Для демонстрации данного подхода будем использовать нативный интерфейс `XGBoost` :

In [33]:

```
xgtrain = xgb.DMatrix(X_train.values, y_train.values)
xgtest = xgb.DMatrix(X_test.values, y_test.values)
evallist = [(xgtest, 'eval'), (xgtrain, 'train')]
```

Шаг 1. Настройка параметров бустинга

Для настройки параметров бустинга мы должны присовить некоторые начальные значения для всех остальных параметров (дерева и регуляризации):

1) `max_depth = 5` . Обычно этот параметр варируется от 3 до 10. Значения 4-6 -- хорошее стартовое приближение

2) `min_child_weight = 1` . Здесь логики нет, выбираем исходя из задачи

3) `gamma = 0.1` . Обычно этот параметр варьируем от 0 до 0.2. В дальнейшей настройке мы его подберем

4) `subsample, colsample_bytree = 0.8` . Это часто используемое начальное приближение. Обычно его варируют от 0.5 до 0.9

Обращаем ваше внимание, что все эти параметры являются лишь первоначальной оценкой и будут настроены позже

Теперь зафиксируем `learning_rate = 0.1` и найдем оптимальное количество деревьев с помощью кросс-валидации:

In [34]:

```
xgb_params = {
    "num_class": 4,
    "booster": "gbtree",
    "n_estimators": 1000,
    "learning_rate": 0.1,
    "max_depth": 5,
    "min_child_weight": 1,
    "gamma": 0.1,
    "subsample": 0.8,
    "colsample_bytree": 0.8
}
```

In [35]:

```
cvresult = xgb.cv(
    xgb_params, xgtrain, num_boost_round=xgb_params['n_estimators'],
    nfold=5, early_stopping_rounds=30
)
```

In [36]:

```
cvresult[-3:]
```

Out[36]:

	train-merror-mean	train-merror-std	test-merror-mean	test-merror-std
99	0.0	0.0	0.100000	0.011693
100	0.0	0.0	0.098750	0.010933
101	0.0	0.0	0.096875	0.011524

Видим, что обучение прекратилось при добавлении 101 дерева. Установим параметр `n_estimators=101`

Шаг 2. Настройка параметров дерева

2.1. Настройка параметров `max_depth` и `min_child_weight`

Первыми настраиваем параметры `max_depth` и `min_child_weight` , т.к. они дадут наибольший вклад в финальный результат модели. Здесь все тривиально, настраиваем по сетке (или перебираем руками):

In [37]:

```
param_grid = {
    "max_depth": range(3, 10),
    "min_child_weight": range(1, 10),
}

xgb_params = {
    "num_class": 4,
    "booster": "gbtree",
    "n_estimators": 101,
    "learning_rate": 0.1,
    "gamma": 0.1,
    "subsample": 0.8,
    "colsample_bytree": 0.8
}
```

In [38]:

```
gs = GridSearchCV(xgb.XGBClassifier(**xgb_params), param_grid,
                  n_jobs=4, scoring='accuracy', verbose=1, cv=5)
```

In [39]:

```
1  gs.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 63 candidates, totalling 315 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    5.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:   30.0s
[Parallel(n_jobs=4)]: Done 315 out of 315 | elapsed:   54.5s finished
```

Out[39]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=None, booster='gbtre
e',
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.8, gamma=0.1,
                                     gpu_id=None, importance_type='gai
n',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step
=None,
                                     max_depth=None, min_child_weight=
None,
                                     missing=nan, monotone_constra...
                                     num_parallel_tree=None,
                                     objective='binary:logistic',
                                     random_state=None, reg_alpha=Non
e,
                                     reg_lambda=None, scale_pos_weight
=None,
                                     subsample=0.8, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='warn', n_jobs=4,
             param_grid={'max_depth': range(3, 10),
                         'min_child_weight': range(1, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='accuracy', verbose=1)
```

In [40]:

```
1  gs.best_params_
```

Out[40]:

```
{'max_depth': 4, 'min_child_weight': 2}
```

Оптимальными параметрами оказались `max_depth = 4`, `min_child_weight = 2`

2.2. Настраиваем параметр `gamma`

Настройка будет происходить для уже подобранных параметров выше. Параметр `gamma` может принимать различные значения в зависимости от задачи. Рекомендуется перебирать от 0 до 0.5:

In [41]:

```
param_grid = {
    "gamma": [i / 10 for i in range(5)]
}

xgb_params = {
    "num_class": 4,
    "booster": "gbtree",
    "n_estimators": 101,
    "learning_rate": 0.1,
    "max_depth": 4,
    "min_child_weight": 2,
    "subsample": 0.8,
    "colsample_bytree": 0.8
}
```

In [42]:

```
gs = GridSearchCV(xgb.XGBClassifier(**xgb_params), param_grid,
                  n_jobs=4, scoring='accuracy', verbose=1, cv=5)
```

In [43]:

```
1  gs.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=4)]: Done  25 out of  25 | elapsed:    4.0s finished
```

Out[43]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=None, booster='gbtre
e',
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.8, gamma=None,
                                     gpu_id=None, importance_type='gai
n',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step
=None,
                                     max_depth=4, min_child_weight=2,
                                     missing=nan, monotone_constraints
=...
                                     n_estimators=101, n_jobs=None, nu
m_class=4,
                                     num_parallel_tree=None,
                                     objective='binary:logistic',
                                     random_state=None, reg_alpha=Non
e,
                                     reg_lambda=None, scale_pos_weight
=None,
                                     subsample=0.8, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='warn', n_jobs=4,
             param_grid={'gamma': [0.0, 0.1, 0.2, 0.3, 0.4]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='accuracy', verbose=1)
```

In [44]:

```
1  gs.best_params_
```

Out[44]:

```
{'gamma': 0.1}
```

Оптимальным значением оказалась `gamma = 0.1`

2.3. Настраиваем `subsample` и `colsample_bytree`

In [45]:

```
param_grid = {
    "subsample": [i / 10 for i in range(4, 11)],
    "colsample_bytree": [i / 10 for i in range(4, 11)]
}

xgb_params = {
    "num_class": 4,
    "booster": "gbtree",
    "n_estimators": 101,
    "learning_rate": 0.1,
    "max_depth": 4,
    "gamma": 0.1,
    "min_child_weight": 2
}
```

In [46]:

```
gs = GridSearchCV(xgb.XGBClassifier(**xgb_params), param_grid,
                  n_jobs=4, scoring='accuracy', verbose=1, cv=5)
```

In [47]:

```
1  gs.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    4.3s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:   24.0s
[Parallel(n_jobs=4)]: Done 245 out of 245 | elapsed:   32.6s finished
```

Out[47]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=None, booster='gbtre
e',
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=0.1,
                                     gpu_id=None, importance_type='gai
n',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step
=None,
                                     max_depth=4, min_child_weight=2,
                                     missing=nan, monotone_constraints
=...
                                     random_state=None, reg_alpha=Non
e,
                                     reg_lambda=None, scale_pos_weight
=None,
                                     subsample=None, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='warn', n_jobs=4,
             param_grid={'colsample_bytree': [0.4, 0.5, 0.6, 0.7, 0.8,
0.9,
                                              1.0],
                         'subsample': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
1.0]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='accuracy', verbose=1)
```

In [48]:

```
1  gs.best_params_
```

Out[48]:

```
{'colsample_bytree': 0.9, 'subsample': 0.4}
```

Оптимальными параметрами оказались `colsample_bytree = 0.9`, `subsample = 0.4`

Шаг 3. Настройка параметров регуляризации

In [49]:

```
param_grid = {
    'alpha': [0, 0.001, 0.005, 0.01, 0.05],
    'lambda': [0, 0.001, 0.005, 0.01, 0.05],
}

xgb_params = {
    "num_class": 4,
    "booster": "gbtree",
    "n_estimators": 101,
    "learning_rate": 0.1,
    "max_depth": 4,
    "gamma": 0.1,
    "min_child_weight": 2,
    "subsample": 0.4,
    "colsample_bytree": 0.9,
}
```

In [50]:

```
gs = GridSearchCV(xgb.XGBClassifier(**xgb_params), param_grid,
                  n_jobs=4, scoring='accuracy', verbose=1, cv=5)
```

In [51]:

```
1  gs.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    5.9s
[Parallel(n_jobs=4)]: Done 125 out of 125 | elapsed:   17.3s finished
```

Out[51]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=None, booster='gbtre
e',
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.9, gamma=0.1,
                                     gpu_id=None, importance_type='gai
n',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step
=None,
                                     max_depth=4, min_child_weight=2,
                                     missing=nan, monotone_constraints
=N...
                                     objective='binary:logistic',
                                     random_state=None, reg_alpha=Non
e,
                                     reg_lambda=None, scale_pos_weight
=None,
                                     subsample=0.4, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='warn', n_jobs=4,
             param_grid={'alpha': [0, 0.001, 0.005, 0.01, 0.05],
                         'lambda': [0, 0.001, 0.005, 0.01, 0.05]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='accuracy', verbose=1)
```

In [52]:

```
1  gs.best_params_
```

Out[52]:

```
{'alpha': 0.005, 'lambda': 0.01}
```

Оптимальными параметрами оказались `alpha = 0.005`, `lambda = 0.01`

In [53]:

```
best_estimator = gs.best_estimator_
best_estimator.fit(X_train, y_train)
y_pred_best = best_estimator.predict(X_test)
accuracy_score(y_test, y_pred_best)
```

Out[53]:

```
0.9
```

Получили самое высокое качество из всех представленных моделей

Другие бустинги

Ниже приведена таблица со сравненением параметров в других популярных бустингах. Подборы гиперпараметров будут происходить аналогично, но исходя из специфики гиперпараметра в данном алгоритме

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100