|  |  |  |
| --- | --- | --- |
| ALGORITHMS | BASIC PARAMETERS THAT ARE VARIED FOR BETTER RESULTS | GRID SEARCH AND RANDOM SEARCH PARAMETERS |
| SVC | clf = SVC(gamma='auto') | [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],  'C': [1, 10, 100, 1000]}, |
| Kmeans | kmeans = KMeans(n\_clusters=2, random\_state=0).fit(X) | ca = KMeans() param\_grid = {"n\_clusters": range(2, 11)} |
| RF Classifier | clf = RandomForestClassifier(n\_estimators=100, max\_depth=2,  **...**  random\_state=0) | random\_grid = {'n\_estimators': n\_estimators,  'max\_features': max\_features,  'max\_depth': max\_depth,  'min\_samples\_split': min\_samples\_split,  'min\_samples\_leaf': min\_samples\_leaf,  'bootstrap': bootstrap} |
| XGBClassifier() | from xgboost import XGBClassifier  model = XGBClassifier()  model.fit(X\_train, y\_train) | #brute force scan for all parameters, here are the tricks  #usually max\_depth is 6,7,8  #learning rate is around 0.05, but small changes may make big diff  #tuning min\_child\_weight subsample colsample\_bytree can have  #much fun of fighting against overfit  #n\_estimators is how many round of boosting  #finally, ensemble xgboost with multiple seeds may reduce variance  parameters = {'nthread':[4], #when use hyperthread, xgboost may become slower  'objective':['binary:logistic'],  'learning\_rate': [0.05], #so called `eta` value  'max\_depth': [6],  'min\_child\_weight': [11],  'silent': [1],  'subsample': [0.8],  'colsample\_bytree': [0.7],  'n\_estimators': [5], #number of trees, change it to 1000 for better results  'missing':[-999],  'seed': [1337]} |
| OneClassSVM | clf = [svm.OneClassSVM](https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html#sklearn.svm.OneClassSVM)(nu=0.1, kernel="rbf", gamma=0.5) | |  | | --- | | :parameters: | |  |  | | |  | | |  |  | | |  |  | | |  |  | | |  |  | | |  |  | | |  |  | | |  |  | | |  |  | | |  |  | |   - estimator : sklearn.base.ClassifierMixin  A binary classifier. Must support the ``predict\_proba`` method.  - sample\_estimator : str  One of:  - mean : arithmetic mean of probabilities  - gmean : arithmetic mean of log probabilities  - max : max probability  - ucb : upper confidence bound : mean + 3\*stdev  - gucb : ucb in log space |
| Hierarchical Clustering | clustering = AgglomerativeClustering().fit(X) | ac = AgglomerativeClustering(memory='mycachedir',  compute\_full\_tree=True) |
| LinearRegression | reg = LinearRegression().fit(X, y) | parameters = {'fit\_intercept':('True', 'False'), 'normalize':('True', 'False'), 'copy\_X':('True', 'False')} |
| RandomForestRegressor | regr = RandomForestRegressor(max\_depth=2, random\_state=0,  **...**  n\_estimators=100) | tuned\_parameters = {'n\_estimators': [500, 700, 1000], 'max\_depth': [None, 1, 2, 3], 'min\_samples\_split': [1, 2, 3]} |
| association\_rules | **from** **mlxtend.frequent\_patterns** **import** association\_rules  rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1) |  |
| apriori | association\_rules = apriori(records, min\_support=0.0045, min\_confidence=0.2, min\_lift=3, min\_length=2) |  |
| SARIMAX | model = SARIMAX(data, order=..., seasonal\_order=...) |  |
| Holt Winter | fit1 = ExponentialSmoothing(np.asarray(train['Count']) ,seasonal\_periods=7 ,trend='add', seasonal='add',) |  |