Name: - Aditya Gavankar

Roll no: - J072

Topic: - Machine Learning **Assignment**: - 7

Sci-kit learn API – Grid Search CV

- **Grid Search CV** is the process of performing hyperparameter tuning in order to determine the optimal values for a given model.
- **GridSearchCV** is a function that comes in Scikit-learn's model_selection package.
- This function helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, we can select the best parameters from the listed hyperparameters.
- We pass predefined values for hyperparameters to the GridSearchCV function.
- This is doneby defining a dictionaryin which we mention a particular hyperparameter along with the values it can take.
- GridSearchCV tries all the combinations of the values passed in the dictionary and evaluates the model for each combination using theCross-Validationmethod.
- Hence after using this function, we get accuracy/loss for every combination of hyperparameters and we can choose the one with the best performance.
- GridSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used.
- The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Code:-
*sklearn.model_selection.GridSearchCV(*estimator*,* param_grid*, *,* scoring=None*,* n_jobs=None*,* refit= True*,* cv=None*,* verbose=0*,* pre_dispatch='2*n_jobs'*,* error_score=nan*,* return_train_score=False*)*

Parameter:-

- **estimator(*estimator object).***
  This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a score function, or scoring must be passed.

- **param_grid(*dict or list of dictionaries)***
  Dictionary with parameters names (str) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

- **scoring(*str, callable, list, tuple or dict*), *default=None***
  Strategy to evaluate the performance of the cross-validated model on the test set.
  If scoring represents a single score, one can use:
  - a single string;
  - a callable that returns a single value.

  If scoring represents multiple scores, one can use:
  - a list or tuple of unique strings;
  - a callable returning a dictionary where the keys are the metric names and the values are the metric scores;
  - a dictionary with metric names as keys and callables a values.

- **n_jobs(*int*), *default=None***
  Number of jobs to run in parallel. None means 1 unless in
  a **joblib.parallel_backend** context. -1 means using all processors.

- **refit(*bool, str, or callable*), *default=True***
  Refit an estimator using the best found parameters on the whole dataset.
  For multiple metric evaluation, this needs to be a str denoting the scorer that would be used to find the best parameters for refitting the estimator at the end.
  Where there are considerations other than maximum score in choosing a best estimator, refit can be set to a function which returns the selected best_index_ given cv_results_. In that case, the best_estimator_ and best_params_ will be set according to the returned best_index_ while the best_score_ attribute will not be available.
  The refitted estimator is made available at the best_estimator_ attribute and permits using predict directly on this GridSearchCV instance.
  Also for multiple metric evaluation, the attributes best_index_, best_score_ and best_params_ will only be available if refit is set and all of them will be determined w.r.t this specific scorer.

- **cv*int, cross-validation generator or an iterable, default=None***
  Determines the cross-validation splitting strategy. Possible inputs for cv are:
  - None, to use the default 5-fold cross validation,
  - integer, to specify the number of folds in a (Stratified)KFold,
  - CV splitter,
  - An iterable yielding (train, test) splits as arrays of indices.
  - For integer/None inputs, if the estimator is a classifier and y is either binary or multiclass, **StratifiedKFold** is used. In all other cases, **KFold** is used. These splitters are instantiated with shuffle=False so the splits will be the same across calls.

- **verbose(*int*)**
  Controls the verbosity: the higher, the more messages.
  - >1 : the computation time for each fold and parameter candidate is displayed;
  - >2 : the score is also displayed;
  - >3 : the fold and candidate parameter indexes are also displayed together with the starting time of the computation.

- **pre_dispatch(*int or str*), *default=n_jobs***
  Controls the number of jobs that get dispatched during parallel execution. Reducing this number can be useful to avoid an explosion of memory consumption when more jobs get dispatched than CPUs can process. This parameter can be:
  - None, in which case all the jobs are immediately created and spawned. Use this for lightweight and fast-running jobs, to avoid delays due to on-demand spawning of the jobs
  - An int, giving the exact number of total jobs that are spawned
  - A str, giving an expression as a function of n_jobs, as in '2*n_jobs'

- **error_score(*'raise' or numeric*), *default=np.nan***
  Value to assign to the score if an error occurs in estimator fitting. If set to 'raise', the error is raised. If a numeric value is given, FitFailedWarning is raised. This parameter does not affect the refit step, which will always raise the error.

- **return_train_score(*bool*), *default=False***
  If False, the cv_results_ attribute will not include training scores. Computing training scores is used to get insights on how different parameter settings impact the overfitting/underfitting trade-off. However, computing the scores on the training set can be computationally expensive and is not strictly required to select the parameters that yield the best generalization performance.

Attributes:-

- ***cv_results_dict of numpy (masked) ndarrays***
  A dict with keys as column headers and values as columns, that can be imported into a pandas DataFrame.
  The key 'params' is used to store a list of parameter settings dicts for all the parameter candidates.
  The mean_fit_time, std_fit_time, mean_score_time and std_score_time are all in seconds.
  For multi-metric evaluation, the scores for all the scorers are available in the cv_results_ dict at the keys ending with that scorer's name ('_<scorer_name>') instead of '_score' shown above. ('split0_test_precision', 'mean_train_precision' etc.)
- **best_estimator_*estimator***
  Estimator that was chosen by the search, i.e. estimator which gave highest score (or smallest loss if specified) on the left out data. Not available if refit=False.
  See refit parameter for more information on allowed values.
- **best_score_*float***
  Mean cross-validated score of the best_estimator
  For multi-metric evaluation, this is present only if refit is specified.
  This attribute is not available if refit is a function.
- **best_params_*dict***
  Parameter setting that gave the best results on the hold out data.
  For multi-metric evaluation, this is present only if refit is specified.
- **best_index_*int***
  The index (of the cv_results_ arrays) which corresponds to the best candidate parameter setting.
  The dict at search.cv_results_['params'][search.best_index_] gives the parameter setting for the best model, that gives the highest mean score (search.best_score_).
  For multi-metric evaluation, this is present only if refit is specified.
- **scorer_*function or a dict***
  Scorer function used on the held out data to choose the best parameters for the model.
  For multi-metric evaluation, this attribute holds the validated scoring dict which maps the scorer key to the scorer callable.
- **n_splits_*int***
  The number of cross-validation splits (folds/iterations).
- **refit_time_*float***
  Seconds used for refitting the best model on the whole dataset.
  This is present only if refit is not False.
- **multimetric_*bool***
  Whether or not the scorers compute several metrics.

Methods:-

decision_function(X)	Call decision_function on the estimator with the best found parameters.
fit(*X*, *y=None*, ***, *groups=None*, ***fit_params*)	Run fit with all sets of parameters.
get_params(*deep=True*)	Get parameters for this estimator.
inverse_transform(Xt)	Call inverse_transform on the estimator with the best found params.
predict(X)	Call predict on the estimator with the best found parameters.
predict_log_proba(X)	Call predict_log_proba on the estimator with the best found parameters.
predict_proba(X)	Call predict_proba on the estimator with the best found parameters.
score(*X*, *y=None*)	Returns the score on the given data, if the estimator has been refit.
score_samples(X)	Call score_samples on the estimator with the best found parameters.
set_params(**params)	Set the parameters of this estimator.
transform(X)	Call transform on the estimator with the best found parameters.