**HYPER-TUNING**

*Hyper parameters are more like handles available to control the output or the behavior of the algorithm used for modeling. They can be supplied to algorithms as arguments.*

*Ways of Implementation of Hyper-Tuning:*

*GRID SEARCH CV:*

*Grid Search is one of the most basic hyper parameter technique used and so their implementation is quite simple. All possible permutations of the hyper parameters for a particular model are used to build models. The performance of each model is evaluated and the best performing one is selected. Since GridSearchCV uses each and every combination to build and evaluate the model performance, this method is highly computational expensive.*

*RANDOMISED SEARCH CV:*

*In randomizedsearchcv, instead of providing a discrete set of values to explore on each hyper parameter, we provide a statistical distribution or list of hyper parameters. Values for the different hyper parameters are picked up at random from this distribution*

ClASSIFICATION\_Algorithms:

->Logistic Regression

->KNN

->SVM

->Decision Tree

->Random Forest

->Xgboost

->Adaboost

***LOGISTIC REGRESSION***:

->Parameters:

**penalty*{‘l1’, ‘l2’, ‘elasticnet’, ‘none’}, default=’l2’***

Used to specify the norm used in the penalization. The ‘newton-cg’, ‘sag’ and ‘lbfgs’ solvers support only l2 penalties. ‘elasticnet’ is only supported by the ‘saga’ solver.

**dual*bool, default=False***

Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n\_samples > n\_features.

**tol*float, default=1e-4***

Tolerance for stopping criteria.

**C*float, default=1.0***

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

**fit\_intercept*bool, default=True***

Specifies if a constant (a.k.a. bias or intercept) should be added to the decision function.

**intercept\_scaling*float, default=1***

Useful only when the solver ‘liblinear’ is used and self.fit\_intercept is set to True. In this case, x becomes [x, self.intercept\_scaling], i.e. a “synthetic” feature with constant value equal to intercept\_scaling is appended to the instance vector. The intercept becomes intercept\_scaling \* synthetic\_feature\_weight.

**class\_weight*dict or ‘balanced’, default=None***

Weights associated with classes in the form {class\_label: weight}. If not given, all classes are supposed to have weight one.

The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n\_samples / (n\_classes \* np.bincount(y)).

Note that these weights will be multiplied with sample\_weight (passed through the fit method) if sample\_weight is specified.

**random\_state*int, RandomState instance, default=None***

Used when solver == ‘sag’, ‘saga’ or ‘liblinear’ to shuffle the data. See [Glossary](https://scikit-learn.org/stable/glossary.html#term-random_state) for details.

**solver*{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=’lbfgs’***

Algorithm to use in the optimization problem.

* For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ and ‘saga’ are faster for large ones.
* For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss; ‘liblinear’ is limited to one-versus-rest schemes.
* ‘newton-cg’, ‘lbfgs’, ‘sag’ and ‘saga’ handle L2 or no penalty
* ‘liblinear’ and ‘saga’ also handle L1 penalty
* ‘saga’ also supports ‘elasticnet’ penalty
* ‘liblinear’ does not support setting penalty='none'

Note that ‘sag’ and ‘saga’ fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing.

**max\_iter*int, default=100***

Maximum number of iterations taken for the solvers to converge.

**multi\_class*{‘auto’, ‘ovr’, ‘multinomial’}, default=’auto’***

If the option chosen is ‘ovr’, then a binary problem is fit for each label. For ‘multinomial’ the loss minimised is the multinomial loss fit across the entire probability distribution, *even when the data is binary*. ‘multinomial’ is unavailable when solver=’liblinear’. ‘auto’ selects ‘ovr’ if the data is binary, or if solver=’liblinear’, and otherwise selects ‘multinomial’.

**verbose*int, default=0***

For the liblinear and lbfgs solvers set verbose to any positive number for verbosity.

**warm\_start*bool, default=False***

When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution. Useless for liblinear solver. See [the Glossary](https://scikit-learn.org/stable/glossary.html#term-warm_start)

**n\_jobs*int, default=None***

Number of CPU cores used when parallelizing over classes if multi\_class=’ovr’”. This parameter is ignored when the solver is set to ‘liblinear’ regardless of whether ‘multi\_class’ is specified or not. None means 1 unless in a **[joblib.parallel\_backend](https://joblib.readthedocs.io/en/latest/parallel.html" \l "joblib.parallel_backend" \o "(in joblib v1.1.0.dev0))** context. -1 means using all processors. See [Glossary](https://scikit-learn.org/stable/glossary.html#term-n_jobs)

**l1\_ratio*float, default=None***

The Elastic-Net mixing parameter, with 0 <= l1\_ratio <= 1. Only used if penalty='elasticnet'. Setting l1\_ratio=0 is equivalent to using penalty='l2', while setting l1\_ratio=1 is equivalent to using penalty='l1'. For 0 < l1\_ratio <1, the penalty is a combination of L1 and L2.