EMPLOYEE ATTRITION RATE PREDICTION

I used XgBoost for prediction of Attrition rates for the employees. Data Wrangling and cleaning part included converting categorical values into numerical values and wherever possible, grouping of data was done so as to have less features,

XGBoost's hyperparameters

Well, there are a plethora of tuning parameters for tree-based learners in XGBoost. But the most common ones that one should know are:

- `learning_rate`: step size shrinkage used to prevent overfitting. Range is [0,1]
- `max_depth`: determines how deeply each tree is allowed to grow during any boosting round.
- `subsample`: percentage of samples used per tree. Low value can lead to underfitting.
- `colsample_bytree`: percentage of features used per tree. High value can lead to overfitting.
- `n_estimators`: number of trees you want to build.
- `objective`: determines the loss function to be used like `reg:linear` for regression problems, `reg:logistic` for

classification problems with only decision, `binary:logistic` for classification problems with probability.

XGBoost also supports regularization parameters to penalize models as they become more complex and reduce them to simple (parsimonious) models.

- `gamma`: controls whether a given node will split based on the expected reduction in loss after the split. A higher value leads to fewer splits. Supported only for tree-based learners.
- `alpha`: L1 regularization on leaf weights. A large value leads to more regularization.
- `lambda`: L2 regularization on leaf weights and is smoother than L1 regularization.

The next step is to instantiate an XGBoost regressor object by calling the `XGBRegressor()` class from the XGBoost library with the hyper-parameters passed as arguments. For classification problems, you would have used the `XGBClassifier()` class.

```
xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3,
learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)
```

Fit the regressor to the training set and make predictions on the test set using the familiar `.fit()` and `.predict()` methods.

```
xg_reg.fit(X_train,y_train)

preds = xg_reg.predict(X_test)
```