Module-4

1) Implement k-fold cross validation on a classification dataset and measure the performance using precision recall & F1 score.

K- fold cross validation splits the data into K equal folds, trains the model on (K-1) folds, and tests on the remaining fold, repeating this K times. The final precision, recall, f1- score are the average cross folds. This reduces overfitting because the model is validated on multiple diff. subsets instead of relying on single train-test split giving a more reliable estimate of performance.

Code Snippit:

```
kf = KFold(n_splits=5)
model = SVC()
precision, recalls, f1s = [], [], []
for train_idx, test_idx in kf.split(X):
        X_train, X_test = X[train_idx], X[test_idx]
        Y_train, Y_test = Y[train_idx], Y[test_idx]
        model.fit(X_train, Y_train)
        preds = model.predict(X_test)
```

2) Take a ML model and perform hyper parameter tuning using grid search. Report the best parameters found and analyze their impact on model accuracy.

Grid search tries all possible combinations of pre defined hyperparameters and evaluates each using cross validation. The combination producing the highest accuracy is selected as the best. Hyper parameters are like depth, number of estimators, kernel or c directly control model complexity and tuning them improves the balance b/w underfitting & overfitting.

Code_ snippet:

```
param_grid = {
        "n_estimators" : [50, 100],
        "max_depth" : [5, 10, None]
  }
grid = GridSearchCV(RandomForestClassifier(), paramgrid,
                                                    cv=5)

grid.fit(x, y)
best_params = grid.best_params_
best_score = grid.best_score_
```

3) Create learning and validation curves for a decision tree model on a regression problem. Interpret the curves to diagnose underfitting or overfitting in the model.

Learning curves show how training and validation error change as training size increases. If both scores are low, the model underfits.

Validation curves show performance change w.r.t a single hyper parameter, helping diagnose whether increasing or decreasing complexity improves generalization

Code snippet:

```
train-sizes, train_scores, val-scores = learning_curve(
        DecisionTreeRegressor(), X, y, cv=5)

depths= range(1,20)
trainscores_val, val-scores_val= validationcurve(
        DecisionTreeRegressor(), X, y,
        param_name="max_depth",
        param_range=depths,
        cv =5
    )
```

4) Design an experiment using bootstraping & jacknife methods for estimating the confidence intervals of a model's prediction ~~ka~~ accuracy. Compare the results & discuss.

Bootstraping repeatedly samples the dataset with replacement and recomputes accuracy to estimate a distribution from which confidence intervals are derived.

Jacknife systematically removes one sample at a time and recomputes accuracy, giving another estimate of variability.

Code snippet:

```
boot_stats = []
for i in range(1000):
    Xb, yb = resample(X, Y)
    model.fit(xb, yb)
    boot_stats.append(accuracy_score(y, mode.predict(x)))

jack_stats = []
for i in range(len(x)):
    xi = np.delete(x, i, axis=0)
    yi = np.delete(y, i)
    model.fit(xi, yi)
    jack_stats.append(accuracy_score(y, model.predict(x)))
```

5) Implement a ranking metric for evaluating the results of a search engine model. Explain how this metric captures diff in ranking quality.

Ranking metrics evaluate how well the model orders documents or search results. Average precision (AP) rewards putting relevant items earlier, while NDCG gives higher weight to correct results appearing in top ranks. These metrics judge ranking quality instead of simple classification correctness.

Code snippet:

```
y_true = [[0, 0, 1, 0]]
y_score = [[0.2, 0.1, 0.9, 0.005]]
score = ~~ndcg(~~ ndcg_score(y_true, y_score)
```