Pre Modeling

DATA PREPROCESSING

> Data Preprocessing is the crucial step that decides the difference between a bad, mediocre and awesome output.

Followed Steps in Data Preprocessing

Data Quality Assessment:
Investigate data quality (number of null values, duplicates, etc).

Data Cleaning:
Clean the null values, while making sure your data set does not lose a majority of its rows.

Data Analysis:
Get deeper, and more comprehensive insights

Data Transformation:
Drawing a bar graph of your categorical feature will always help in determining the span of the categories.

You can also combine categories. For instance, if a feature has 30 categories and the top 5 occupy 99.9 percent of the instances, you can combine the rest 25 into one group.

Treat the continuous variables with suspicion. You might encounter the variables as (101,102,103 ..). These types of variables should also be treated as categorical.

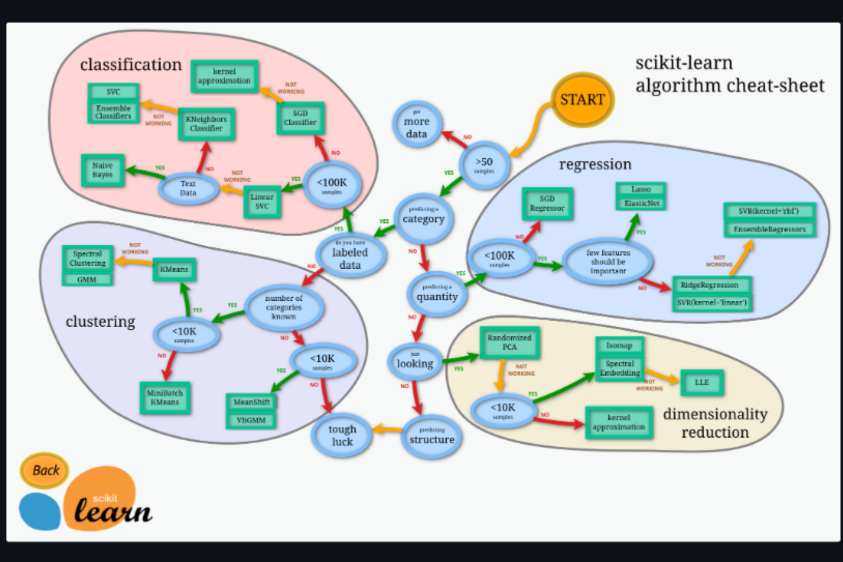
Scaling and Normalization

Scaling is done to Normalize data so that priority is not given to a particular feature.
Role of Scaling is mostly important in algorithms that are distance based

- **Encoding:**

Data Reduction:
i.e. drop all the redundant columns including the ones related to encoding.

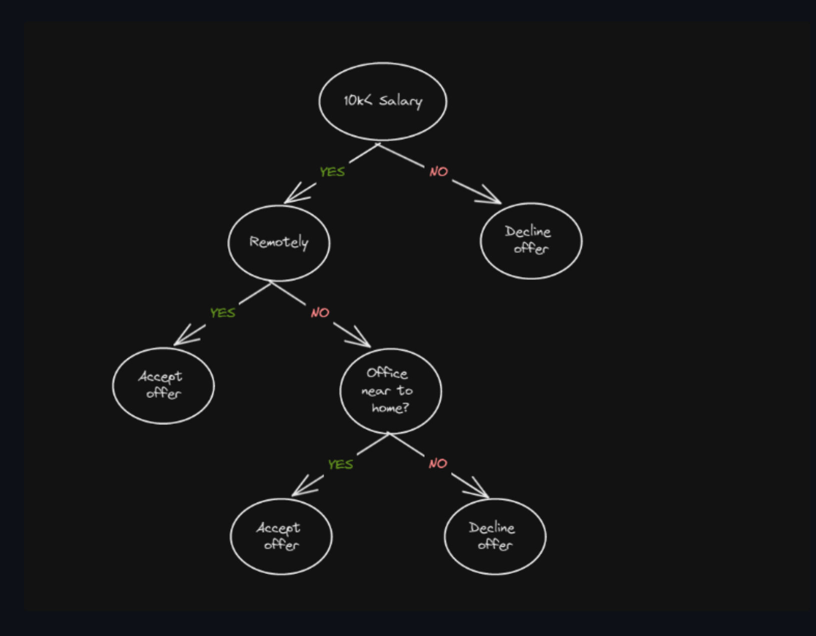
Model Selection



[[ML Models Cheat Sheet]]

Tree-based Models

Decision Tree



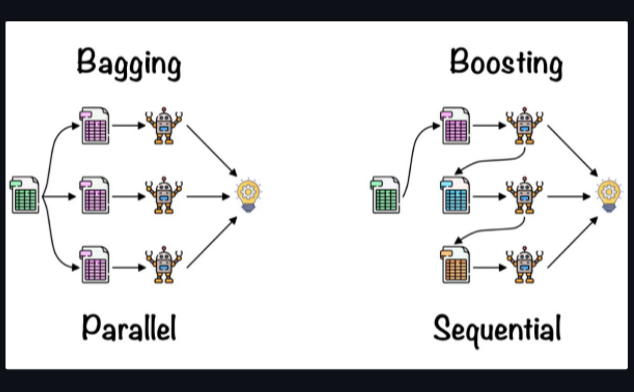
Types of Decision Trees

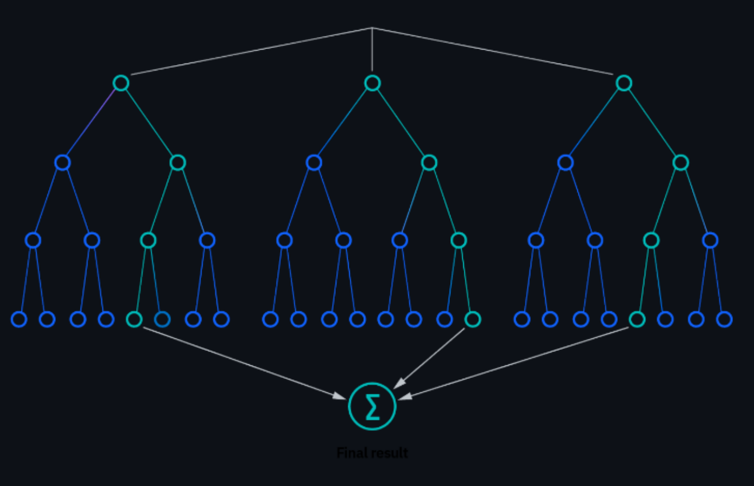
ID3: A shorthand for `Iterative Dichotomiser 3` This algorithm leverages entropy and information gain as metrics to evaluate candidate splits.

C4.5: A later iteration of ID3. It can use information gain or gain ratios to evaluate split points within the decision trees.

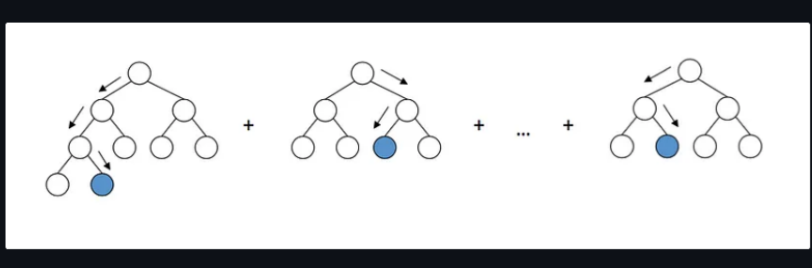
CART: `classification and regression trees` introduced by Leo Breiman. It utilizes Gini impurity to identify the ideal attribute to split on.

Sklearn's decision tree algorithm uses the **CART algorithm**





Random Forest



Gradient Boosting

Gradient-Boosted Decision Trees

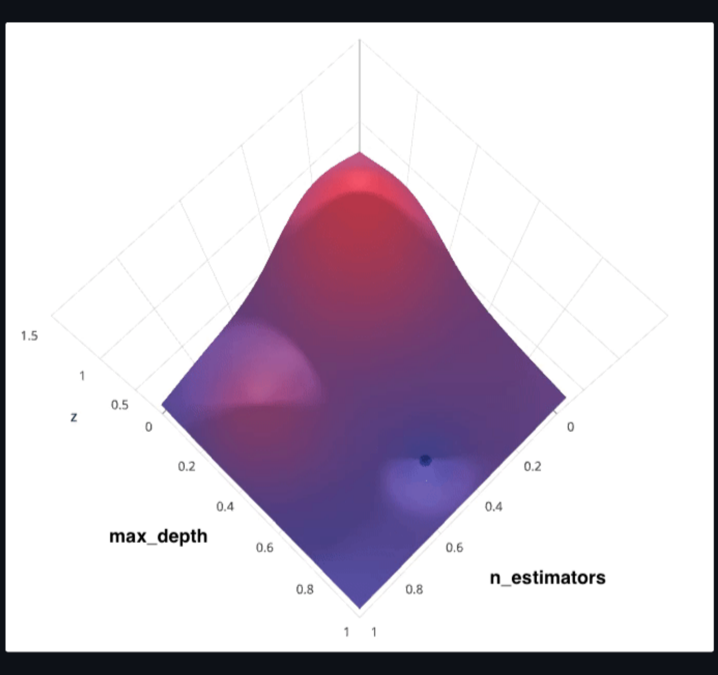
Similar to random forest for classification and regression. Both random forest and GBDT build a model consisting of multiple decision trees. The difference is how they're built and combined.

Random forest bagging minimizes the variance and overfitting, while GBDT boosting reduces the bias and underfitting.

Model Hyperparameter Tuning & Interpretation

Hyperparameter Tuning

Its about how we sample possible model architecture candidates from the space of possible hyperparameter values (`searching` the hyperparameter space for the optimum values.)



Grid Search:

> With this technique, we simply build a model for each possible combination of all of the hyperparameter values provided, evaluating each model, and selecting the architecture which produces the best results.

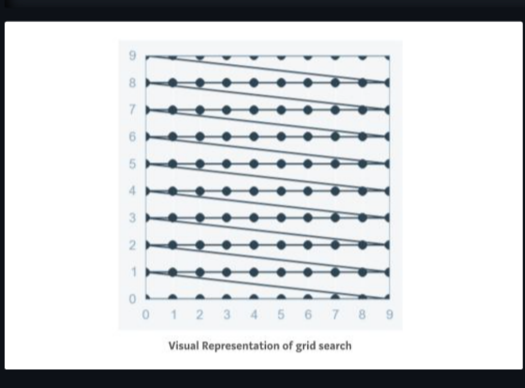
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    'n_estimators': [3, 10, 30],
    'max_features': [2, 4, 6, 8],
]

rf = RandomForestRegressor()

grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='mean_squared_error', return_train_sco

grid_search.fit(X_val, y_val)
```



Visual Representation of grid search

Random Search:

> Not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions. The number of parameter settings that are tried is given by n_iter.

```
from sklearn.model_selection import RandomizedSearchCV
...
rnd_search = RandomizedSearchCV(rf, param_grid, cv=5, scoring='mean_squared_error', return_trai
...
```

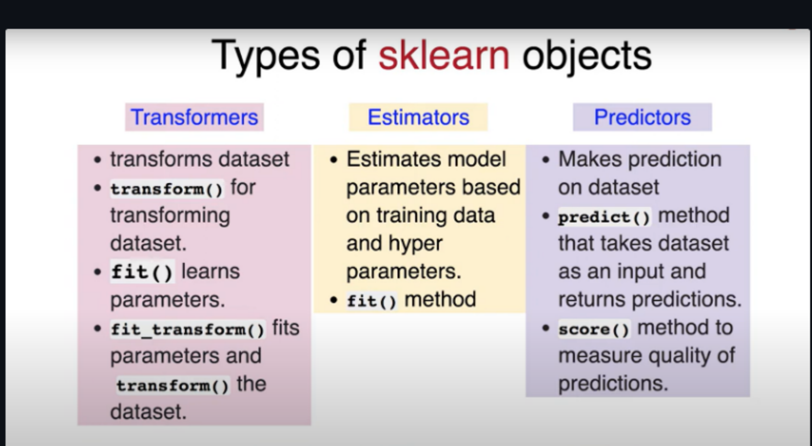
Visual Representation of Random search

Interpretation

Interpret the models to understand which features have the most significant impact.
For `ensemble models` like Random Forest, Gradient Boosting, XGBoost,etc. we can use the feature importances provided by the models to determine the most significant features.
For `linear models` like Linear Regression, we can examine the coefficients of the features.

Scikit-Learn in a Nutshell

Basics of Scikit-Learn API



1. Choose a class of model by importing the appropriate estimator class from Scikit-Learn.
2. Choose model hyperparameters by instantiating this class with desired values.
3. Arrange data into a features matrix and target vector.
4. Fit the model to your data by calling the `fit()` method of the model instance.
5. Apply the Model to new data:
   For supervised learning, often we predict labels for unknown data using the `predict()` method.
   For unsupervised learning, we often transform or infer properties of the data using the `transform()` or `predict()` method.

Choose a class of model

every class of model is represented by a Python class inside a specified module

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, ...
```

Choose model hyperparameters

```
model = GradientBoostingRegressor(learning_rate=0.5, n_estimators=400, max_depth=6,
```

Arrange data into a features matrix and target vector

```
y = df[target]
X = df.drop(columns=[target])
```

Fit the model to your data

```
model.fit(X_train, y_train)          # Notice no returned value/object
```

Apply the Model to new data

```
predictions = model.predict(new_data)
```

Scikit-Learn Pipelines

Pipelines: It's like Lego (putting it all together)

Fit all the transformers one after the other and transform the data. Finally, fit the transformed data using the final estimator.

```
from sklearn.pipeline import Pipeline
clf = Pipeline([('pca', decomposition.PCA(n_components=150)),
        ('svm', svm.LinearSVC(C=1.0))])

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

REFERENCES