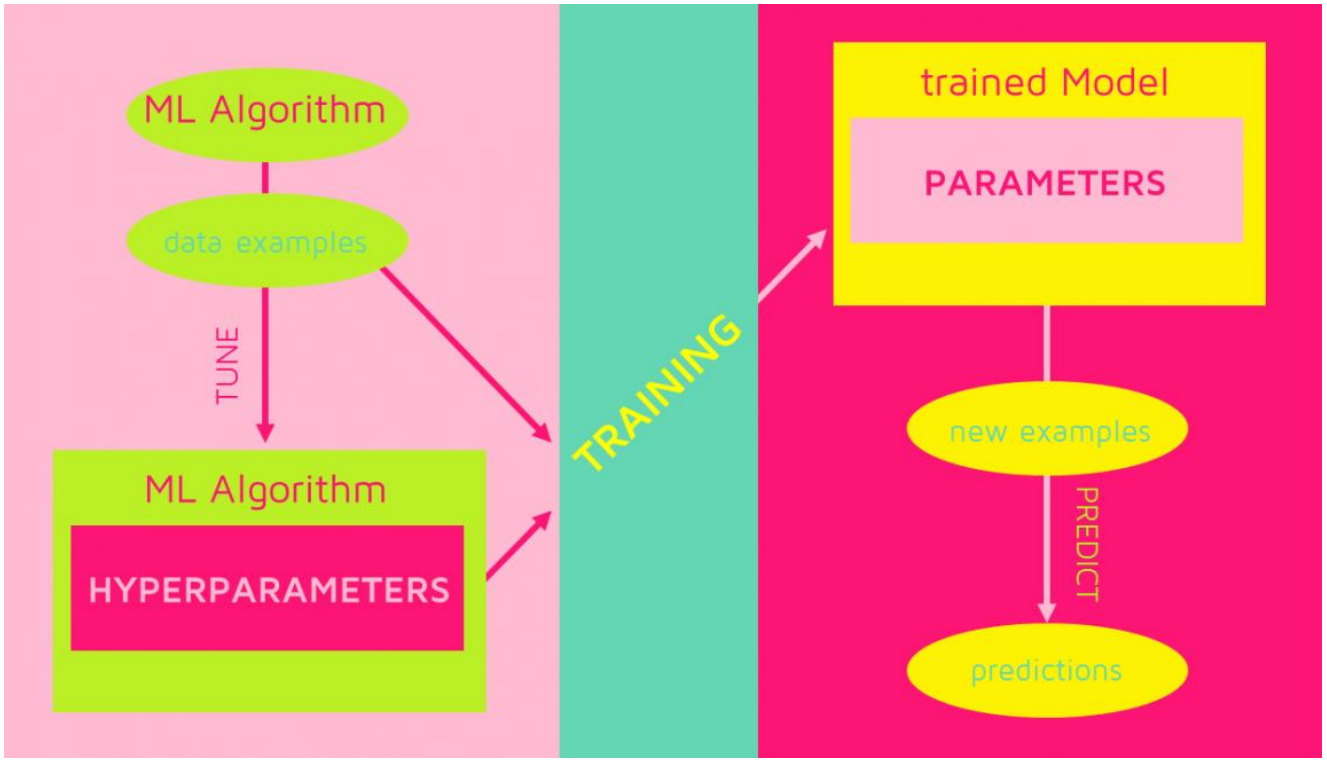
Hyperparameter Tuning for ML models

Model hyperparameters

- Hyperparameters are the parameters that are explicitly defined by the user to control the learning process
- There are external to the model, and their values cannot be changed during the training process.
- Examples of hyperparameters:
  - The depth of a decision tree
  - Number of trees in a Random Forest
  - The K for KNN
  - The penalty in Logistic Regression Classifier i.e. L1 or L2 regularization

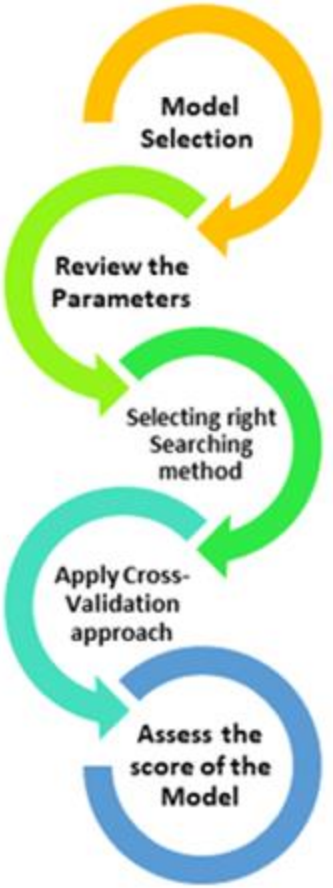
Model parameters vs model hyperparameters



- Choosing appropriate hyperparameters is an essential task when applying ML. Hyperparameters can affect the speed and also the accuracy of the final model.

Hyperparameter tuning

- Hyperparameter tuning (or hyperparameter optimization) is the process of determining the right combination of hyperparameters that maximizes the model performance.
- It is an important step in any Machine Learning project since it leads to optimal results for a model.
- It is a process that includes :
  - Select the right type of model.
  - Review the list of parameters of the model and build the hyperparameter space
  - Define a method for searching the hyperparameter space
  - Applying the cross-validation scheme approach
  - Assess the model score to evaluate the model



Designed by Author - Shanthababbu

Hyperparameter tuning approaches

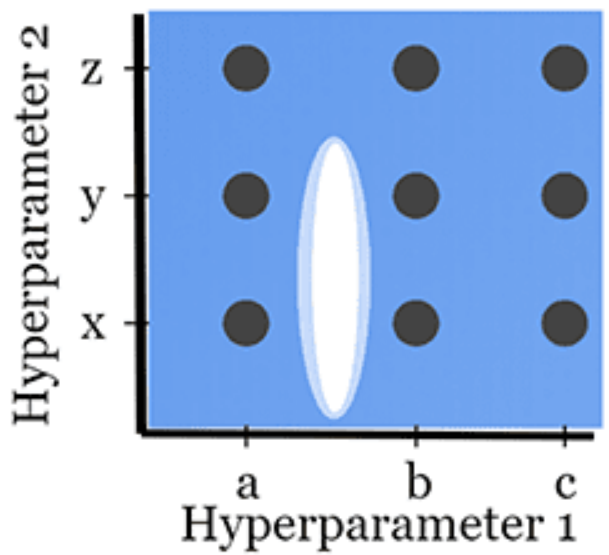
- Some of the common methods are:
  - **Manual search**: test different hyper-parameter values manually and select the one that performs best
  - **Grid search**: performs an exhaustive search by evaluating all hyperparameters' combinations.
  - **Random search**: instead of an exhaustive search, random combinations of hyperparameters are tested.

Random Search vs Grid Search

Grid Search

Pseudocode

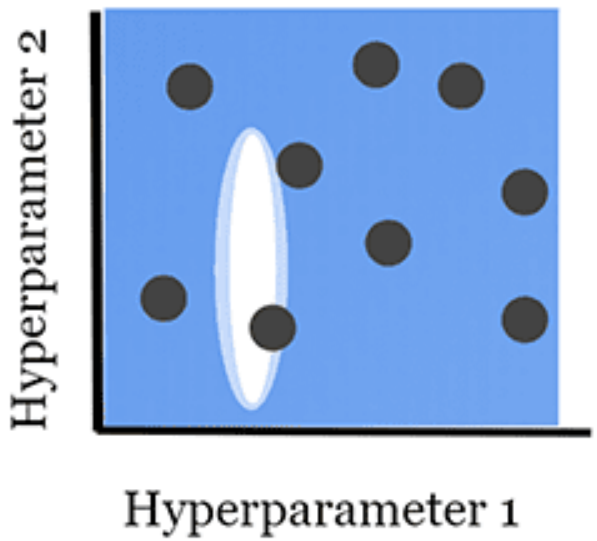
```
Hyperparameter_One = [a, b, c]
Hyperparameter_Two = [x, y, z]
```



Random Search

Pseudocode

```
Hyperparameter_One = random.num(range)
Hyperparameter_Two = random.num(range)
```



Random Search vs Grid Search

- Grid search can be extremely time-consuming: Use grid search if you already have a range of known hyperparameter values that will perform well. Make sure to keep your parameter space small.
- Random search is faster than grid search : Use random search on a broad range of values if you don't already have an idea of the parameters that will perform well on your model. It should always be used when you have a large parameter space.
- It is a good idea to use both random search and grid search to get the best possible results.
  - use random search first with a large parameter space since it is faster. Then, use the best hyperparameters found by random search to narrow down the parameter grid, and feed a smaller range of values to grid search.

Hyperparameter tuning in scikit-learn

- Scikit-learn has implementations for grid search and random search. For both of those methods, scikit-learn trains and evaluates a model in a k fold cross-validation setting over various parameter choices and returns the best model.
- Package sklearn.model_selection
  - RandomizedSearchCV for random search
  - GridSearchCV for grid search:

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

GridSearchCV

- Common parameters

1.estimator: Pass the model instance for which you want to check the hyperparameters.

2.params_grid: the dictionary object that holds the hyperparameters you want to try

3.scoring: evaluation metric that you want to use, you can simply pass a valid string/ object of evaluation metric

4.cv: number of cross-validation you have to try for each selected set of hyperparameters

5.verbose: you can set it to 1 to get the detailed print out while you fit the data to GridSearchCV

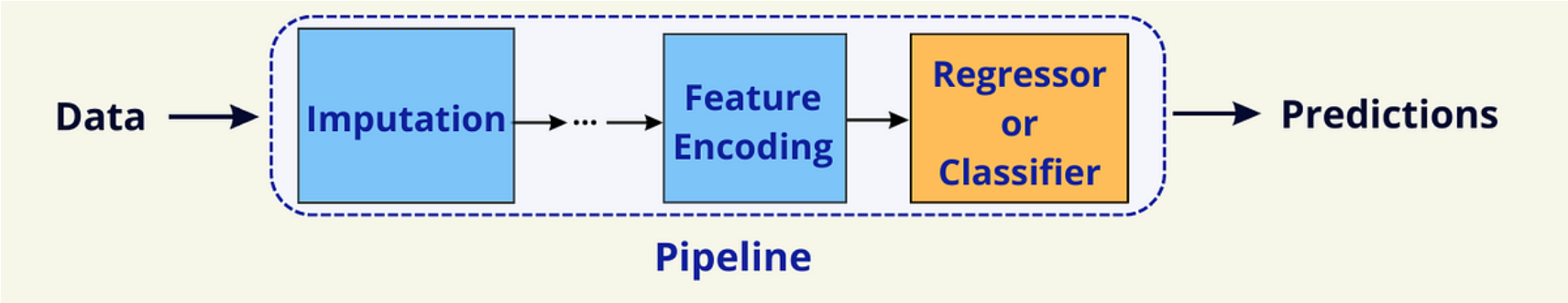
6.n_jobs: number of processes you wish to run in parallel for this task if it -1 it will use all available processors.

GridSearchCV

- GridSearchCV implements a fit method
  - fit, is invoked on the instance of GridSearchCV with training data (X_train) and related label (y_train).
- Once the GridSearchCV estimator is fit, the following attributes are used to get vital information:
  - best_score_: Gives the score of the best model which can be created using most optimal combination of hyper parameters
  - best_params_: Gives the most optimal hyper parameters which can be used to get the best model
  - best_estimator_: Gives the best model built using the most optimal hyperparameters

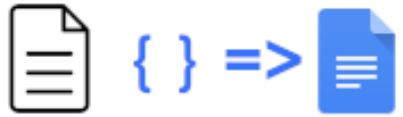
Pipelines

- A machine learning pipeline can be created by putting together a sequence of steps involved in training a machine learning model.
- It can be used to automate a machine learning workflow.
- The pipeline can involve pre-processing, feature selection, classification/regression, and post-processing.



Scikit-Learn Pipeline

Data Pipelines & ML Pipelines



Transformer ⇨ **Estimator**

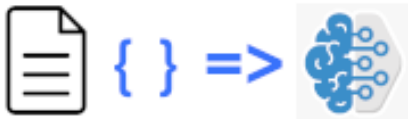
Function that takes data and fit & transforms them into augmented data or feature

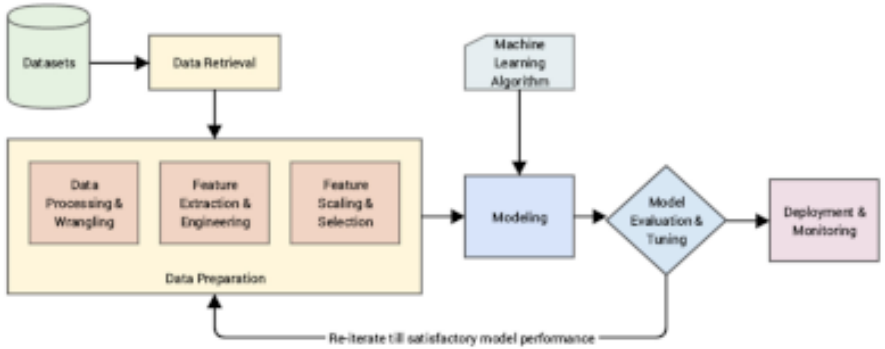
Function that takes data as input and fit the data and produces a model we can use to predict

StandardScaler,TfidfVectorizer

LogisticRegression,KNN

Data To Data ⇨ **Data To Model**





Jesus Saves @JCharisTech

Pipelines in Scikit-learn

- Scikit-learn works around the concept of transformers and estimators
- Transformer: entities capable of transforming data.
  - E.g., StandardScaler, MinMaxScaler…
- After transformation, data needs to be passed to estimator for training or prediction
  - E.g. LinearRegression, DecisionTree…
- Pipeline : Sequentially apply a list of transforms and a final estimator. Intermediate steps of pipeline must implement fit and transform methods and the final estimator only needs to implement fit.
- Pipelines are implemented using Pipeline Class, chaining the components together

```
class sklearn.pipeline.Pipeline(steps, *, memory=None, verbose=False)
```

Pipelines in Scikit-learn

- Example :

```
pipelineRFC = make_pipeline(StandardScaler(),
RandomForestClassifier(criterion='gini', random_state=1))
#
# Create the parameter grid
#
param_grid_rfc = [{
    'randomforestclassifier__max_depth':[2, 3, 4],
    'randomforestclassifier__max_features':[2, 3, 4, 5, 6]
}]
#
# Create an instance of GridSearch Cross-validation estimator
#
gsRFC = GridSearchCV(estimator=pipelineRFC,
                     param_grid = param_grid_rfc,
                     scoring='accuracy',
                     cv=10,
                     refit=True,
                     n_jobs=1)
#
# Train the RandomForestClassifier
#
gsRFC = gsRFC.fit(X_train, y_train)
#
# Print the training score of the best model
#
print(gsRFC.best_score_)
#
# Print the model parameters of the best model
#
print(gsRFC.best_params_)
#
# Print the test score of the best model
#
clfRFC = gsRFC.best_estimator_
print('Test accuracy: %.3f' % clfRFC.score(X_test, y_test))
```