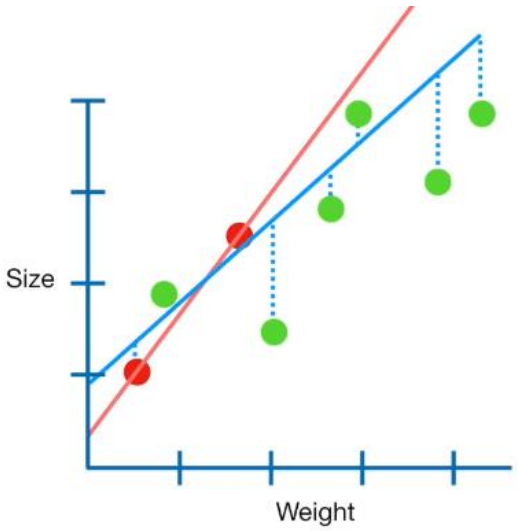
Featurization, Model Selection and Tuning - Handbook

Regularization

Ridge regression



Here, we have weight and size measurements from a bunch of mice. Since the data looks relatively linear, we use linear regression/least squares.

Let's assume we get the equation: size=0.9+0.75*weight (Blue line)

When we have a lot of measurements we can be fairly confident that the least squares line accurately reflects the relationship between size and weight but what if we had only two measurements?

The regression line will overlap the two data points and the minimum sum of squares residuals will be zero. (Red Line)

New line equation maybe: size=0.4+1.3*weight. (Red Line)

The sum of squared residuals for just the two data points, the training data, is very small, in this case zero. But if we try to find the sum of squared residuals for the green datapoints, testing data, it will be large. That means the red line has a high variance. We would say that the red line is overfit to the training data.

The main idea behind ridge regression is to find a line that does not fit the training data as well. We introduce a small amount of bias to the line. So, by starting with a slightly worse fit, ridge regression can provide better long term predictions.

When least squares determines values for the parameters in the equation,

Size= y axis intercept + slope x weight, it maximizes the sum of squared residuals.

In contrast, when ridge determines the values for the parameters in that equation, it minimizes:

the sum of squared residuals + lambda * (slope)^2.

Squared slope here adds penalty to the traditional least squares method and lambda determines how severe that penalty is.

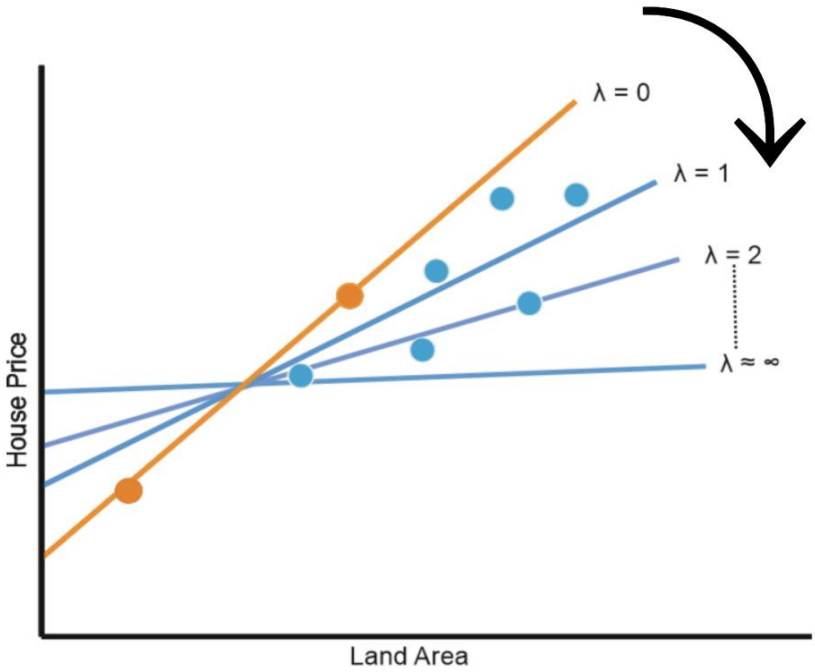
So, without small amount of bias that the penalty creates, the least squares fit has a large amount of variance. In contrast, the ridge regression line which has a small amount of bias due to the penalty, has less variance.

To summarize, when sample sizes are relatively small, the ridge regression can improve predictions by making them less sensitive to the Training Data. This is done by adding ridge regression penalty to sum of squared residuals(which must be minimized) -> SSR+ lambda * slope squared.

The ridge penalty is the lambda times sum of all squared parameters, except for the y intercept. The lambda is determined by using cross validation. Lastly, even if there isn't enough data to find the least squared parameter estimates, ridge regression can still find a solution with cross validation and the ridge regression penalty alone.

Lasso regression.

It is very similar to ridge with some notable differences.



The main idea for ridge, was that starting with a slightly worse fit, ridge provided better long term predictions using bias variance tradeoff.

For lasso, instead of squared slope, we take the absolute value.

Lasso: SSR+ lambda x |slope|

Just like ridge, here lambda can be any value between 0 and positive infinity and this is determined using cross validation.

Just like ridge, lasso regression line results in a little bit of bias but less variance then least squares.

Just like ridge, lasso regression penalty contains all of the estimated parameters except for the y intercept.

Now, an important difference between the two is that ridge and lasso regression do not shrink the parameters equally.

When lambda=0, ridge and lasso regression line is same as least square line.

As lambda increases in value slope gets smaller until slope=0.

The big difference here is that ridge regression can only shrink slope asymptotically close to zero while lasso regression can shrink the slop all the way to zero.

So, lasso regression can exclude useless variables from equations, it is a little better than ridge regression at reducing the variance in models that contain a lot of useless variables. In contrast, ridge regression tends to do better when most variables are useful.

Up Sampling:

It is the process of randomly duplicating observations from the minority class

1- First, we will separate observations from each class into different Data Frames.
2- Next, we will resample the minority class with replacement, setting the number of samples to match that of the majority class.
3- Finally, we'll combine the up-sampled minority class Data Frame with the original majority class Data Frame.

Down sampling:

It involves randomly removing observations from the majority class to prevent the model getting biased towards the majority class.

1. Separate observations from each class into different Data Frames.
2. Resample the majority class without replacement, setting the a number of samples to match that of the minority class.
3. Finally, combine the down-sampled majority class Data Frame with the original minority class Data Frame.

KFold cross validation:

Image source: Link

Steps:

1. The dataset is split into training and test dataset.
2. The training dataset is then split into K-folds where (K-1) fold is used for training and 1 fold is used for validation
3. The model is executed with training data (K-1 folds) and then evaluated with validation data as 1 fold. The performance of the model is recorded.
4. The above steps (step 3, step 4, and step 5) is repeated until each of the k-fold got used for validation purpose. This is why it is called k-fold cross-validation.
5. Finally, the mean and standard deviation of the model performance is computed by taking all of the model scores calculated in step 5 for each of the K models.
6. Hyper parameter tuning is done by repeating steps 1-5

7. Finally, the hyperparameters which result in the most optimal mean and the standard value of model scores get selected.
8. The model is then trained using the training data set (step 2) and the model performance is computed on the test data set (step 1).

Keep in mind that: for a relatively small K, the number of folds increases while a large K value increases the runtime of the cross validation algorithm. If training folds are smaller then we end up with models that result in high variance for model estimates. For very small data sets, the leave-one-out cross-validation (LOOCV) technique is used. In cases of unequal class proportions, stratified k-fold cross-validation in suggested to achieve better model estimates.

CODE:

```
# evaluate a logistic regression model using k-fold cross-validation
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# create dataset
X, y = make_classification(n_samples=100, n_features=20, n_informative=15, n_redundant=5, random_state=1)
# prepare the cross-validation procedure
cv = KFold(n_splits=10, random_state=1, shuffle=True)
# create model
model = LogisticRegression()
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

LOOCV:

Leave-one-out cross-validation, or LOOCV, is a configuration of k-fold cross-validation where *k* is set to the number of examples in the dataset.

LOOCV is an extreme version of k-fold cross-validation that has the maximum computational cost. It requires one model to be created and evaluated for each example in the training dataset. The benefit of so many fit and evaluated models is a more robust estimate of model performance as each row of data is given an opportunity to represent the entirety of the test dataset.

CODE:

```
# loocv to manually evaluate the performance of a random forest classifier
from sklearn.datasets import make_blobs
from sklearn.model_selection import LeaveOneOut
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# create dataset
X, y = make_blobs(n_samples=100, random_state=1)
# create loocv procedure
cv = LeaveOneOut()
# enumerate splits
y_true, y_pred = list(), list()
for train_ix, test_ix in cv.split(X):
# split data
   X_train, X_test = X[train_ix, :], X[test_ix, :]
   y_train, y_test = y[train_ix], y[test_ix]
# fit model
  model = RandomForestClassifier(random_state=1)
  model.fit(X_train, y_train)
# evaluate model
  yhat = model.predict(X_test)
# store
  y_true.append(y_test[0])
  y_pred.append(yhat[0])
# calculate accuracy
acc = accuracy_score(y_true, y_pred)
print('Accuracy: %.3f' % acc)
```

Data Leakage:

Data leakage would be if we included the true label of a data instance as a feature in the model or having test data accidentally included in the training data which leads to over fitting. Leakage can cause your system to learn a sub optimal model that does much worse in actual deployment than a model developed in a leak free setting.

Bootstrapping:

A method of inferring results for a population from results found on a collection of smaller random samples of that population, using replacement during the sampling process.

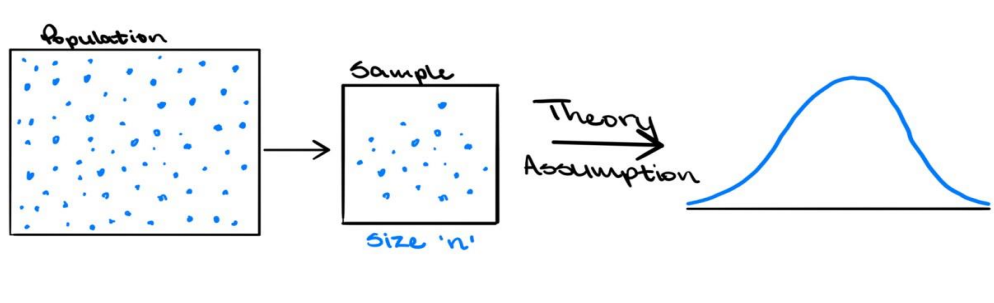


Image source: Link

Out of bag method:

It is used to evaluate the classifier.

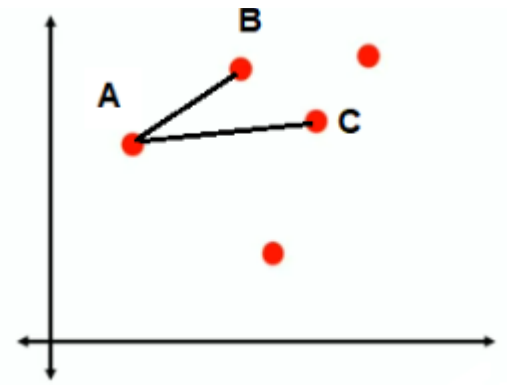
We take a given number of samples from the original dataset and replicate them to get a new dataset. For example, if our original dataset has 10 datapoints. While sampling, we take datapoints- 1,3,2,2,4,5,1,7,3,4,1. We now have 10 elements (bootstrapped data) for training data. The datapoints that were not used- 6,8,9,10 becomes the OOB data. We can then use this to evaluate the classifier that was trained with the bootstrapped data.

Synthetic minority oversampling technique (SMOTE):

For a model to learn the decision boundary, we need adequate amount of each class in the dataset. When one of the class has too few instances, this becomes a problem. We can overcome this by replicate examples from the minority class in the training dataset or generating new synthetic datapoints from the minority class. This augmentation of data by creating synthetic points is called SMOTE.

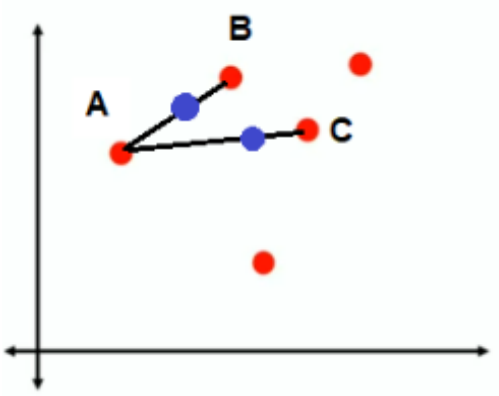
SMOTE works by selecting K datapoints that are close to a randomly selected minority class instance. Then a neighbor is randomly selected and a synthetic example is created at a randomly selected point between the two examples in feature space

Minority class before SMOTE:



Here, B, C are the nearest neighbors of A. We create synthetic data at some random randomly selected point in the line segment between A&B and A&C. We can also have two synthetic data in the line segment between A&B.

After generating new datapoints for A:



Similarly, we can do this for all minority datapoints:

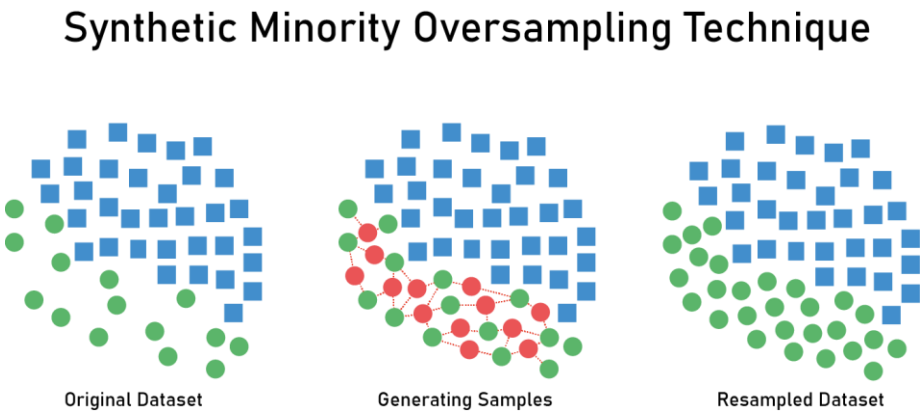


Image source: Link

CODE:

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(sampling_strategy=0.5,k_neighbors=5,random_state = 100)
X_sm, y_sm = sm.fit_sample(X, Y)
```

CENTRAL LIMIT THEOREM (CLT):

The central limit theorem states that the distribution of sample means approximates a normal distribution as the sample size gets larger, regardless of the population's distribution.

Usually a sample size of 30 or greater is used for CLT to hold true.

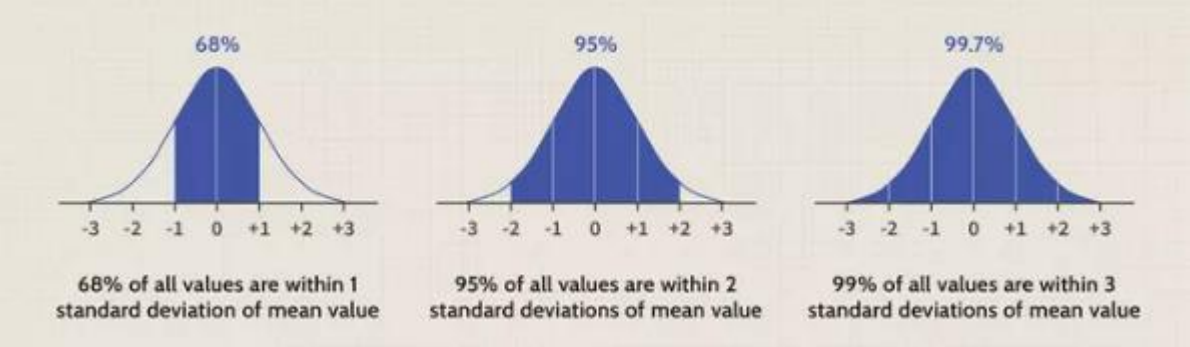


Image source: Link

Average of the sample means and standard deviations will almost be equal to the population mean and standard deviation as the sample size grows.

Confidence intervals are a way of quantifying the uncertainty of an estimate.

A smaller confidence interval means the model estimates are more precise while a larger confidence interval means the model estimates are less precise.

We can perform cross validation and get the mean and standard deviation of the average estimate of the model. Then we do the below:

First, we must choose a significance level for the confidence level, such as 95% CI, represented as 5.0% significance level (e.g. 100 – 95).

CODE:

```
When we consider  95% confidence interval: (100 - alpha)
alpha = 5.0
To calculate the value at the 2.5th percentile of the distribution:
calculate lower percentile (e.g. 2.5)
lower_p = alpha / 2.0
# retrieve observation at lower percentile
lower = max(0.0, percentile(scores, lower_p))

To calculate value at the 97.5th percentile of the distribution:
# calculate upper percentile (e.g. 97.5)
upper_p = (100 - alpha) + (alpha / 2.0)
# retrieve observation at upper percentile
upper = min(1.0, percentile(scores, upper_p))
```

Hence, we can say that there is a 95% likelihood that the range lower to upper covers the true statistic mean.

So, if we repeated this experiment over and over, each time drawing a new sample, we would find that for approximately 95% of these experiments, the calculated interval would contain the true model estimates. For this reason, we call this interval the 95% confidence interval estimate.

Model performance measures:

Confusion Matrix:

Summarizing how model performed on the testing data.

Correct predictions: TP, TN

Wrong predictions: FN, FP

Problem statement example: Build a model to find if a person had heart disease or not.

True positive (TP): The patients that had heart disease that were correctly identified by the algorithm.

True negative (TN): The patient that did not have heart disease that were correctly identified by the algorithm.

False negative (FN): when a patient has a heart disease but the algorithm said they didn't.

False positive (FP): Are patients that do not have heart disease but the algorithm says they do.

Sensitivity: Says what percentage of patients with heart disease were correctly identified.

Sensitivity= True positive / (True positive + False negative)

Specificity: Tells us what percentage of patients without heart disease were correctly identified.

Specificity: TN/ TN+FP

If correctly identifying positives is more important then we should choose a method with higher sensitivity.

If correctly identifying negatives is more important then we choose a method with higher specificity.

Precision/ predicted positive values = TP/ TP+ FP = TP/ (predicted positive)

False positive rate/ false alarm= FP/ actual negatives = FP/ (FP+ TN) = 1- specificity

False negative rate = FN/ actual positives = FN/ TP+FN = 1- sensitivity.

F1 score: It is the harmonic mean of precision and recall.

F1 score= TP/ (TP + ½*(FP+FN)) = 2 * [(precision*recall)/ (precision + recall)]

F1 does not consider True negatives because precision and recall do not have it in their calculation either.

Receiver operating curve and area under the curve (ROC and AUC)

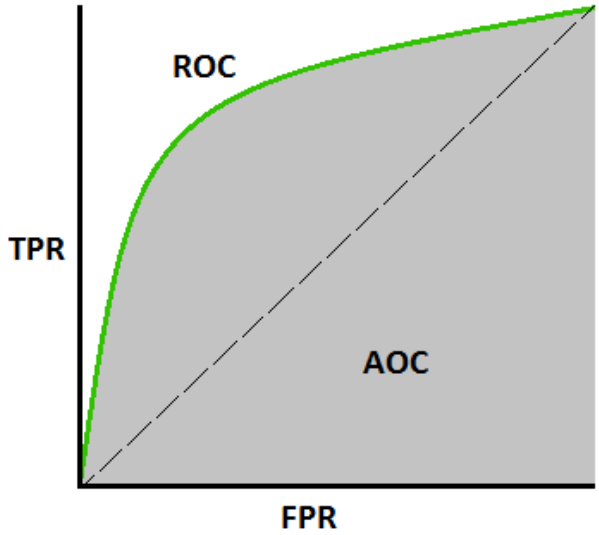


Image source: Link

Taking an example of logistic regression, one way to classify a person is to set a threshold at 0.5 and classify all persons with probabilities of having a heart disease > 0.5 as heart patients.

To evaluate the effectiveness of the model with classification threshold set to 0.5, we can test it with patient that we know has a heart disease or not.

The Roc graph has a Y axis that contains information of true positive rate or sensitivity. (What portion of heart patients were correctly classified)

And the X axis shows the false positive rate (1- specificity) - the portion of not heart patients that were incorrectly classified

Any point on the diagonal line means that the portion of the correctly classified heart patients is the same as the portion of incorrectly classified samples that are not heart patients.

A point to the left of the diagonal means that the proportion of correctly classified samples that were heart patients (True positives) is greater than the proportion of the samples that were incorrectly classified as heart patients (False positives)

The ROC graph summarizes all of the confusion matrices that each threshold produced.

AUC makes it easier to compare one ROC with another. The ROC with a greater AUC is the one we need to choose.

So, to summarize, ROC helps in identifying the best threshold for decision making and AUC helps us decide which model(ROC) is better.

The better performing model will have a higher area under the curve with the curve closer to the upper left corner.

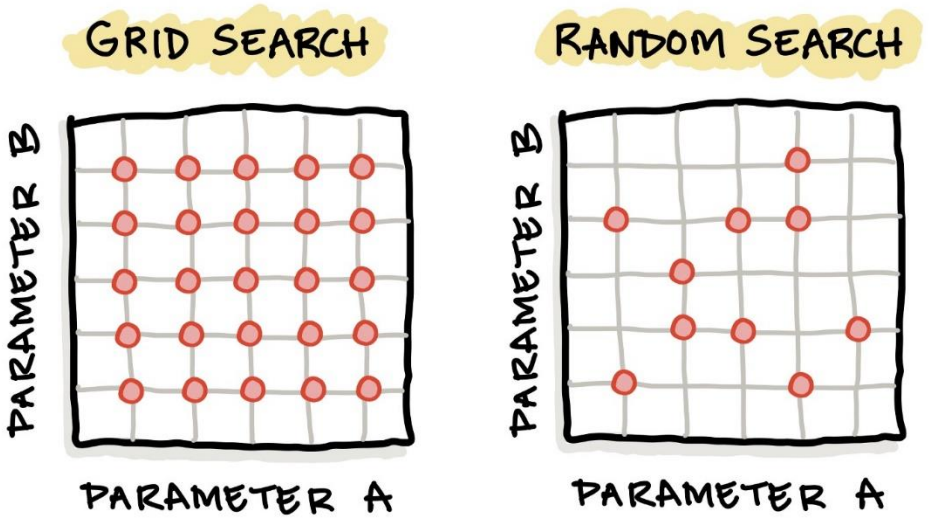


Image source: **Link**

Grid Search CV

GridSearchCV is the process of performing hyperparameter tuning in order to determine the optimal values for a given model. As mentioned above, the performance of a model significantly depends on the value of hyperparameters. Note that there is no way to know in advance the best values for hyperparameters so ideally, we need to try all possible values to know the optimal values. Doing this manually could take a considerable amount of time and resources and thus we use GridSearchCV to automate the tuning of hyperparameters. This function helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, we can select the best parameters from the listed hyperparameters.

GridSearchCV tries all the combinations of the values passed in the dictionary and evaluates the model for each combination using the Cross-Validation method. Hence after using this function we get accuracy/loss for every combination of hyperparameters and we can choose the one with the best performance.

The best values for hyperparameters for this SVM model is not always the best parameter for all models. The obtained hyperparameters may be the best for the dataset we are working on. But for any other dataset, the model can have different optimal values for hyperparameters that may improve its performance.

RandomSearch CV:

We used the sci-kit learn (sklearn) library when implementing grid search, particularly GridSearchCV. From the same library, we shall use RandomizedSearchCV. Similar to GridSearchCV, it is meant to find the best parameters to improve a given model.

A key difference is that it does not test all parameters. Instead, the search is done at random. As we shall note later, a practical difference between the two is that RandomizedSearchCV allows us to specify the number of parameter values we seek to test. This is through the use of n_iter.

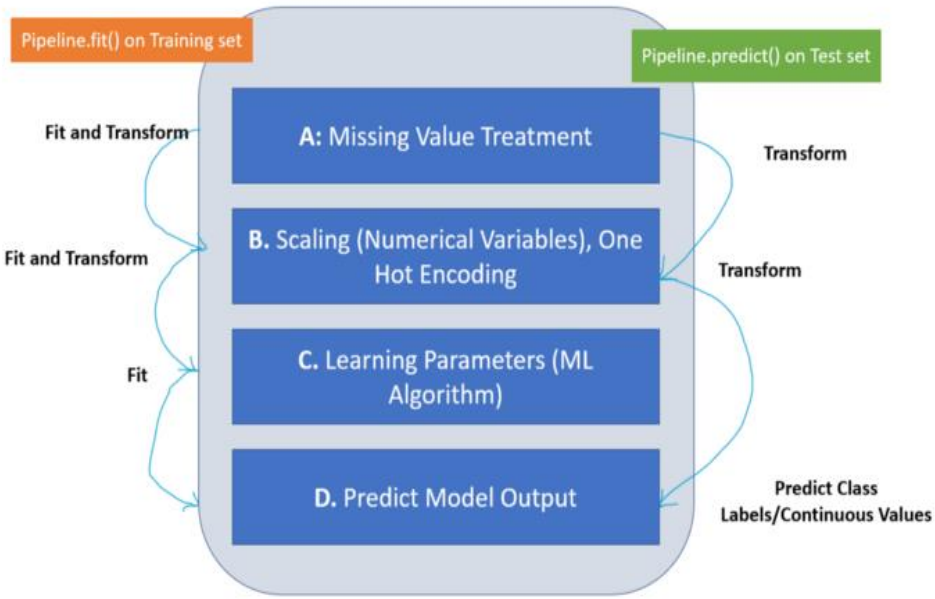
Grid search would guarantee accuracy by exhausting all possible combinations. Even though random search may not be as accurate as grid search, here we get to choose the number of combinations to try out. The

random search model takes a much shorter time to train and find the optimized parameters than grid search CV which is computationally more efficient.

Reference: https://www.mygreatlearning.com/blog/gridsearchcv/

Pipeline in Machine Learning

- Machine learning (ML) pipelines consist of several steps to train a model. Machine learning pipelines are iterative as every step is repeated to continuously improve the accuracy of the model and achieve a successful algorithm.
- The main objective of having a proper pipeline for any ML model is to exercise control over it. A well-organized pipeline makes the implementation more flexible.
- A pipeline consists of a sequence of components which are a compilation of computations. Data is sent through these components and is manipulated with the help of computation.
- Pipelines are not one-way flows. They are cyclic in nature and enables iteration to improve the scores of the machine learning algorithms and make the model scalable.
- ML Pipeline is defined as a collection of preprocessing steps and a model. This means when raw data is passed to the ML Pipeline, it preprocesses the data to the right format, scores the data using the model and pops out a prediction score.



How do Pipelines work?

Pipelines consist of a series of fit and transform operations that vary between training and test data. Usually, fit and transform together or fit function is used on the training set; transform alone is used on the test set. The fit and transform operation first identify important information about the data distribution and then transforms the data based on this learned information. The fit() learns the Median Value of the corresponding attribute by studying the data distribution, whereas the transform() replaces the missing value with the Median learned from

the data. An, e.g., involves, let's say, we are using a SimpleImputer(), a python package for missing value treatment, to replace missing values with Median values.

Test data, on the other hand, uses only the transform operation. This is mainly done to avoid any data leak when building the model. The idea is to perform pre-processing on the test data based on information learned from the training set than cleaning the entire data as a whole. Any missing information in the incoming data stream needs to be handled before it can be fed into the prediction model. This is important, especially when you put the model to production.

Source: ML- pipeline

Link for reference

https://www.kaggle.com/code/pouryaayria/a-complete-ml-pipeline-tutorial-acu-86/notebook

********** **HAPPY LEARNING!** **********