M02: End-to-End ML Project

Codes	
Created	@May 23, 2021
Related to Neuroscience Notes (Property)	
Source	Hand-on ML BOOK
Tags	中 Machine Learning
URL	

Contents

Main Steps of an End-to-End Machine Learning Project

- Look at the Bigger Picture.
- Get the Data.
- Discover and visualize the data for machine learning algorithms.
- Select a model and train it.
- Fine-tune the model.
- Present your Solution.
- Launch, monitor, and maintain your system.

Pipelines

A sequence of data processing components is called a data pipeline. Pipelines are very common in ML systems, since there is a lot of data to manipulate and many data transformations to apply.

Components typically run asynchronously. Each component pulls in a large amount of data, process it, and spits out the result in another data store. Then after some time another component pulls in the data and spits out its output.

Each component is self-contained: the interface between components is simply the data store. This makes this system easy to grasp and modify, and different parties can focus on different components. Moreover, if a component breaks-down, the downstream components can often continue to run normally by just using the last output from the broken component making the architecture quite robust. On other hand, a broken component can go unnoticed for some time if proper monitoring

is not implemented. The data gets stale and the overall system's performance drops.

Performance Measure

Root Mean Square Error (RMSE)

$$RMSE(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2}$$

NOTATIONS:

- ***m*** is the number of instances in the dataset you are measuring the RMSE on.
- $\mathbf{x}^{(i)}$ is a vector of all feature values (excluding the label) of the i^{th} instance in the dataset, and $y^{(i)}$ is its label(the desired output value for that instance).
- $\mathbf{X}$ is a matrix containing all the features(excluding labels) of all instances in the dataset. There is one row per instance, and the i^{th} row is equal to the transpose of $\mathbf{x}^{(i)}$, noted $(\mathbf{x}^{(i)})^T$.
- h is your system's prediction function, also called a hypothesis. When your system is given an instance's feature $\mathbf{x}^{(i)}$, it outputs a predicted value $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$ for that instance.
- $RMSE(\mathbf{X}, h)$ is the cost function measured on the set of examples using the hypothesis h.

👉 RMSE is generally preferred performance measure for regression tasks.

Mean Absolute Error(MAE)

$$MAE(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right|$$

DISTANCE MEASURES OR NORMS:

Distance metric uses distance function which provides a relationship metric between each element in the dataset.

▼ Resources

Different Types of Distance Metrics used in Machine Learning

Many Supervised and Unsupervised machine learning models such as K-NN and K-Means depend upon the distance between two data points to predict the output. Therefore, the metric we use to compute

https://medium.com/@kunal_gohrani/different-types-of-distance-metrics-used-in-machine-learning-e9928c5e26c7

Types of Distances in Machine Learning

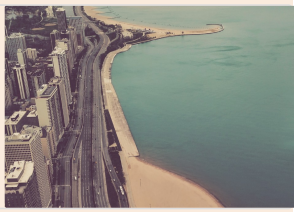
Ever wondered how the Machine Learning algorithms calculate the distance? Or for once, did it come to your mind that there can be more than one way to calculate

https://medium.com/analytics-vidhya/types-of-distances-in-machine-learning-5b1233380775

Importance of Distance Metrics in Machine Learning Modelling

A number of Machine Learning Algorithms - Supervised or Unsupervised, use Distance Metrics to know the input data pattern in order to make any Data Based decision. A good distance metric

https://towardsdatascience.com/importance-of-distance-metrics-in-machine-learning-modelling-e51395ffe60d



- **Minkowski distance** →

👉 "A Normed vector space is a vector space on which a norm is defined."

Suppose A is a vector space then a norm on A

$$\left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{1/p}$$

Minkowski distance is a generalized distance metric. We can manipulate the above formula by substituting 'p' to calculate the distance between two data points in different ways. Thus, Minkowski

is a real-valued function ||A||which satisfies below conditions -

1. **Zero Vector-** Zero vector will have zero length.
2. **Scalar Factor-** The direction of the vector doesn't change when you multiply it with a positive number though its length will be changed.
3. **Triangle Inequality-** If the distance is a norm then the calculated distance between two points will always be a straight line.

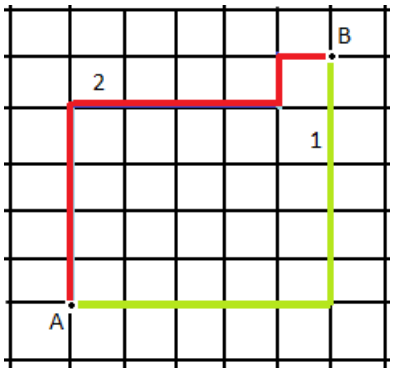
The distance can be calculated using the formula on the right:-

Distance is also known as Lp norm distance.

Some common values of 'p' are:-

- p = 1, Manhattan Distance
- p = 2, Euclidean Distance
- p = infinity, Chebychev Distance

- **Manhattan distance →**
  - We use this if we have to calculate distance between 2 points in a grid-like path.



  - The formula of Manhattan distance is calculated by substituting $p = 1$ in Mikowski formula. The formula is:

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

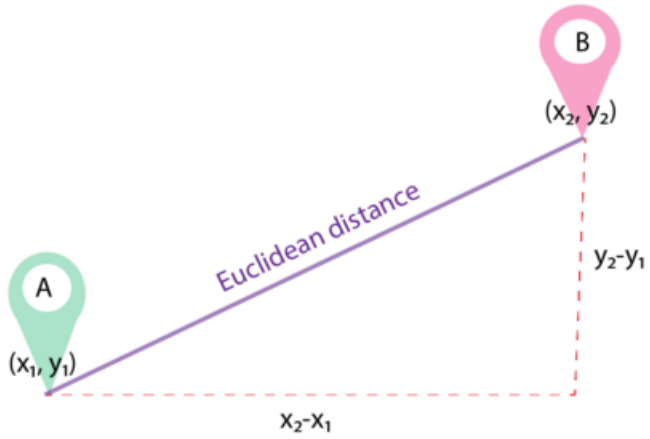
When Manhattan distance metric is preferred?

“ for a given problem with a fixed (high) value of the dimensionality d, it may be preferable to use lower values of p. This means that the L1 distance metric (Manhattan Distance metric) is the most preferable for high dimensional applications.”

Thus, ***Manhattan Distance is preferred over the Euclidean distance metric as the dimension of the data increases.*** This occurs due to something known as the ‘curse of dimensionality’.

- **Euclidean distance →**

  Euclidean distance is the straight line distance between 2 data points in a plane.



  - The formula of Manhattan distance is calculated by substituting $p = 2$ in Mikowski formula. The formula is:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- **Hamming distance →**

  Hamming distance is a metric for comparing two binary data strings. While comparing two binary strings of equal length, Hamming distance is the number of bit positions in which the two bits are different.

  The Hamming distance between two strings, a and b is denoted as d(a,b).

  In order to calculate the Hamming distance between two strings, and, we perform their XOR operation, (a⊕ b), and then count the total number of

1s in the resultant string.

Suppose there are two strings 11011001 and 10011101.

11011001 ⊕ 10011101 = 01000100. Since, this contains two 1s, the Hamming distance, d(11011001, 10011101) = 2.

- **Cosine distance →**

  Cosine distance & Cosine Similarity metric is mainly used to find similarities between two data points. As the cosine distance between the data points increases, the cosine similarity, or the amount of similarity decreases, and vice versa. Thus, Points closer to each other are more similar than points that are far away from each other. Cosine similarity is given by Cosθ, and cosine distance is 1- Cosθ.

  Where is it used?

  Cosine metric is mainly used in Collaborative Filtering based recommendation systems to offer future recommendations to users.

  Taking the example of a movie recommendation system, Suppose one user (User #1) has watched movies like The Fault in our Stars, and The Notebook, which are of romantic genres, and another user (User #2) has watched movies like The Proposal, and Notting Hill, which are also of romantic genres. So the recommendation system will use this data to recommend User #1 to see The Proposal, and Notting Hill as User #1 and User #2 both prefer the romantic genre and its likely that User #1 will like to watch another romantic genre movie and not a horror one.

  Similarly, Suppose User #1 loves to watch movies based on horror, and User #2 loves the romance genre. In this case, User #2 won't be suggested to watch a horror movie as there is no similarity between the romantic genre and the horror genre.

Note →

Manhattan distance is usually preferred over the more common Euclidean distance when there is high dimensionality in the data. Hamming distance is used to measure the distance between categorical variables, and the Cosine distance metric is mainly used to find the amount of similarity between two data points.

- **Mahalanobis distance →**

  Mahalanobis Distance is used for calculating the distance between two data points in a multivariate space.

  "The Mahalanobis distance is a measure of the distance between a point P and a distribution D. The idea of measuring is, how many standard deviations away P is from the mean of D."

  The benefit of using mahalanobis distance is, it takes covariance in account which helps in measuring the strength/similarity between two different data objects. The distance between an observation and the mean can be calculated as below -

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})}.$$

> 👉 ***The higher the norm index, the more it focuses on large values and neglects small ones. This is why the RMSE is more sensitive to outliers than the MAE. But when the outliers are exponentially rare (like in a bell-shaped curve), the RMSE performs very well and is generally preferred.***

Handling the Missing data

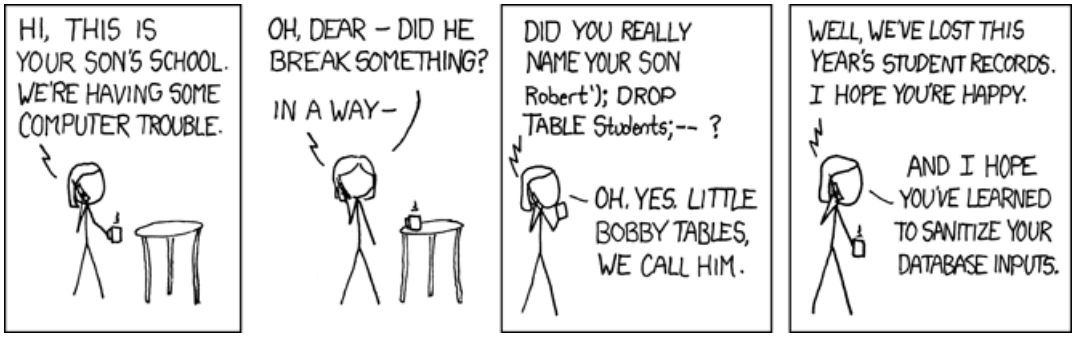
The missing values can be classified into 3 categories\types:

1. Missing completely at Random(MCAR)
    - No relationship between whether a data point is missing any values in the data set, missing or observed.
    - Just a random subset of data.
2. Missing at Random(MAR)
    - The propensity of missing values has a systematic relationship with the observed data but not the missing data.
    - the observation which is missing has nothing to do with the missing values but it has correlation with the observed variables. For example if you are taking a survey of mental disorders in men and women, it might be less likely that men will report their depression or vice-versa but it has nothing to do with their level of depression.
3. Missing Not at Random(MNAR)
    - here is a distinct relationship between the propensity of a value to be missing and its values. So in our depression survey respondents with higher depression values fail to fill the survey **because** of their level of depression.

Techniques to deal with missing data.

1. **Drop Missing data→**

    In the first two types of missing data, MCAR and MAR, in general, it is safe to remove the data with missing values depending upon their occurrences, while in the third case removing observations with missing values can produce a bias in the model.



xkcd.com

2. **Imputation**→

   Imputation means to replace or fill the missing data with some value.

   Ways to impute the data.

   - A constant value that belongs to the set of possible values of that variable, such as 0, distinct from all other values.
     - By the use of fillna function
   - A mean, median or mode value for the column
     - Using fillna function.
   - A value estimated by another predictive model.
     - Predictive Modelling
   - Multiple Imputation
     - As opposed to single imputations to complete datasets, creating multiple imputations accounts for the statistical uncertainty in the imputations. In simpler words, multiple imputations narrow uncertainty about missing values by calculating several different options (imputations). Several versions of the same data set are created, which are then combined to make the best values. One of the most used methods for imputation is **MICE(Multivariate Imputation by Chained Equations).**

Feature Selection Techniques

- Effects the performance of the model hugely.

"***feature selection is a process where we automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in.***"

Benefits of feature selection on performance of the model.

- **Reduce Overfitting**: Less redundant data means less opportunity to make decisions based on noise.
- **Improves Accuracy**: Less misleading data means modeling accuracy improves.

- **Reduces Training Time:** fewer data points reduce algorithm complexity and algorithms train faster.

Feature Selection Methods

1. **Univariate selection:**

   Statistical tests can be used to select those features that have the strongest relationship with the output variable.

```
// Example code
 import pandas as pd
 import numpy as np
 from sklearn.feature_selection import SelectKBest
 from sklearn.feature_selection import chi2
 data = pd.read_csv("D://Blogs//train.csv")
 X = data.iloc[:,0:20]  #independent columns
 y = data.iloc[:,-1]    #target column i.e price range
 #apply SelectKBest class to extract top 10 best features
 bestfeatures = SelectKBest(score_func=chi2, k=10)
 fit = bestfeatures.fit(X,y)
 dfscores = pd.DataFrame(fit.scores_)
 dfcolumns = pd.DataFrame(X.columns)
 #concat two dataframes for better visualization
 featureScores = pd.concat([dfcolumns,dfscores],axis=1)
 featureScores.columns = ['Specs','Score']  #naming the dataframe columns
 print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
            Specs          Score
13            ram  931267.519053
11      px_height   17363.569536
0    battery_power   14129.866576
12       px_width    9810.586750
8        mobile_wt      95.972863
6       int_memory      89.839124
15           sc_w      16.480319
16      talk_time      13.236400
4               fc      10.135166
14           sc_h       9.614878
```

2. **Feature Importance:**

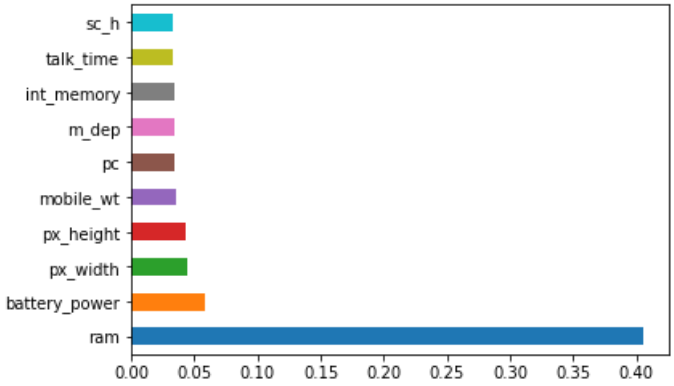
   You can get the feature importance of each feature of your dataset by using the feature importance property of the model.

   Feature importance gives you a score for each feature of your data, the higher the score more important or relevant is the feature towards your output variable.

   Feature importance is an inbuilt class that comes with Tree Based Classifiers, we will be using Extra Tree Classifier for extracting the top 10 features for the dataset.

```
import pandas as pd
 import numpy as np
 data = pd.read_csv("D://Blogs//train.csv")
 X = data.iloc[:,0:20]  #independent columns
 y = data.iloc[:,-1]    #target column i.e price range
 from sklearn.ensemble import ExtraTreesClassifier
 import matplotlib.pyplot as plt
 model = ExtraTreesClassifier()
 model.fit(X,y)
```

```
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```



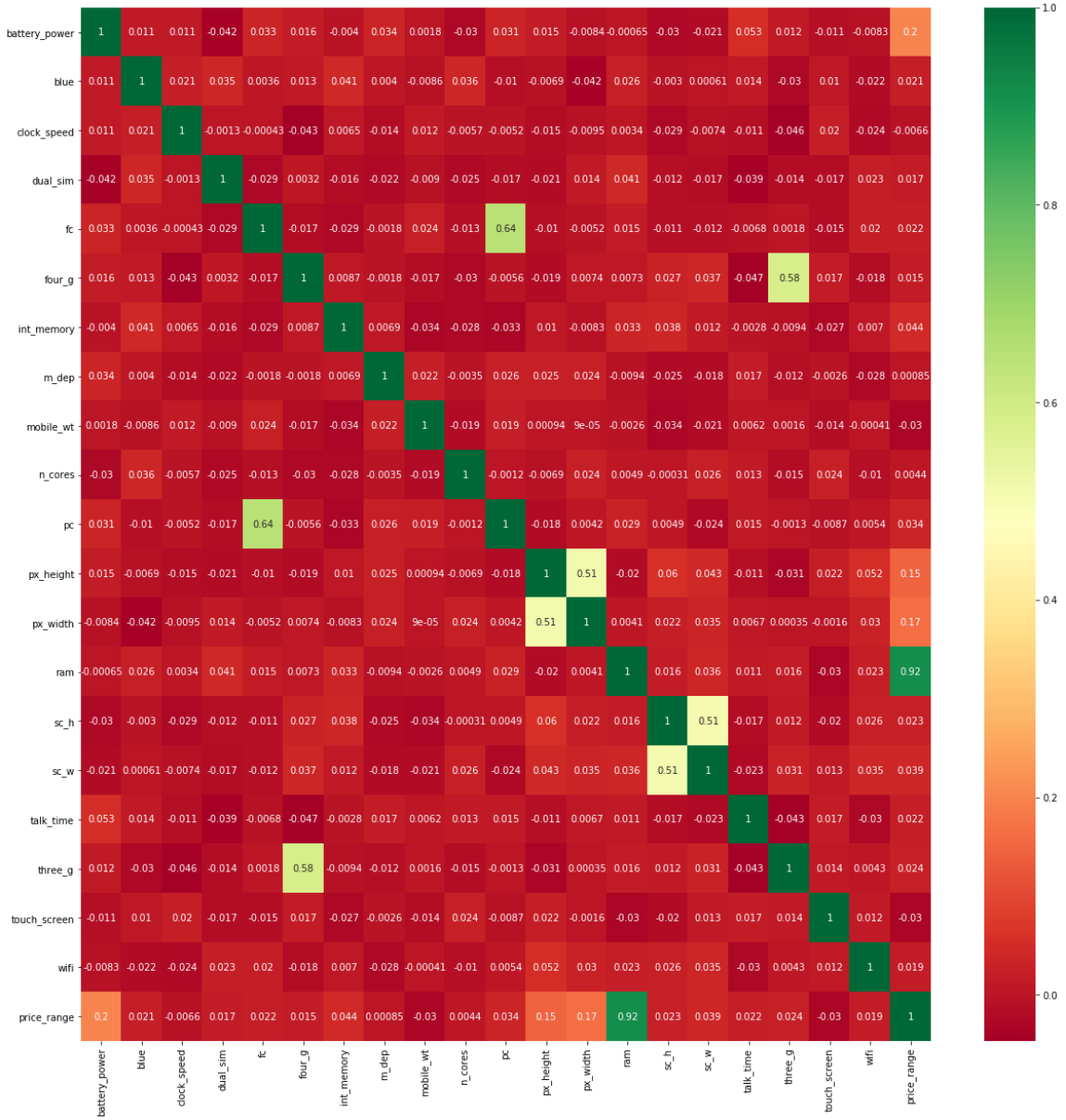
3. **Correlation Matrix with Heatmap:**

   Correlation states how the features are related to each other or the target variable.

   Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable)

   Heatmap makes it easy to identify which features are most related to the target variable, we will plot heatmap of correlated features using the seaborn library.

```
import pandas as pd
 import numpy as np
 import seaborn as snsdata = pd.read_csv("D://Blogs//train.csv")
 X = data.iloc[:,0:20]  #independent columns
 y = data.iloc[:,-1]    #target column i.e price range
 #get correlations of each features in dataset
 corrmat = data.corr()
 top_corr_features = corrmat.index
 plt.figure(figsize=(20,20))
 #plot heat map
 g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



Modern approach to Feature Selection

3 main goals of feature selection

- Improve accuracy.
- Reduce Computational Cost
- Produce a more interpretable model.

Manual feature Selection

Reasons for removing a feature from training →

- A feature is highly correlated with another feature meaning they are providing same information.
- Feature provide little to no information.

- Feature that have little to no statistical relation with the target variable.

Techniques to manually perform feature selection

1. Correlation Plot→

   create a visualization that plots the correlation measure for every feature in the data set.

   In the resulting visualisation, we can identify some closely correlated features. We may want to remove some of these, and some features have a very low correlation with the target variable, which we may also want to remove.

2. Feature Importance→

   Once we have trained a model it is possible to apply further statistical analysis to understand the effects features have on the output of the model and determine from this which features are most useful.

   There are several tools and techniques available to determine feature importance. Some techniques are unique to a specific algorithm, whereas others can be applied to a wide range of models and are known as **model agnostic**.

   This gives a good indicator of those features that are having an impact on the model and those that are not.

Automated Feature Selection

Techniques for Automated perform feature selection:

1. Variance threshold.

   In statistics, variance is the squared deviation of a variable from its mean, in other words, how far are the data points spread out for a given variable?

   Suppose we were building a machine learning model to detect breast cancer and the data set had a boolean variable for gender. This data set is likely to consist almost entirely of one gender and therefore nearly all data points would be 1. This variable would have extremely low variance and would be not at all useful for predicting the target variable.

2. Univariate feature selection

   Univariate feature selection applies univariate statistical tests to features and selects those which perform the best in these tests. Univariate tests are tests that involve only one dependent variable. This includes analysis of variance (ANOVA), linear regressions and t-tests of means.

3. Recursive feature elimination

   This method performs model training on a gradually smaller and smaller set of features. Each time the feature importances or coefficients are

calculated and the features with the lowest scores are removed. At the end of this process, the optimal set of features is known.

Feature Engineering

Where the goal of feature selection is to reduce the dimensionality of a data set by removing unnecessary features, feature engineering is about transforming existing features and constructing new features to improve the performance of a model.

There are three main reasons why we may need to perform feature engineering to develop an optimal model.

1. **Features cannot be used in their raw form.** This includes features such as dates and times, where a machine learning model can only make use of the information contained within them if they are transformed into a numerical representation e.g. integer representation of the day of the week.
2. **Features can be used in their raw form but the information contained within the feature is stronger if the data is aggregated or represented in a different way.** An example here might be a feature containing the age of a person, aggregating the ages into buckets or bins may better represent the relationship to the target.
3. **A feature on its own does not have a strong enough statistical relationship with the target but when combined with another feature has a meaningful relationship.** Let's say we have a data set that has a number of features based on credit history for a group of customers and a target that denotes if they have defaulted on a loan. Suppose we have a loan amount and a salary value. If we combined these into a new feature called "loan to salary ratio" this may give more or better information than those features alone.

Manual feature engineering

Feature engineering can be performed through data analysis, intuition and domain knowledge. Often similar techniques to those used for manual feature selection are performed.

For example, observing how features correlate to the target and how they perform in terms of feature importance indicates which features to explore further in terms of analysis.

If we go back to the example of the loan data set, let's imagine we have an age variable which from a correlation plot appears to have some relationship to the target variable. We might find when we further analyse this relationship that those that default are skewed towards a particular age group. In this case, we might engineer a feature that picks out this age group and this may provide better information to the model.

Automated feature engineering

Manual feature engineering can be an extremely time-consuming process and requires a large amount of human intuition and domain knowledge to get right. There are tools available that have the ability to automatically synthesise a large number of new features.

The Featuretools python library is an example of a tool that can perform automated feature engineering on a data set. Let's install this library and walk through an example of automated feature engineering on the breast cancer data set.

SciKit-Learn Design

Consistency

All objects share a consistent and simple interface

Estimators:

Any object that can estimate some parameters based on a dataset is called an estimator(e.g., Imputer is an estimator). The estimation itself is preformed by the .fit() function that only takes dataset as a parameter. Any other parameter needed to guide the estimation process is called an hyperparameter(such as imputer's strategy), and it must be set as an instance variable

Transformers:

Some Estimators can also transform a dataset and hence are called *transformers*. The transformation is performed by transform() method which takes dataset as an input and returns the transformed dataset. `fit_transform()` function is equivalent to calling `fit()` and then `tranform()` (but sometimes `fit_tranform()` is optimized and much faster.)

Predictors:

Some estimators, given a dataset are capable for making predictions; they are called *predictors*. For example, the LinearRegression model is a predictor. A predictor has a `predict()` method that takes a dataset of new instances and returns a dataset of corresponding predictions. It also has a `score()` method that measures the quality of predictions, given a test set.

Inspection:

All the estimator's hyperparameter are accessible via public instance variables, and all the estimator's learned parameters are accessible via public instance variables with an underscore suffix.

Nonproliferation of class:

> 👉 ***Nonproliferation***→ the prevention of an increase or spread of something.

Datasets are represented as NumPy arrays or SciPy sparse metrics, instead of homemade classes. Hyperparameters are just regular Python strings or numbers.

Composition:

Existing building blocks are used as much as possible.

Sensible defaults:

Scikit-Learn provides reasonable default values for most parameters, making it easy to quickly create a baseline working system.

Fine Tuning the models

Grid Search

For fine-tuning the model one way is to fiddle with hyperparameter manually, until we find a great combination to work with. This would be very tedious and time consuming.

Instead we can use `GridSearchCV().` It just need the hyperparameters that we want to try and values of them. It will use cross validation to try out all the hyperparameter values.

> 👉 If you have no idea what value a hyperparameter should have, a simple approach is to try out consecutive powers of 10(or a smaller number if you want a more fine-grained search).

Randomized Search

The grid approach is fine when we are evaluating a few combination of hyperparameters, but when the hyperparameter search space is large , it is often preferable to use `RandomizedSearchCV`.This class, instead of trying out all the possible combinations, it evaluates a given number of random combinations by selecting a random value for each hyperparameter at every iteration.

Benefits:

- The search will explore N number of different values for each hyperparameter(instead of just a few values per hyperparameter with the grid search).
- Simply by setting the number of iterations, we have more control over the computing budget we want to allocate to hyperparameter search.

💡 `RandomizedSearchCV()` tend to find better hyperparameters than `GridSearchCV()` in same amount of time.

Ensemble Methods

- To combine the models that perform best. The group(or "ensemble") will perform better that the best individual model.

After you shipped your model and send it for production then you need to prevent your model from "rotting". For this you might have to implement a monitoring system which requires human intervention(in some case) and some times its just code.

For making this process easier you can make your model as much automated as possible.

Some things that can be automated are as follows👇

- Collect fresh data regularly and label it. (e.g., using human raters)
- Write a script to train the model and fine-tune the hyperparameters automatically. This script could run automatically, for example, every day or even week, depending on your needs.
- Write another script that will evaluate both the new model and the previous model on the updated test set, and deploy the model to production if the performance has not decreased (*if it did make sure to investigate why?*).