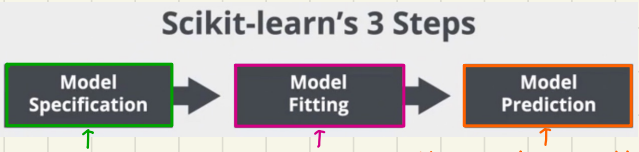
SCIKIT-LEARN

THE CORE SKLEARN API

- Has wide range of algorithmic options covering: REGRESSION, CLASSIFICATION, and UNSUPERVISED LEARNING
- Provides libraries on: DATA PREPARATION, MODEL SELECTION, and EVALUATION



Instantiate the model class and tell scikit-learn how you want to configure the algorithm

Training the model

When you want to use your model for evaluation or prediction purposes, this step steps in.

3 steps represented in actual code

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression (C = 1)
model.fit(df[x], df[y])
prediction = model.predict(df[x])
```

In the following coding example, we can call the training and a loop:

Import diff algorithms

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

Gives us the ability to automate the process of testing and choosing different algorithms = MODEL SELECTION

dictionary

```
models = {  'LR' :LogisticRegression(C = 1),
            'KNN' :KNeighborsClassifier(n_neighbors = 100),
            'DT' :DecisionTreeClassifier(min_samples_leaf = 256)
        }
```

Each element is a different algorithm

fit method within the loop

```
for m in models:
    models[m].fit(df[X], df[y])
```

NOTE: KNN = K-Nearest neighbors

sklearn.model_selection.train_test_split(): splits arrays or matricies into random train and test subsets.

sklearn.model_selection.**train_test_split**(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None*) [source]

Parameters:

arrays : sequence of indexables with same length / shape[0]
Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : float or int, default=None
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : float or int, default=None
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

random_state : int, RandomState instance or None, default=None
Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.

shuffle : bool, default=True
Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

stratify : array-like, default=None
If not None, data is split in a stratified fashion, using this as the class labels. Read more in the User Guide.

Returns:

splitting : list, length=2 * len(arrays)
List containing train-test split of inputs.

New in version 0.16: If the input is sparse, the output will be a `scipy.sparse.csr_matrix`. Else, output type is the same as the input type.

sklearn.metrics.accuracy_score(): Accuracy classification score.

```
sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)
```

- y_true: ground truth (correct) labels
- y_pred: predicted labels, as returned by classifier
- normalize: False = returns the number of correctly classified samples; True = returns fraction of correctly classified samples

sklearn.neighbors.KNeighborsClassifier(): Classifier implementing the k-nearest neighbors vote.

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5*, ***, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*) [source]

Parameters:

n_neighbors : *int, default=5*
Number of neighbors to use by default for `kneighbors` queries.

weights : *{'uniform', 'distance'} or callable, default='uniform'*
Weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*
Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size : *int, default=30*
Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : *int, default=2*
Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : *str or callable, default='minkowski'*
The distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric. For a list of available metrics, see the documentation of `DistanceMetric` and the metrics listed in `sklearn.metrics.pairwise.PAIRWISE_DISTANCE_FUNCTIONS`. Note that the "cosine" metric uses `cosine_distances`. If metric is "precomputed", X is assumed to be a distance matrix and must be square during fit. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors.

metric_params : *dict, default=None*
Additional keyword arguments for the metric function.

choice of distance metric can be another type of hyperparameter

n_jobs : *int, default=None*
The number of parallel jobs to run for neighbors search. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details. Doesn't affect `fit` method.

class sklearn.preprocessing.**OneHotEncoder**(***, *categories='auto'*, *drop=None*, *sparse=True*, *dtype=<class 'numpy.float64'>*, *handle_unknown='error'*, *min_frequency=None*, *max_categories=None*) [source]

Encode categorical features as a one-hot numeric array.

Parameters:

categories : *'auto' or a list of array-like, default='auto'*
Categories (unique values) per feature:

- 'auto' : Determine categories automatically from the training data.
- list : `categories[i]` holds the categories expected in the ith column. The passed categories should not mix strings and numeric values within a single feature, and should be sorted in case of numeric values.

The used categories can be found in the `categories_` attribute.

New in version 0.20.

drop : *{'first', 'if_binary'} or an array-like of shape (n_features,), default=None*
Specifies a methodology to use to drop one of the categories per feature. This is useful in situations where perfectly collinear features cause problems, such as when feeding the resulting data into an unregularized linear regression model.

However, dropping one category breaks the symmetry of the original representation and can therefore induce a bias in downstream models, for instance for penalized linear classification or regression models.

- None : retain all features (the default).
- 'first' : drop the first category in each feature. If only one category is present, the feature will be dropped entirely.
- 'if_binary' : drop the first category in each feature with two categories. Features with 1 or more than 2 categories are left intact.
- array : `drop[i]` is the category in feature `X[:, i]` that should be dropped.

sparse : *bool, default=True*
Will return sparse matrix if set True else will return an array.

dtype : *number type, default=float*
Desired dtype of output.

handle_unknown : *{'error', 'ignore', 'infrequent_if_exist'}, default='error'*
Specifies the way unknown categories are handled during `transform`.

- 'error' : Raise an error if an unknown category is present during transform.
- 'ignore' : When an unknown category is encountered during transform, the resulting one-hot encoded columns for this feature will be all zeros. In the inverse transform, an unknown category will be denoted as None.
- 'infrequent_if_exist' : When an unknown category is encountered during transform, the resulting one-hot encoded columns for this feature will map to the infrequent category if it exists. The infrequent category will be mapped to the last position in the encoding. During inverse transform, an unknown category will be mapped to the category denoted `'infrequent'` if it exists. If the `'infrequent'` category does not exist, then `transform` and `inverse_transform` will handle an unknown category as with `handle_unknown='ignore'`. Infrequent categories exist based on `min_frequency` and `max_categories`. Read more in the User Guide.

Changed in version 1.1: `'infrequent_if_exist'` was added to automatically handle unknown categories and infrequent categories.

min_frequency : *int or float, default=None*
Specifies the minimum frequency below which a category will be considered infrequent.

- If `int`, categories with a smaller cardinality will be considered infrequent.
- If `float`, categories with a smaller cardinality than `min_frequency * n_samples` will be considered infrequent.

max_categories : *int, default=None*
Specifies an upper limit to the number of output features for each input feature when considering infrequent categories. If there are infrequent categories, `max_categories` includes the category representing the infrequent categories along with the frequent categories. If `None`, there is no limit to the number of output features.

New in version 1.1: Read more in the User Guide.

CHEAT SHEET

K-Nearest Neighbors

Algorithm Name	KNN
Description	For a given test point, the KNN algorithm identifies the k most similar training points and finds the most common label among them. This label is used as a prediction for the test point.
Applicability	Often competitive in low-dimensional spaces in settings with many classes; used for classification or regression.
Assumptions	"Similar inputs have similar labels"; KNN assumes that the user has a way to compute distances that reflect meaningful dissimilarities.
Underlying Mathematical Principles	• Distance metrics
Additional Details	• Hyperparameter is number of neighbors (k) • Dealing with ties — fall back to smaller k-values • Distance metric used is application specific
Example	Identify individuals visible in a photo uploaded to a social media account.