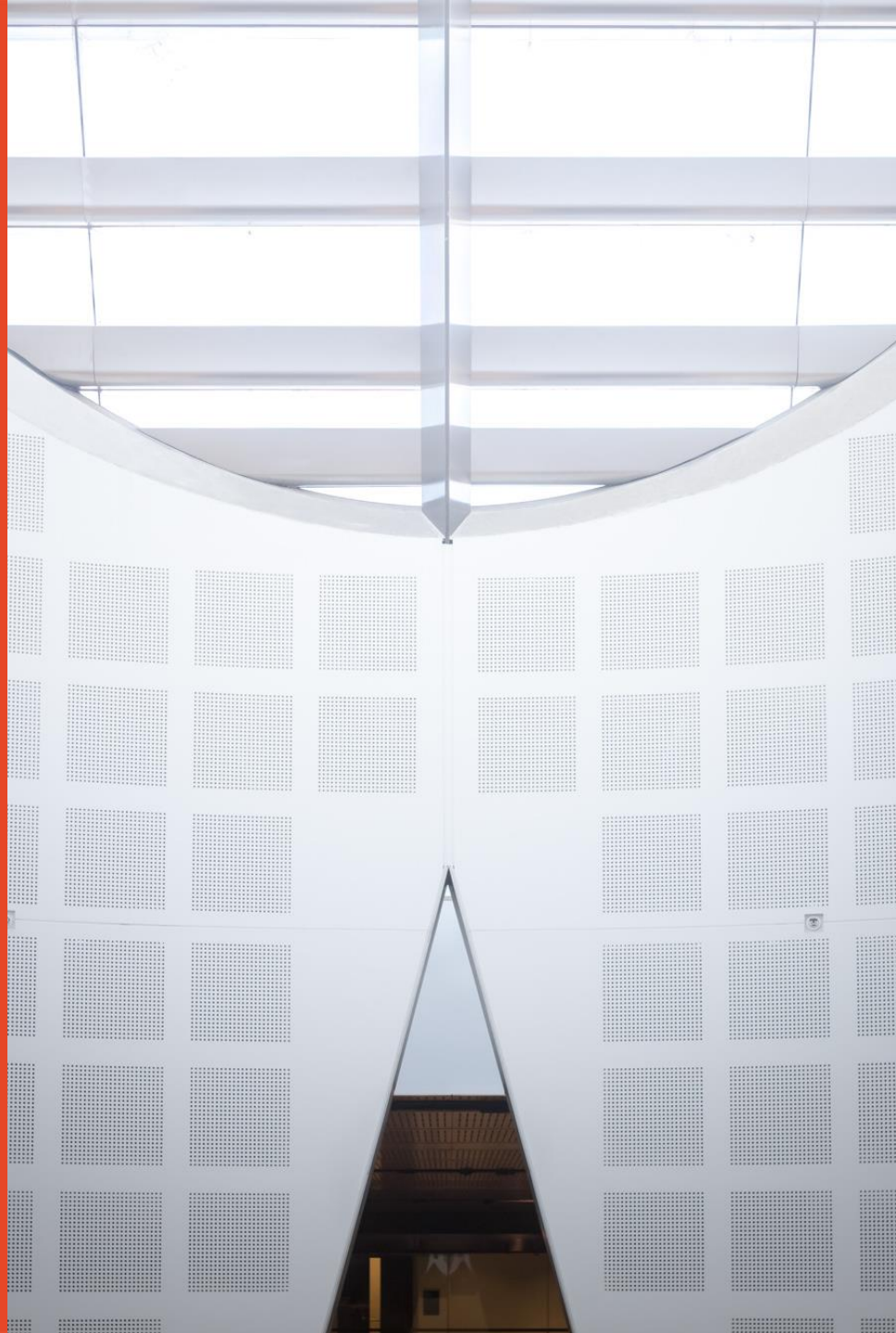
Clinic Prediction Models Package Use Example



THE UNIVERSITY OF SYDNEY

Linear Regression (sklearn package)

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Linear Regression Introduction (sklearn package)

class sklearn.linear_model.**LinearRegression**(*, *fit_intercept=True*, *normalize='deprecated'*, *copy_X=True*, *n_jobs=None*, *positive=False*) [source]

Main Parameters:

- *fit_intercept*: **bool**, **default=True**. Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).
- *normalize*: **bool**, **default=False.** This parameter is ignored when *fit_intercept* is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use **StandardScaler** before calling *fit* on an estimator with *normalize=False*.

Linear Regression Introduction (sklearn package)

class sklearn.linear_model.**LinearRegression**(*, *fit_intercept=True*, *normalize='deprecated'*, *copy_X=True*, *n_jobs=None*, *positive=False*) [source]

Main Attributes:

- *coef_*: **array of shape (n_features,) or (n_targets, n_features).** Estimated coefficients for the linear regression problem. If multiple targets are passed during the fit (y 2D), this is a 2D array of shape (n_targets, n_features), while if only one target is passed, this is a 1D array of length n_features.
- *intercept_*: **float or array of shape (n_targets,).** Independent term in the linear model. Set to 0.0 if *fit_intercept = False*.
- *rank_*: **int.** Rank of matrix X. Only available when X is dense.

For more details and other parameter definitions, link.

Linear Regression Introduction (sklearn package)

class sklearn.linear_model.**LinearRegression**(*, *fit_intercept=True*, *normalize='deprecated'*, *copy_X=True*, *n_jobs=None*, *positive=False*) [source]

Main Methods:

- *fit(X, y[, sample_weight])*: Fit (train) linear model.
- *predict(X)*: Predict using the linear model.
- *get_params([deep])*: Get parameters for this estimator.

Linear Regression Usage (sklearn package)

For the used data, we use the SOCR Body Density Data as before. Our task is to find out he relationship between height and weight.

```
X = df[['Height']]
y = df[['Weight(kg)']]

# Seperate the data into the train set and the test set.
X_train,X_test,y_train,y_test= model_selection.train_test_
split(X, y, test_size=0.1,random_state=42)
```

For using it simply, we just import the *sklearn.linear_model* package, and instantiate a Linear Regression model, like

```
# Call the linear regression model
from sklearn import linear_model
line_reg = linear_model.LinearRegression()
```

Linear Regression Usage (sklearn package)

Train the LR model with the separated training data:

```
# Train
line_reg.fit(X_train,y_train)
```

Get the parameters:

```
w_0 = line_reg.intercept_
w_1 = line_reg.coef_
print(f'w_0: {w_0}, w_1: {w_1}.')
```

Output:

```
w_0: [7.89612647], w_1: [[0.41119549]].
```

Linear Regression Usage (sklearn package)

Calculate the errors:

```
y_predict_in_train= line_reg.predict(X_train)
y_predict_in_test = line_reg.predict(X_test)
error_in_train  = metrics.mean_squared_error(y_predict_in_
train,y_train)
error_in_test   = metrics.mean_squared_error(y_predict_in_
test,y_test)
print("MSE in train: {}".format(error_in_train))
print("MSE in test: {}".format(error_in_test))
```

Output:

```
MSE in train: 168.54498628490126
MSW in test: 88.25561906418989
```

k -NN (sklearn package)

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

k-NN Introduction (sklearn package)

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]

Main Parameters:

- *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.

k-NN Introduction (sklearn package)

Main Parameters:

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

k-NN Introduction (sklearn package)

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]

Main Parameters:

- *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 (l_1), and Euclidean distance (l_2) for $p = 2$. For arbitrary p, Minkowski distance (l_p) is used.

k-NN Introduction (sklearn package)

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]

Main Parameters:

- *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.

k-NN Introduction (sklearn package)

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]

Main Parameters:

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

k-NN Introduction (sklearn package)

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]

Main Methods:

- *fit(X, y)*: Fit the *k*-nearest neighbors classifier from the training dataset.
- *predict(X)*: Predict the class labels for the provided data.
- *get_params([deep]):* Get parameters for this estimator.

SVM (sklearn package)

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

SVM Introduction (sklearn package)

```
class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001,
cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False,
random_state=None)
```
[source]

Main Parameters:

- *C*: **float, default=1.0.** Regularization parameter. The strength of the regularization is inversely proportional to **C**. Must be strictly positive. The penalty is a squared l_2 penalty.
- *kernel*: **{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'.** Specifies the kernel type to be used in the algorithm. If none is given, '**rbf**' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape $(n\_samples, n\_samples)$.

SVM Introduction (sklearn package)

class `sklearn.svm.`**SVC**(*, *C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

Main Parameters:

- *degree*: **int, default=3.** Degree of the polynomial kernel function (‘**poly**’). Ignored by all other kernels.
- *max_iter:* **int, default=-1**. Hard limit on iterations within solver, or **-1** for no limit.

Main Methods:

- *fit(X, y[, sample_weight])*: Fit the SVM model according to the given training data.
- *predict(X)*: Perform classification on samples in X.

SVM Introduction (sklearn package)

class `sklearn.svm.`**SVC**(*, *C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

Main Methods:

- *get_params([deep])*: Get parameters for this estimator.
- *score(X, y[, sample_weight])*: Return the mean accuracy on the given test data and labels.
- *decision_function(X)*: Evaluate the decision function for the samples in X.

Perceptron (sklearn package)

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

Perceptron Introduction (sklearn package)

class `sklearn.linear_model.`**`Perceptron`**(*, *penalty=None, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, eta0=1.0, n_jobs=None, random_state=0, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False*) [source]

Main Parameters:

- *penalty:* **{'l2','l1','elasticnet'}, default=None.** The penalty (aka regularization term) to be used.
- *alpha*: **float, default=0.0001**. Constant that multiplies the regularization term if regularization is used.
- *tol*: **float, default=1e-3.** The stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol).
- *random_state*: **int, RandomState instance or None, default=0.** Used to shuffle the training data, when shuffle is set to True. Pass an int for reproducible output across multiple function calls. See Glossary.

Perceptron Introduction (sklearn package)

class `sklearn.linear_model.`**`Perceptron`**(*, *penalty=None, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, eta0=1.0, n_jobs=None, random_state=0, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False*) [source]

Main Parameters:

- *warm_start:* **bool, default=False**. When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution. See the Glossary.

Main Methods:

- *fit(X, y[, coef_init, intercept_init, ...])*: Fit linear model with Stochastic Gradient Descent.
- *predict(X)*: Predict class labels for samples in X.
- *score(X, y[, sample_weight])*: Return the mean accuracy on the given test data and labels.

Summary

In this tutorial, we mainly learn how to use *sklearn* package to train and test common clinical prediction models for clinical tasks:

- ***Linear Regression***
- ***k-NN***
- ***SVM***
- ***Perceptron***

Thank you!