**Python For Data Science** *Cheat Sheet*



**DataCamp**

**Learn Python for Data Science Interactively**

**Loading The Data**

**Also see NumPy & Pandas**

**Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.**

>>> import numpy as np

>>> X = np.random.random((10,5))

>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])

>>> X[X < 0.7] = 0

**Create Your Model**

**Supervised Learning Estimators**

**Unsupervised Learning Estimators**

**Principal Component Analysis (PCA)**

>>> from sklearn.decomposition import PCA

>>> pca = PCA(n\_components=0.95)

**K Means**

>>> from sklearn.cluster import KMeans

>>> k\_means = KMeans(n\_clusters=3, random\_state=0)

**Linear Regression**

>>> from sklearn.linear\_model import LinearRegression

>>> lr = LinearRegression(normalize=True)

**Support Vector Machines (SVM)**

>>> from sklearn.svm import SVC

>>> svc = SVC(kernel='linear')

**Naive Bayes**

>>> from sklearn.naive\_bayes import GaussianNB

>>> gnb = GaussianNB()

**KNN**

>>> from sklearn import neighbors

>>> knn = neighbors.KNeighborsClassifier(n\_neighbors=5)

**Model Fitting**

Fit the model to the data

Fit to data, then transform it

Fit the model to the data

**Supervised learning**

>>> lr.fit(X, y)

>>> knn.fit(X\_train, y\_train)

>>> svc.fit(X\_train, y\_train)

**Unsupervised Learning**

>>> k\_means.fit(X\_train)

>>> pca\_model = pca.fit\_transform(X\_train)

**Tune Your Model**

**Grid Search**

**Randomized Parameter Optimization**

>>> from sklearn.grid\_search import RandomizedSearchCV

>>> params = {"n\_neighbors": range(1,5),

"weights": ["uniform", "distance"]}

>>> rsearch = RandomizedSearchCV(estimator=knn,

param\_distributions=params, cv=4,

n\_iter=8,

random\_state=5)

>>> rsearch.fit(X\_train, y\_train)

>>> print(rsearch.best\_score\_)

>>> from sklearn.grid\_search import GridSearchCV

>>> params = {"n\_neighbors": np.arange(1,3),

"metric": ["euclidean", "cityblock"]}

>>> grid = GridSearchCV(estimator=knn,

param\_grid=params)

>>> grid.fit(X\_train, y\_train)

>>> print(grid.best\_score\_)

>>> print(grid.best\_estimator\_.n\_neighbors)

**Prediction**

Predict labels Predict labels

Estimate probability of a label

Predict labels in clustering algos

**Supervised Estimators**

>>> y\_pred = svc.predict(np.random.random((2,5)))

>>> y\_pred = lr.predict(X\_test)

>>> y\_pred = knn.predict\_proba(X\_test)

**Unsupervised Estimators**

>>> y\_pred = k\_means.predict(X\_test)

**Evaluate Your Model’s Performance**

**Classification Metrics**

**Regression Metrics**

**Clustering Metrics**

**Cross-Validation**

>>> from sklearn.cross\_validation import cross\_val\_score

>>> print(cross\_val\_score(knn, X\_train, y\_train, cv=4))

>>> print(cross\_val\_score(lr, X, y, cv=2))

**Adjusted Rand Index**

>>> from sklearn.metrics import adjusted\_rand\_score

>>> adjusted\_rand\_score(y\_true, y\_pred)

**Homogeneity**

>>> from sklearn.metrics import homogeneity\_score

>>> homogeneity\_score(y\_true, y\_pred)

**V-measure**

>>> from sklearn.metrics import v\_measure\_score

>>> metrics.v\_measure\_score(y\_true, y\_pred)

**Mean Absolute Error**

>>> from sklearn.metrics import mean\_absolute\_error

>>> y\_true = [3, -0.5, 2]

>>> mean\_absolute\_error(y\_true, y\_pred)

**Mean Squared Error**

>>> from sklearn.metrics import mean\_squared\_error

>>> mean\_squared\_error(y\_test, y\_pred)

**R² Score**

>>> from sklearn.metrics import r2\_score

>>> r2\_score(y\_true, y\_pred)

**Standardization**

**Encoding Categorical Features**

**Normalization**

**Imputing Missing Values**

**Binarization**

**Generating Polynomial Features**

**Preprocessing The Data**

>>> from sklearn.preprocessing import PolynomialFeatures

>>> poly = PolynomialFeatures(5)

>>> poly.fit\_transform(X)

>>> from sklearn.preprocessing import Binarizer

>>> binarizer = Binarizer(threshold=0.0).fit(X)

>>> binary\_X = binarizer.transform(X)

>>> from sklearn.preprocessing import Imputer

>>> imp = Imputer(missing\_values=0, strategy='mean', axis=0)

>>> imp.fit\_transform(X\_train)

>>> from sklearn.preprocessing import Normalizer

>>> scaler = Normalizer().fit(X\_train)

>>> normalized\_X = scaler.transform(X\_train)

>>> normalized\_X\_test = scaler.transform(X\_test)

>>> from sklearn.preprocessing import LabelEncoder

>>> enc = LabelEncoder()

>>> y = enc.fit\_transform(y)

>>> from sklearn.preprocessing import StandardScaler

>>> scaler = StandardScaler().fit(X\_train)

>>> standardized\_X = scaler.transform(X\_train)

>>> standardized\_X\_test = scaler.transform(X\_test)

**Training And Test Data**

>>> from sklearn.model\_selection import train\_test\_split

>>> X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,

y, random\_state=0)

Scikit-Learn

|  |  |
| --- | --- |
| **Accuracy Score** |  |
| >>> knn.score(X\_test, y\_test) | Estimator score method |
| >>> from sklearn.metrics import accuracy\_score  >>> accuracy\_score(y\_test, y\_pred) | Metric scoring functions |
| **Classification Report**  >>> from sklearn.metrics import classification\_report  >>> print(classification\_report(y\_test, y\_pred)  **Confusion Matrix**  >>> from sklearn.metrics import confusion\_matrix  >>> print(confusion\_matrix(y\_test, y\_pred)) | Precision, recall, f1-score and support |

Learn Python for data science **Interactively** at [www.DataCamp.com](http://www.DataCamp.com/)



**Scikit-learn**

**Scikit-learn** is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

**A Basic Example**

>>> from sklearn import neighbors, datasets, preprocessing

>>> from sklearn.model\_selection import train\_test\_split

>>> from sklearn.metrics import accuracy\_score

>>> iris = datasets.load\_iris()

>>> X, y = iris.data[:, :2], iris.target

>>> X\_train, X\_test, y\_train, y\_test= train\_test\_split(X,y,random\_state=33)

>>> scaler = preprocessing.StandardScaler().fit(X\_train)

>>> X\_train = scaler.transform(X\_train)

>>> X\_test = scaler.transform(X\_test)

>>> knn = neighbors.KNeighborsClassifier(n\_neighbors=5)

>>> knn.fit(X\_train, y\_train)

>>> y\_pred = knn.predict(X\_test)

>>> accuracy\_score(y\_test, y\_pred)

**Cross-Validation (loop-wise)**

**Ridge/Lasso/Elastic net regression**

**Backward feature selection (regression)**

*RFECV: backward feature eliminateion and cross validation selection of the best number of features.*

from sklearn.feature\_selection import RFECV

estimator = LinearRegression(normalize=True)

selector = RFECV(estimator, 1, cv = k)

selector = selector.fit(x,y)

*Best amount of features to select given final CV score*

selector.n\_features\_

*Array with best features to select*

selector.support\_

*Ranking of each feature*

selector.ranking\_

**Lasso**

from sklearn.linear\_model import Lasso

clf = Lasso(alpha = f, max\_iter = 10 000)

clf.fit(x\_train, y\_train)

clf.coef\_

y\_pred = clf.predict(x\_test)

**Ridge**

from sklearn.linear\_model import Ridge

clf = Ridge(alpha = f, max\_iter = 10000)

clf = clf.fit(x\_train, y\_train)

clf.coef\_

y\_pred = clf.predict(x\_test)

**Elastic net**

from sklearn.linear\_model import ElasticNet

en = ElasticNet(alpha, l1\_ratio, random\_state = 0)

en.fit(x, y)

en.predict(x\_test)

en\_coef\_

*Split data before this procedure*

from sklearn.model\_selection import KFold

Kf = Kfold(n\_splits = 5, shuffle = False, random\_state = None)

for train\_index, test\_index in kf.split(x):

x\_train, x\_test = x.iloc[train\_index], x.iloc[test\_index]

y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]

*model fit and model prediction has to be done here*

**Ensemble learners (regression)**

Bagging

from sklearn.ensemble import BaggingRegressor

regr = BaggingRegressor(base\_estimator = DecisionTreeRegressor(random\_state=0, max\_depth = 2), n\_estimators = 10).fit(x\_train, y\_train)

y\_pred = regr.predict(x\_test)

RandomForest

from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(max\_depth = 1, n\_estimators = 10).fit(x\_train, y\_train)

y\_pred = regr.predict(x\_test)

Boosting

from sklearn.ensemble import AdaBoostRegressor

regr = AdaBoostRegressor(random\_state=0, n\_estimators=100, base\_estimator = DecisionTreeRegressor(random\_state=0, max\_depth = 2))

regr = regr.fit(x\_train, y\_train)

y\_pred = regr.predict(x\_test)

**Regression**

from sklearn.tree import DecisionTreeRegressor

reg = DecisionTreeRegressor(max\_depth)

reg = reg.fit(x\_train,y\_train)

y\_pred = reg.pred(y\_test, y\_pred)

**Classification**

from sklearn.tree import DecisionTreeClassifier

class = DecisionTreeClassifier(max\_depth)

class = class.fit(x\_train,y\_train)

y\_pred = class.pred(y\_test, y\_pred)

**Decision Tree**

from sklearn import metrics

from sklearn.cross\_validation import cross\_val\_score

cross\_val\_score(model, x, y, cv = 5, scoring = score)

**Obtain the metrics names that can be used**

sorted(metrics.SCORERS.keys())

**Cross-Validation (with metric adaptations)**

**Lasso**

from sklearn.linear\_model import LassoCV

clf = LassoCV(cv = 5, random\_state = 0).fit(x,y)

clf.coef\_

y\_pred = clf.pred(x)

**Ridge**

from sklearn.linear\_model import RidgeCV

clf = RidgeCV(cv = 5).fit(x,y)

clf.coef\_

y\_pred = clf.predict(x\_test)

**Elastic net**

from sklearn.linear\_model import ElasticNetCV

regr = ElasticNetCV(cv=5, random\_state=0)

**Ridge/Lasso/Elastic net regression with CV**