Cheatography

SciKit Learn for Machine Learning Cheat Sheet

by Damini via cheatography.com/121435/cs/22218/

Loading the data

```
>>> import numpy as np
>>> X = np.ran dom.ra ndo m(( -
10,5))
>>> y = np.arr ay( ['M ',' -
M', 'F' ,'F ',' -
M', 'F' ,'M ',' -
M', 'F' ,'F ','F'])
>>> X[X < 0.7] = 0
```

Training and Test data

```
>>> from sklearn.model_selection
import train_test_split
>>> X_train, X_test, y_train,
y_test = train_ tes t_s plit(X,
y, random _st ate=0)
```

Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.ra-
ndom((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test)
```

Pre-processing the data

Model Fitting

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_t-
rain)
```

Create model

Supervised Learning Estimators

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.KNeighborsClassif-
ier(n_neighbors=5)
```

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)
```

Tune Your Model

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import accura-
cy_score
>>> accuracy_score(y_test, y_pred)
```

Classification Report

```
>>> from sklearn.metrics import classi-
fication_report
>>> print(classification_report(y_test,
y_pred))
```

Confusion Matrix

```
>>> from sklearn.metrics import confus-
ion_matrix
>>> print(confusion_matrix(y_test,
y_pred))
```

Regression Metrics

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)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjust-
ed_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_meas-
ure_score
>>> metrics.v_measure_score(y_true,
y_pred)
```

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputting Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Grid Search

```
>>> 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)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(-1,5), "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV-(estimator=knn, param_distributions=p-arams, cv=4, n_iter=8, random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

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))
```

By **Damini**
cheatography.com/damini/

Not published yet.
Last updated 31st March, 2020.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com