scikit learn Reference Sheet

PRAGMATIC INSTITUTE

The data

Your data needs to be contained in a two-dimensional feature matrix and, in the case of supervised learning, a one-dimensional label vector. The data has to be numeric (NumPy array, SciPy sparse matrix, pandas DataFrame).

Transformers: preprocessing the data

EXAMPLE

`ex_transf = ExampleTransformer()`—creates a new instance

`ex_transf.fit(X_train)`—fits transformer on training data

`transf_X = ex_transf.transform(X_train)`—transforms training data

`transf_X_test = ex_transf.transform(X_test)`—transforms test data

STANDARDIZE FEATURES (ZERO MEAN, UNIT VARIANCE)

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

SCALE EACH FEATURE BY ITS MAX ABS VALUE

```
from sklearn.preprocessing import MaxAbsScaler
max_scaler = MaxAbsScaler()
```

GENERATE POLYNOMIAL FEATURES

```
from sklearn.preprocessing import PolynomialFeatures
poly_transform = PolynomialFeatures(degree=n)
```

ONE-HOT ENCODE CATEGORICAL FEATURES

```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
```

PRINCIPAL COMPONENT ANALYSIS

```
from sklearn.decomposition import PCA
pca = PCA(n_components=n)
```

Splitting into training data and test data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test  = train_test_split(X, y)
```

Predictors: supervised learning

EXAMPLE

`ex_predictor = ExamplePredictor()`—creates a new instance

`ex_predictor.fit(X_train, y_train)`—fits model on training data

`y_pred = ex_predictor.predict(X_train)`—predicts on training data

`y_pred_probs = ex_predictor.predict_proba(X_train)`—classifiers only, predicts class probabilities on training data

LINEAR REGRESSION

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

DECISION TREE REGRESSION MODEL

```
from sklearn.tree import DecisionTreeRegressor
tree = DecisionTreeRegressor(max_depth=n)
```

RANDOM FOREST REGRESSION MODEL

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
logr = LogisticRegression()
```

RANDOM FOREST CLASSIFICATION MODEL

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
```

Predictors: unsupervised learning

EXAMPLE

`ex_predictor = ExamplePredictor()`—creates a new instance

`ex_predictor.fit(X_train)`—fits model on training data

`y_pred = ex_predictor.predict(X_train)`—predicts on training data

K-MEANS CLUSTERING

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=n)
```

Evaluating model performance

```
from sklearn import metrics
```

REGRESSION METRICS

`metrics.mean_absolute_error(y_true, y_pred)`—Mean absolute error

`metrics.mean_squared_error(y_true, y_pred)`—Mean squared error

`metrics.r2_score(y_true, y_pred)`—R^2 score

CLASSIFICATION METRICS

`metrics.accuracy_score(y_true, y_pred)`—Accuracy score

`metrics.precision_score(y_true, y_pred)`—Precision score

`metrics.recall_score(y_true, y_pred)`—Recall score

`metrics.classification_report(y_true, y_pred)`—Classification report

`metrics.roc_auc_score(y_true, y_pred_probs)`—ROC AUC score

`metrics.log_loss(y_true, y_pred_probs)`—Cross-entropy loss

CLUSTERING METRICS

`metrics.silhouette_score(X_train, y_pred)`—Silhouette score

CROSS-VALIDATION

```
from sklearn.model_selection import cross_val_score
cross_val_score(lr, X_train, y_train, cv=5)
```

Pipeline

EXAMPLE

```
from sklearn.pipeline import Pipeline
pipe = Pipeline([('feature scaling', StandardScaler()),
                 ('linear regression', LinearRegression())])
```

`pipe.fit(X_train, y_train)`—fits model on training data

`y_pred = pipe.predict(X_train)`—predicts on training data

`y_pred_test = pipe.predict(X_test)`—predicts on test data

```
scaler = pipe.named_steps['feature scaling']
lr = pipe.named_steps['linear regression']
```

Feature union

EXAMPLE

```
from sklearn.pipeline import FeatureUnion
union = FeatureUnion([('transf_1', ExampleTransformer1()),
                      ('transf_2', ExampleTransformer2())])
```

`union.fit(X_train)`—fits on training data

`X_transf = union.transform(X_train)`—transforms training data

Transforming only some features/columns

EXAMPLE

```
from sklearn.compose import ColumnTransformer
example_transf = ColumnTransformer(
   [(transformer_name, transformer, columns_to_transform)])
example_transf.fit(X_train)
X_transf = example_transf.transform(X_train)
```

Optimizing hyperparameters

```
from sklearn.grid_search import GridSearchCV
grid = GridSearchCV(estimator=DecisionTreeRegressor(),
   param_grid={'max_depth': range(3, 10)})
grid.fit(X_train, y_train)
```

`print(grid.best_estimator_)`—estimator that was chosen by the search

`print(grid.best_params_)`—parameters that gave the best results