Scikit-learn Cheat Sheet

A concise cheat sheet for the scikit-learn library, covering essential functionalities for machine learning in Python. This guide includes key concepts, model selection, preprocessing techniques, and evaluation metrics with practical examples.

Core Concepts & Model Selection

Supervised Learning Estimators

Scikit-learn offers various supervised learning estimators for different tasks.

Linear Models:

- `LinearRegression`: For regression tasks.
- `LogisticRegression`: For classification tasks.
- `Ridge`: Linear least squares with L2 regularization.

Example:

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

Support Vector Machines (SVM):

- `SVC`: Support Vector Classification.
- `SVR`: Support Vector Regression.

Example:

```
from sklearn.svm import SVC
model = SVC()
```

Ensemble Methods:

- `RandomForestClassifier`: For classification tasks.
- `RandomForestRegressor`: For regression tasks.
- `GradientBoostingClassifier`: For classification tasks.

Example:

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

Unsupervised Learning Estimators

Scikit-learn provides unsupervised learning estimators for tasks like clustering and dimensionality reduction.

Clustering:

- `KMeans`: K-Means clustering.
- `AgglomerativeClustering`: Agglomerative clustering.

Example:

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
```

Dimensionality Reduction:

- `PCA`: Principal Component Analysis.
- `TruncatedSVD`: Truncated Singular Value Decomposition.

Example:

```
from sklearn.decomposition import PCA
model = PCA(n_components=2)
```

Model Fitting and Prediction

`fit(X, y)`	Fit the model using the training data `X` and target `y`. `model.fit(X_train, y_train)`
`predict(X)`	Predict class labels or values for data `X`. `y_pred = model.predict(X_test)`
`transform(X)`	Apply dimensionality reduction or feature extraction to `X`. `X_transformed = model.transform(X)`
`fit_transform(X)`	Fit the model and then transform `X`. `X_transformed = model.fit_transform(X)`

Preprocessing and Feature Engineering

Scaling and Normalization

Scaling and normalization techniques adjust feature values to a standard range.

StandardScaler: Standardize features by removing the mean and scaling to unit variance.

```
from sklearn.preprocessing import
StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

MinMaxScaler: Scales features to a range between zero and one.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

RobustScaler: Scale features using statistics that are robust to outliers.

```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
X_scaled = scaler.fit_transform(X)
```

Encoding Categorical Variables

Encoding transforms categorical data into numerical format.

OneHotEncoder: Encodes categorical features as a one-hot numeric array.

```
from sklearn.preprocessing import
OneHotEncoder
encoder =
OneHotEncoder(handle_unknown='ignore')
X_encoded = encoder.fit_transform(X)
```

LabelEncoder: Encodes target labels with value between 0 and n_classes-1.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_encoded = le.fit_transform(y)
```

Imputation

`SimpleImputer` Fills missing values with a specified strategy (e.g., mean, median, most_frequent).

```
from sklearn.impute import
SimpleImputer
imputer =
SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
```

Model Evaluation and Validation

Metrics for Classification

Evaluation metrics quantify the performance of classification models.

Accuracy: Ratio of correctly predicted instances to total instances.

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
```

Precision: Ratio of true positives to the sum of true positives and false positives.

```
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
```

Recall: Ratio of true positives to the sum of true positives and false negatives.

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
```

F1-score: Weighted average of precision and recall.

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
```

Confusion Matrix: Table showing the correct and incorrect predictions, broken down by class.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Metrics for Regression

Evaluation metrics quantify the performance of regression models.

Mean Squared Error (MSE): Average of the squares of the errors.

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
```

Root Mean Squared Error (RMSE): Square root of the MSE.

```
import numpy as np
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

R-squared (Coefficient of Determination): Proportion of variance in the dependent variable that can be predicted from the independent variables.

```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
```

Cross-Validation

`cross_val_score` Evaluate a model by cross-validation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
```

`KFold` Provides train/test indices to split data in train/test sets.

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5, shuffle=True, random_state=42)
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

Pipeline and Grid Search

Pipeline

Pipelines streamline the sequence of data transformations and model fitting.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipe = Pipeline([('scaler', StandardScaler()), ('logistic', LogisticRegression())])
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```

Grid Search

Grid search systematically searches hyperparameter space for the best model.

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

param_grid = {'C': [0.1, 1, 10], 'gamma': [0.01, 0.1, 1]}
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=2)
grid.fit(X_train, y_train)
print(grid.best_estimator_)
```

Column Transformer

Apply different transformers to different columns of an array or pandas DataFrame.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

ct = ColumnTransformer([
    ('num', StandardScaler(), ['numerical_feature1', 'numerical_feature2']),
    ('cat', OneHotEncoder(), ['categorical_feature1', 'categorical_feature2'])
])

X_transformed = ct.fit_transform(X)
```