Scikit-learn Cheatsheet

Scikit-learn Standard Analysis Workflow

1. **Data Preparation**: Separate input features (`X`) and the prediction target (`y`).
2. **Model Selection**: Import the algorithm class suitable for the problem.
3. **Model Instantiation**: Create a model object by setting hyperparameters.
4. **Training**: Train the model using `model.fit(X, y)`.
5. **Prediction**: Generate predicted values for new data with `model.predict(X_new)`.
6. **Evaluation**: Precisely measure performance using various metrics.

Data Preprocessing

Use the `sklearn.preprocessing` module to improve data quality.

- **Scaling**:
  - `StandardScaler`: Standardize data to have mean 0 and variance 1.
  - `MinMaxScaler`: Compress data into the range [0, 1].
  - `RobustScaler`: Use median and quartiles to minimize the influence of outliers.
- **Encoding**:
  - `LabelEncoder`: Convert categorical text labels into integers.
  - `OneHotEncoder`: Convert categorical variables into one-hot vectors in sparse matrix form.
- **Imputation**:
  - `SimpleImputer`: Replace missing data with mean, median, mode, etc.

```
from sklearn.preprocessing import
StandardScaler
scaler = StandardScaler()
# Perform training and transformation
simultaneously
X_scaled = scaler.fit_transform(X)
```

Model Selection Guide

Supervised Learning

- **Regression**:
  - `LinearRegression`: The most basic linear regression model.
  - `Ridge`, `Lasso`: Linear models with regularization to prevent overfitting.
  - `SVR`: Regression model based on Support Vector Machines.
  - `RandomForestRegressor`: Powerful regression model based on decision tree ensembles.
- **Classification**:
  - `LogisticRegression`: Baseline model for binary and multiclass classification.
  - `SVC`: Support vector classifier that maximizes the boundary between classes.
  - `KNeighborsClassifier`: Classifies based on distance to neighbor data.
  - `RandomForestClassifier`: Combines multiple decision trees for improved performance.

Unsupervised Learning

- **Clustering**:
  - `KMeans`: Group data into K clusters.
  - `DBSCAN`: Density-based clustering to find clusters in space.
- **Dimensionality Reduction**:
  - `PCA`: Reduce dimensions while preserving as much variance as possible.
  - `TSNE`: Project high-dimensional data into lower dimensions for visualization.

Data Splitting and Cross-Validation

Improve the generalization performance of models via the `sklearn.model_selection` module.

- **Train/Test Split**:

```
from sklearn.model_selection import
train_test_split
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2,
random_state=42
)
```

- **Cross-Validation (CV)**:

```
from sklearn.model_selection import
cross_val_score
# Evaluate model stability with 5-fold
cross-validation
scores = cross_val_score(model, X, y,
cv=5)
```

Model Evaluation Metrics (`sklearn.metrics`)

Regression Metrics

- `mean_absolute_error` (MAE): Mean of absolute differences between actual and predicted.
- `mean_squared_error` (MSE): Mean of squared differences (sensitive to large errors).
- `r2_score`: Coefficient of determination; how much variance is explained by the model.

Classification Metrics

- `accuracy_score`: Ratio of correct predictions out of all data.
- `precision_score`: Ratio of actual positives among predicted positives.
- `recall_score`: Ratio of correctly identified positives among actual positives.
- `f1_score`: Harmonic mean of precision and recall.
- `roc_auc_score`: Discriminative ability across classification thresholds.

```
from sklearn.metrics import
accuracy_score, classification_report

y_pred = model.predict(X_test)
print(f"Accuracy:
{accuracy_score(y_test, y_pred)}")
# Check major metrics at a glance
print(classification_report(y_test,
y_pred))
```

Hyperparameter Optimization

- **Grid Search**: Exhaustively search through all specified parameter combinations.

```
from sklearn.model_selection import
GridSearchCV
param_grid = {'n_estimators': [100,
200], 'max_depth': [3, 5]}
grid_search = GridSearchCV(model,
param_grid, cv=5)
grid_search.fit(X_train, y_train)
print(f"Best Params:
{grid_search.best_params_}")
```

- **Randomized Search**: Efficiently explore parameters by trying random combinations.

Pipelines

Bundle preprocessing and training steps into one unit to manage code and prevent leakage mistakes.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import
StandardScaler
from sklearn.svm import SVC

# Bundle scaling and SVC training into
one unit
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC())
])

pipe.fit(X_train, y_train)
print(f"Test Score: {pipe.score(X_test,
y_test)}")
```