Machine Learning Algorithms Cheatsheet

1. Data Handling & Preprocessing

Importing from UCI ML Repository

```
from ucimlrepo import fetch_ucirepo

# Fetch dataset
dataset = fetch_ucirepo(id=<id>)

# Features and target
X = dataset.data.features
y = dataset.data.targets
```

Pandas / IO

```
df = pd.read_csv("<path>")
df.head()
df.info()
df.describe()
df.columns
df.shape
df.drop(columns=[<col_list>], inplace=<True/False>)
df.rename(columns={<old>:<new>,...}, inplace=<True/False>)
```

Feature Type Identification / Selection

```
feature_type = <data>.select_dtypes(
    include=<np.number / "object">
).columns.tolist()

# Select numeric columns
num_colmn = df.select_dtypes(
include=['int64', 'float64']
).columns

# binary columns
binary_cols = [col for col in df.columns if df[col].nunique() == 2]

# categorical columns
cat_cols = df.select_dtypes(include='object').columns.tolist()

# remove binary from cat
cat_cols = [col for col in cat_cols if col not in binary_cols]

# numeric columns
num_cols = df.select_dtypes(
include=['int64', 'float64']
).columns.tolist()
```

Nulls & Masks

```
any(df["col"].isna())
any(df["col"].isnull())
mask = df["col"].isna()
df["col"].isnull().sum()
```

Fill / Impute

```
df["col"].fillna(value=<value>, inplace=<True/False>)

from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy=
<"mean"/"median"/"most_frequent">)
X = imp.fit_transform(X)
```

Outlier Analysis and Removal

```
Q1 = <df>[<col>].quantile(0.25)
Q3 = <df>[<col>].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

<df> = <df>[(<df>[<col>] >= lower_bound) & (<df>[<col>] <= upper_bound)]
```

Descriptive

```
mean = df["col"].mean()
median = df["col"].median()
mode = df["col"].mode()
std = df["col"].std()
qk = df["col"].quantile(<k>)
```

Transformations

```
trans = np.log1p(df["col"])
orig = np.expm1(trans)          # inverse log1p
df["col"] = np.where(<condition>, <true_val>, <false_val>)
```

Scaling & Encoding

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import MaxAbsScaler, RobustScaler

X_std = StandardScaler().fit_transform(X)
X_min = MinMaxScaler().fit_transform(X)
X_maxabs = MaxAbsScaler().fit_transform(X)
X_robust = RobustScaler().fit_transform(X)

from sklearn.preprocessing import LabelBinarizer
lb = LabelBinarizer(
    neg_label=<0>, pos_label=<1>, sparse_output=<False>
)

y_bin = lb.fit_transform(y)
y_inv = lb.inverse_transform(y_bin)

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, LabelEncoder
X_le = LabelEncoder().fit_transform(X_cat)
X_ord = OrdinalEncoder().fit_transform(X_cat)
X_ord = OrdinalEncoder(categories=<categories_list>).fit_transform(X_cat)
X_oh = OneHotEncoder(
handle_unknown=<"ignore"/"error">
).fit_transform(X_cat).toarray()
```

Polynomial

```
from sklearn.preprocessing import PolynomialFeatures
X_poly = PolynomialFeatures(
degree=<d>, include_bias=<True/False>
).fit_transform(X)
```

Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=<float>, random_state=<int>, stratify=<y_or_None>
)
```

2. Supervised Learning Models (Syntactic)

Pipeline

```
from sklearn.pipeline import make_pipeline
pipe = make_pipeline(<step1>, <step2>, ...)
```

Linear & Logistic

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso,
LogisticRegression

LinearRegression()
Ridge(alpha=<alpha>)
Lasso(alpha=<alpha>)

LogisticRegression(
    penalty=<"l1"/"l2"/"elasticnet"/"none">,
    C=<float>,
    solver=<"liblinear"/"saga"/"lbfgs">,
    l1_ratio=<float_or_None>,
    random_state=<int>
)
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

DecisionTreeClassifier(
    criterion=<"gini"/"entropy">,
    max_depth=<int_or_None>,
    min_samples_split=<int>,
    min_samples_leaf=<int>,
    random_state=<int>
)

DecisionTreeRegressor(
    criterion=<"squared_error"/"friedman_mse"/"absolute_error">,
    max_depth=<int_or_None>,
    min_samples_split=<int>,
    random_state=<int>
)
```

Naive Bayes, KNN, SVM

```
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.svm import SVC, SVR

GaussianNB()
MultinomialNB()
BernoulliNB()

KNeighborsClassifier(n_neighbors=<int>, algorithm=
<"auto"/"ball_tree"/"kd_tree">)

SVC(kernel=<"linear"/"poly"/"rbf"/"sigmoid">,
    C=<float>, degree=<int>, gamma=<"scale"/"auto">,
    probability=<True/False>, random_state=<int>)

SVR(
    kernel=<"linear"/"poly"/"rbf"/"sigmoid">,
    C=<float>,
    degree=<int>,                  # for 'poly' kernel
    gamma=<"scale"/"auto">,        # kernel coefficient
    epsilon=<float>                # tolerance margin in regression
)
```

Bagging & RandomForest

```
from sklearn.ensemble import BaggingClassifier, BaggingRegressor,
RandomForestClassifier, RandomForestRegressor

BaggingClassifier(estimator=<est>, n_estimators=<int>, max_samples=<float>,
                  max_features=<float>, random_state=<int>)

RandomForestClassifier(n_estimators=<int>, criterion=<"gini"/"entropy">,
           max_depth=<int_or_None>, min_samples_split=<int>,
           max_features=<"sqrt"/"log2"/<float>>, random_state=<int>)

BaggingRegressor(...), RandomForestRegressor(...)
```

Boosting

```
from sklearn.ensemble import AdaBoostClassifier, AdaBoostRegressor,
GradientBoostingClassifier, GradientBoostingRegressor
from xgboost import XGBClassifier, XGBRegressor

AdaBoostClassifier(
    estimator=DecisionTreeClassifier(...),
    n_estimators=<int>, learning_rate=<float>, random_state=<int>
)

AdaBoostRegressor(...)

GradientBoostingClassifier(
    n_estimators=<int>, learning_rate=<float>, max_depth=<int>,
    min_samples_split=<int>, min_samples_leaf=<int>,
    subsample=<float>, max_features=<"sqrt"/"log2">, random_state=<int>
)

GradientBoostingRegressor(...)

XGBClassifier(
    use_label_encoder=<False/True>, eval_metric=<"logloss">,
    n_estimators=<int>, max_depth=<int>, learning_rate=<float>,
    subsample=<float>, random_state=<int>
)

XGBRegressor(...)
```

Stacking

```
from sklearn.ensemble import StackingClassifier, StackingRegressor

StackingClassifier(
    estimators=[("<name>", <estimator>), ...],
    final_estimator=<estimator>, cv=<int>, n_jobs=<int>
)

StackingRegressor(...)
```

General

```
model.fit(<X_train>, <y_train>)
y_pred = model.predict(<X_test>)
y_proba = model.predict_proba(<X_test>)[:, <class_index>]
score = model.score(<X_test>, <y_test>)
```

3. Unsupervised Learning Models (Syntactic)

K Means

```
from sklearn.cluster import KMeans

kmeans = KMeans(
    n_clusters=<int>, random_state=<int>
)

kmeans.fit(<X>)
wcss = kmeans.inertia_
labels = kmeans.labels_
centers = kmeans.cluster_centers_
```

4. Validation & Hyperparam Search

K-Fold & CV

```
from sklearn.model_selection import KFold, cross_val_score, cross_validate,
StratifiedKFold

KFold(n_splits=<int>, shuffle=<True/False>, random_state=<int>)

StratifiedKFold(
    n_splits=<int>,
    shuffle=<True/False>,
    random_state=<int>
)

cross_val_score(estimator=<model>,
X=<X>, y=<y>,
cv=<int_or_KFold>, scoring=<metric>,
return_train_score=<True/False>)
```

Grid / Randomized Search

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

search=GridSearchCV(estimator=<model>,
            param_grid={<param>: [<values>], ...},
            cv=<int_or_KFold>,
            scoring=<metric>,
            n_jobs=<int>)

search=RandomizedSearchCV(estimator=<model>,
                  param_distributions={<param>: [<values>], ...},
                  n_iter=<int>, cv=<int>, scoring=<metric>, n_jobs=<int>)

search.best_estimator_
search.cv_results_
search.best_score_
```

```
search.best_params_
```

5. Metrics & Visualization

Note: y_test = y_true

Classification Metrics

```
from sklearn.metrics import (
  accuracy_score, precision_score, recall_score, f1_score,
  confusion_matrix, classification_report, roc_auc_score,
  roc_curve, average_precision_score
)

accuracy_score(<y_true>, <y_pred>)
precision_score(<y_true>, <y_pred>,
average=<"binary"/"micro"/"macro"/"weighted">)
recall_score(<y_true>, <y_pred>)
f1_score(<y_true>, <y_pred>)
confusion_matrix(<y_true>, <y_pred>)
classification_report(<y_true>, <y_pred>)
```

Probabilistic / ROC

```
y_proba = model.predict_proba(<X_test>)[:, <class_index>]
fpr, tpr, thr = roc_curve(<y_true>, <y_proba>)

auc=roc_auc_score(<y_true>, <y_proba>)
```

Regression Metrics

```
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error

r2_score(<y_true>, <y_pred>)
adjusted_r2 = 1 - (1 - r2)*(n - 1)/(n - p - 1)
mean_squared_error(<y_true>, <y_pred>)
mean_absolute_error(<y_true>, <y_pred>)
```

Clustering Metrics

```
from sklearn.metrics import (
    silhouette_score,
    calinski_harabasz_score,
    davies_bouldin_score,
    adjusted_rand_score,
    normalized_mutual_info_score
)

silhouette_score(<X>, <labels>)
calinski_harabasz_score(<X>, <labels>)
davies_bouldin_score(<X>, <labels>)
adjusted_rand_score(<y_true>, <labels>)
normalized_mutual_info_score(<y_true>, <labels>)
```

Visualization (Syntactic)

```
sns.heatmap(<conf_matrix>, annot=<True/False>, fmt="<fmt>",
    cmap=<"cmap">, xticklabels=<labels>, yticklabels=<labels>)

fpr, tpr, thr = roc_curve(<y_true>, <y_score>)
plt.plot(fpr, tpr); plt.plot([0,1],[0,1], linestyle="<ls>")

sns.regplot(x=<y_true>, y=<y_pred>, ci=<int_or_None>)
sns.scatterplot(x=<x>, y=<y>, hue=<col_or_None>)
plt.bar(<feature_names>, <coef_or_importance>)

fig, axes = plt.subplots(
    nrows=<int>, ncols=<int>,
    figsize=(<width>, <height_per_row>*nrows)
)

plt.hist(
    <data>, bins=<int>,
    color=<"color">, edgecolor=<"color">
)

sns.scatterplot(
    x=<1d_X>, y=<1d_Y>,
    hue=<labels>,
    palette=<"deep"/"tab10"/"coolwarm">,
    legend=<"full"/"brief"/False>,
    s=<int>
)

sns.boxplot(
    data=<DataFrame>,
    x=<column>,
    ax=<axes[i]>
)

sns.histplot(
    data=<DataFrame>,
    x=<column>,
    bins=<int>,          # optional
    color=<"color">,     # optional
    edgecolor=<"color">, # optional
    ax=<axes[i]>
)

residuals = <y_true> - <y_pred>

plt.scatter(<y_pred>, residuals)
plt.axhline(0, color=<"color">, linestyle=<"style">)
```

6. Principal Component Analysis

```
from sklearn.decomposition import PCA

pca = PCA(
    n_components=<float_or_int>,     # e.g., 0.95 or 2
    random_state=<int>
)

X_train_pca = pca.fit_transform(X_train)
X_test_pca  = pca.transform(X_test)

pca.explained_variance_ratio_
pca.singular_values_
```

7. Statistical Significance Test

```
from scipy.stats import friedmanchisquare, f_oneway

model_scores = [
    [<score1>, <score2>, ...],  # Model 1
    [<score1>, <score2>, ...],  # Model 2
    ...                         # Model k
]


# Friedman Test (non-parametric)
friedman_stat, friedman_p = friedmanchisquare(*model_scores)

# One-way ANOVA Test (parametric)
anova_stat, anova_p = f_oneway(*model_scores)

if friedman_p < 0.05:
    print("Significant difference among models | proceed with post-hoc test")
else:
    print("No significant difference | all models perform similarly")

if anova_p < 0.05:
    print("At least one model differs significantly")
else:
    print("No significant difference detected")
```

8. Available Metric Strings (sklearn)

```
# Classification:
"accuracy", "precision", "recall", "f1",
"f1_micro", "f1_macro", "f1_weighted",
"precision_micro", "precision_macro", "precision_weighted",
"recall_micro", "recall_macro", "recall_weighted",
"roc_auc", "average_precision"

# Regression:
"r2",
"neg_mean_squared_error",
"neg_root_mean_squared_error",
"neg_mean_absolute_error",
"neg_median_absolute_error",
"explained_variance"
```

9. Miscellaneous (NumPy & Pandas)

```
# NumPy
np.sqrt(<x>), np.exp(<x>), np.log(<x>)
np.log1p(<x>), np.expm1(<x>)
np.mean(<x>), np.median(<x>), np.std(<x>), np.var(<x>)
np.max(<x>), np.min(<x>), np.argmax(<x>), np.argmin(<x>)
np.unique(<x>), np.dot(<a>,<b>)
np.linalg.inv(<A>), np.linalg.det(<A>)
np.corrcoef(<x>,<y>)
np.linspace(<start>,<stop>,<num>), np.arange(<start>,<stop>,<step>)
np.random.rand(<n>), np.random.randn(<n>), np.random.seed(<int>)
np.percentile(<data>,<q>)

# Pandas quick
df.head(<n>), df.tail(<n>), df.sample(<n>)
df.isnull().sum(), df.dropna(axis=<0_or_1>)
df.groupby(<cols>).agg({<col>: <func>})

Parameter Grid :

KNN:
n_neighbours: range(1,10)
algorithm="kd_tree" or "ball_tree"

SVM:
'C': [0.1, 1, 10],

gamma ; 'scale',auto only for rbf poly svm

degree 2,3  only for poly

Decession Tree:
'max_depth': [None, 3, 5, 7, 10],
 'min_samples_split': [2, 5, 10],
 'min_samples_leaf': [1, 2, 4],
 'criterion': ['gini', 'entropy']

AdaBoost:
n_estimators': [50, 100, 200],
'learning_rate': [0.01, 0.1, 1],
'estimator':[
 DecisionTreeClassifier(max_depth=1, random_state=42),
 DecisionTreeClassifier(max_depth=2, random_state=42)]

Gradient Boosting :
'n_estimators': [50, 100, 200],
'learning_rate': [0.01, 0.1, 0.2],
'max_depth': [1, 3, 5],
'subsample': [0.8, 1.0]

XGBoost
'n_estimators': [50, 100, 200],
'learning_rate': [0.01, 0.1, 0.2],
 'max_depth': [3, 5, 7],
 'gamma': [0, 0.1, 0.3],
 'subsample': [0.8, 1.0],
 'colsample_bytree': [0.8, 1.0]

Random Forest :
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'criterion': ['gini', 'entropy'],
    'max_features': ['sqrt', 'log2'],
    'min_samples_split': [2, 5, 10]

grid_search = GridSearchCV(
    estimator=dt,
    param_grid=param_grid,
    cv=5,                # 5-fold cross validation
    scoring='accuracy', # Metric to optimize
    n_jobs=-1            # Use all CPU cores
)

grid_search.fit(x_train, y_train)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)
print("best parameters :",grid_search.best_params_)
print("best score :",grid_search.best_score_)
dtaccuracy=accuracy_score(y_test,y_pred)

results_df = pd.DataFrame(grid_search.cv_results_)
best_idx = grid_search.best_index_
print("\nFold-wise accuracy for best parameters:")
print(results_df.loc[best_idx, [c for c in results_df.columns if
c.startswith('split')]])

Post-hoc testing:

model_means = [np.mean(scores) for scores in model_scores]
best_index = np.argmax(model_means)  # or np.argmin if lower is better
print(f"Best model: Model {best_index + 1} with mean score =
{model_means[best_index]:.4f}")
```