Cheat Sheet: Linear and Logistic Regression

Comparing different regression types

Model Name	Description	Code Syntax
Simple linear regression	**Purpose:** To predict a dependent variable based on one independent variable. **Pros:** Easy to implement, interpret, and efficient for small datasets. **Cons:** Not suitable for complex relationships; prone to underfitting. **Modeling equation:** $y = b_0 + b_1x$	```from sklearn.linear_model import LinearRegression model = LinearRegression() model.fit(X, y)```
Polynomial regression	**Purpose:** To capture nonlinear relationships between variables. **Pros:** Better at fitting nonlinear data compared to linear regression. **Cons:** Prone to overfitting with high-degree polynomials. **Modeling equation:** $y = b_0 + b_1x + b_2x^2 + ...$	```from sklearn.preprocessing import PolynomialFeatures from sklearn.linear_model import LinearRegression poly = PolynomialFeatures(degree=2) X_poly = poly.fit_transform(X) model = LinearRegression().fit(X_poly, y)```
Multiple linear regression	**Purpose:** To predict a dependent variable based on multiple independent variables. **Pros:** Accounts for multiple factors influencing the outcome. **Cons:** Assumes a linear relationship between predictors and target. **Modeling equation:** $y = b_0 + b_1x_1 + b_2x_2 + ...$	```from sklearn.linear_model import LinearRegression model = LinearRegression() model.fit(X, y)```
Logistic regression	**Purpose:** To predict probabilities of categorical outcomes. **Pros:** Efficient for binary classification problems. **Cons:** Assumes a linear relationship between independent variables and log-odds. **Modeling equation:** $\log(p/(1-p)) = b_0 + b_1x_1 + ...$	```from sklearn.linear_model import LogisticRegression model = LogisticRegression() model.fit(X, y)```

Associated functions commonly used

Function/Method Name	Brief Description	Code Syntax
train_test_split	Splits the dataset into training and testing subsets to evaluate the model's performance.	```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)```
StandardScaler	Standardizes features by removing the mean and scaling to unit variance.	```from sklearn.preprocessing import StandardScaler scaler = StandardScaler() X_scaled = scaler.fit_transform(X)```
log_loss	Calculates the logarithmic loss,	```from sklearn.metrics import log_loss loss = log_loss(y_true, y_pred_proba)```

Function/Method Name	Brief Description	Code Syntax
	a performance metric for classification models.	
mean_absolute_error	Calculates the mean absolute error between actual and predicted values.	``` from sklearn.metrics import mean_absolute_error mae = mean_absolute_error(y_true, y_pred) ```
mean_squared_error	Computes the mean squared error between actual and predicted values.	``` from sklearn.metrics import mean_squared_error mse = mean_squared_error(y_true, y_pred) ```
root_mean_squared_error	Calculates the root mean squared error (RMSE), a commonly used metric for regression tasks.	``` from sklearn.metrics import mean_squared_error import numpy as np rmse = np.sqrt(mean_squared_error(y_true, y_pred)) ```
r2_score	Computes the R-squared value, indicating how well the model explains the variability of the target variable.	``` from sklearn.metrics import r2_score r2 = r2_score(y_true, y_pred) ```

Author(s)

Jeff Grossman
Abhishek Gagneja