Supervised Learning
Regression

Regression vs Classification
Business applications (by task, by sector)
Terminology: from variable to grid search
Mathemical explanation. Linear and Non-linear
Data requirements for core regression models
The process: initiation - evaluation - fine-tuning
Python code for each stage of the process

Julia Lenc

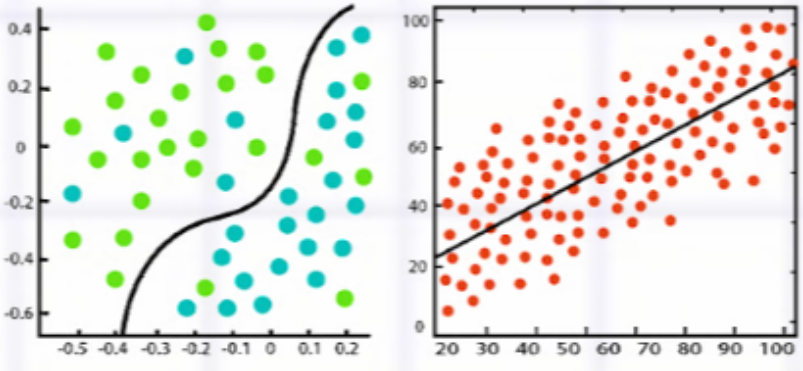
Supervised Learning Models

Regression

Prediction of a continuous value - number or quantity.
Key questions: "How much?", "What will the value be?"
Popular applications: sales forecast, budget changes, pricing

Classification

Assignment of a probability or a category to an observation.
Key questions: "Which group?", "How likely?", “Is it A or B?”
Popular applications: churn prediction, recommender systems



Classification Regression

Julia Lenc

Business Applications

Areas

🧩 **Value Creation**: demand forecasting, market research (“synthetic consumer”).

💰 **Revenue Model**: promo efficiency, price elasticity, customer lifetime value (CLV).

🔍 **Market Opportunity**: market size forecast, pockets of growth, stores or leads potential.

🛒 **Go-to-Market**: omnichannel strategies marketing effectiveness, trade fundamentals (price, promo).

🏬 **Operations**: inventory and stocks optimization, transportation cost modeling, demand forecasts, stock disruptions, customerservice (virtual assistants).

Julia Lenc

Business Applications

Consumer sectors

🛍️ **FMCG and Mass Fashion**: demand forecasting, marketing mix optimization, price and promo modeling.

🛒 **Retail:** revenue forecasting, footfall projection, customer spend prediction, basket size modeling.

🚗 **Durables:** replacement cycle forecasting, warranty prediction, price elasticity, cross-sell.

✈️ **Travel:** occupancy rates prediction, personalized offers, dynamic pricing.

💰 **Banking and Insurance:** loan amount estimate, insurance claims forecasting, individual premiums calculation, customer lifetime value.

📶 **Telecom:** data usage modeling, ARPU (average revenue per user) prediction, customer lifetime value.

Julia Lenc

Business Applications

Business-to-Business sector

🏭 **Industrial products and services**: sales opportunity and market potential modeling, customer lifetime value.

💻 **SaaS**: Subscription revenue forecast, usage prediction e.g., storage, computing.

📽️ **Free platforms and search engines**: Ad revenue forecasting, user engagement projection (time, views).

🏠 **Sharing and Gig economy. Marketplaces:** ride-share demand, occupancy rate prediction, e.g., Uber, Airbnb.

Julia Lenc

Terminology

Foundations of Regression

- **Dependent variable**: the outcome we aim to predict (e.g., sales revenue, temperature). Target. Output.
- **Independent variable**: the input(s) used to predict the dependent variable (e.g., ad spend, time of year).
- **Constant (Alpha, α, intercept):** the baseline value of the dependent variable when all independent variables are set to zero.Y-intercept of the regression line.
- **Coefficient (Beta, β, slope):** describes how much the dependent variable changes per unit increase in an independent variable. Positive or negative signs indicate the relationship's direction.

Julia Lenc

Terminology

Evaluating Model Fit

- **Redisual**: The difference between the observed and predicted dependent variable values (aka "error").
- **Square Sum of Errors (SSE)**: the sum of squared residuals. Measures the total error. Low SSE indicates a better model fit.
- **R^2 (Coefficient of Determination):** indication of how much of the variance in dependent variable is explained by independent variables. Range: 0 (no fit) - 1 (perfect fit).
- **Cost function:** a formula to measure error across all predictions. MAE, MSE, RSME, MAPE (see: Evaluation).

Julia Lenc

Terminology

Key Statistical Assumptions

- **Homoscedasticity**: the residuals have a constant variance across all levels of the independent variables. This means the spread of prediction errors doesn't change
- **Multicollinearity:** when two or more independent variables are highly correlated, it undermines their individual contributions to predicting the dependent.

Overfitting

- **Overfitting:** when a model is too complex and fits the noise in the training data, it performs poorly on new or unseen data (low generalizability).
- **Regularization L1 and L2**: techniques to reduce overfitting by penalizing large coefficients. L1 (Lasso) shrinks some coefficients to zero, L2 (Ridge) reduces their magnitude.

Julia Lenc

Terminology

Optimization techniques

- **Gradient descent**: an iterative optimization algorithm that adjusts the model's coefficients to minimize the cost function (error). Steps are taken in the direction of steepest descent (negative gradient).
- **Grid search:** a process to optimize hyperparameters by systematically testing all possible combinations and selecting the best-performing set.

Julia Lenc

Mathematical explanation

$$y = a + bx$$

Intercept Slope

Linear regression is setting the data (x) into a line (y).

y : Dependent variable (outcome, target)

x : Independent variable (input)

α : Intercept ß : Slope

Multifactor linear regression

$y = \alpha + ß1*x1 + ß2*x2 + ... + ß2*x2$

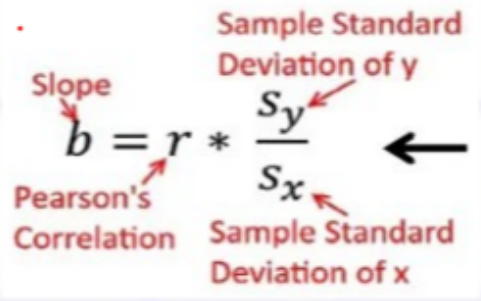
x1, ..., xn : independent variables

ß1, ..., ßn: coefficients estimating impact of each x on y

Visuals: www.FairlyNerdy.com

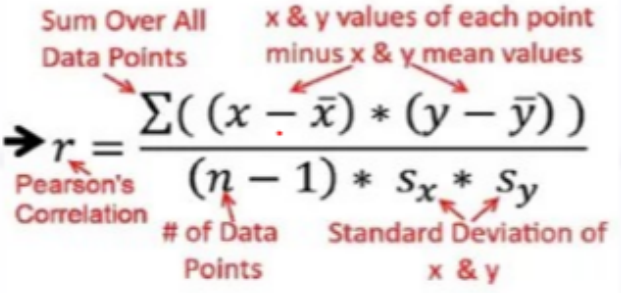
Julia Lenc

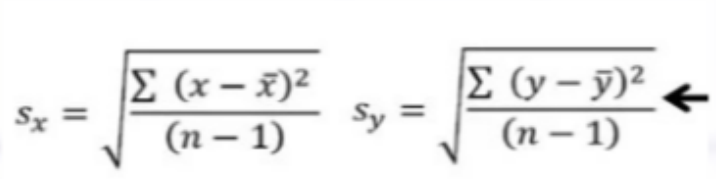
Mathematical explanation



Slope shows impact of x on y and depends on xy correlation and the difference in standard deviations.

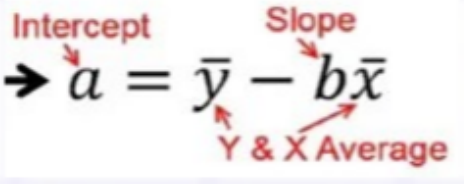
Correlation shows how much x and y change together. Can be positive and negative. Range [-1, 1].





Sample **standard deviations** show the spread of values around x and y means.

Intercept is the point where the line crosses y-axis (x is zero).



Visuals: www.FairlyNerdy.com

Julia Lenc

Other types of regression

Polynomial

Fits a curve (e.g., quadradic, cubic) to data, not a straight line.

Business application: **stock price prediction, profit modeling**

$$\hat{y} = \beta_0 x^0 + \beta_1 x^1 + \beta_2 x^2 + \cdots + \beta_m x^m$$

n = number of $x_i y_i$ variable pairs in the data
m = order of the polynomial to be used for regression
$\beta_{(0-m)}$ = polynomial coefficient for each corresponding $x^{(0-m)}$
$\hat{y}$ = estimated y variable based on the polynomial regression calculations.

Quantile

Predicts specific quantiles (e.g., median). Estimates conditional quantile relationships. Better for skewed, non-normal distributions.

Business application: **healthcare median costs.**

Poisson

Models count data or event rates.

Business application: **call center events, purchase frequency modeling.**

Visuals: Wikipedia

Julia Lenc

Data Requirements

	Linear Regression	Non-Linear Regression
Linearity	🔴	🟡
Normal distribution	🟡	🟡
No outliers *	🟡	🟡
Independence of errors	🟡	🟡
Constant spread of errors	🔴	🟡 (polynomial)
Independence of features	🔴	🔴

Legend:

🔴 Must-Have: essential for the method to give valid results.

🟡 Desirable: desirable for better performance or interpretation.

* If outliers are legit (not errors), data scientist decides whether to keep. E.g., having a 20 room villa in a sample of 2 bedroom apartments might disturb house prices modeling.

Julia Lenc

The process

Pre-requisites

- Business question is understood and formulated as
  - **S**pecific (problem to solve)
  - **M**easurable (success criteria)
  - **A**chievable (data and capabilities in place)
  - **R**elevant (answers will lead to action)
  - **T**ime-bound (stakeholders and deadline are known).
- Data meets requirements and is pre-processed.
  See here how to prepare the data.

Modeling process

- This covers only modeling process, not production. End point: deployment or decision to increase model complexity.
  1. Initial model set up
  2. First evaluation
  3. If meets evaluation criteria → check overfit
     If overfit → try regularization
  4. If the model don't meet criteria at initial evaluation or after regularization doesn't help → check pre-processing (incl. feature engineering)
  5. If doen't help → increase model complexity (Decision Tree or SVR)

Julia Lenc

1. Model initialization

Objectives

1. Import necessary libraries
2. Loading the dataset
3. Split the dataset into training and testing sets so the model can learn from one part (training – 80% of the dataset) and be evaluated on another unseen part (testing – 20% of the dataset) to ensure reliable performance.

```
# Step 1: Import libraries and load data
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Load the dataset
data = pd.read_csv('your_file.csv')  # Replace 'your_file.csv' with the actual file path

X = data.drop(columns='target')  # Features
y = data['target']  # Target variable

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Explanation:
# Setting random_state=42 ensures consistent results across runs by fixing the randomness.
# Without it, splits will be random each time, leading to potential inconsistencies in results.
```

Julia Lenc

2. Initial run and evaluation

Objectives

1. Modeling (here: linear regression)
2. Evaluating model performance. See here which evaluation metric to choose

```
# Step 2: Fit the model and make predictions
model = LinearRegression()  # Initialize the linear regression model
model.fit(X_train, y_train)  # Train the model on the training data

# Predictions
y_pred = model.predict(X_test)  # Make predictions on the test data

# Function for MAPE (Mean Absolute Percentage Error)
def mean_absolute_percentage_error(y_true, y_pred):
    return 100 * (abs((y_true - y_pred) / y_true).mean())

# Initial evaluation. Set the most relevant evaluation metric
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
rmse = mse ** 0.5  # Root Mean Squared Error
mape = mean_absolute_percentage_error(y_test, y_pred)  # Mean Absolute Percentage Error
r2 = r2_score(y_test, y_pred)  # R² Score

# Print evaluation metrics
print(f"MAE: {mae:.2f}, MSE: {mse:.2f}, RMSE: {rmse:.2f}, MAPE: {mape:.2f}%, R²: {r2:.2f}")

# Explanation:
# - MAE (Mean Absolute Error): Average of absolute differences between true and predicted values.
# - MSE (Mean Squared Error): Average of squared differences. Penalizes large errors.
# - RMSE (Root Mean Squared Error): Square root of MSE, same unit as the target.
# - MAPE (Mean Absolute Percentage Error): Percentage-based error to assess relative accuracy.
# - R² (R-squared): Explains the proportion of variance captured by the model (closer to 1 is best).
```

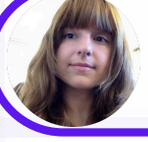
Julia Lenc

3a. Overfit check

Objectives

If the model performs much worse on test data than on training data, this is a signal of overfit. Click here to learn about overfit.

```
# Step 2: Fit the model and make predictions
model = LinearRegression()  # Initialize the linear regression model
model.fit(X_train, y_train)  # Train the model on the training data

# Predictions
y_pred = model.predict(X_test)  # Make predictions on the test data

# Function for MAPE (Mean Absolute Percentage Error)
def mean_absolute_percentage_error(y_true, y_pred):
    return 100 * (abs((y_true - y_pred) / y_true).mean())

# Initial evaluation. Set the most relevant evaluation metric
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
rmse = mse ** 0.5  # Root Mean Squared Error
mape = mean_absolute_percentage_error(y_test, y_pred)  # Mean Absolute Percentage Error
r2 = r2_score(y_test, y_pred)  # R² Score

# Print evaluation metrics
print(f"MAE: {mae:.2f}, MSE: {mse:.2f}, RMSE: {rmse:.2f}, MAPE: {mape:.2f}%, R²: {r2:.2f}")

# Explanation:
# - MAE (Mean Absolute Error): Average of absolute differences between true and predicted values.
# - MSE (Mean Squared Error): Average of squared differences. Penalizes large errors.
# - RMSE (Root Mean Squared Error): Square root of MSE, same unit as the target.
# - MAPE (Mean Absolute Percentage Error): Percentage-based error to assess relative accuracy.
# - R² (R-squared): Explains the proportion of variance captured by the model (closer to 1 is best).
```



Julia Lenc

3a. Regularization

Objectives

Try L1 and L2 regularization. Re-evaluate the model to see whether overfit decreases. Click here to learn about regularization.

```
# Step 3aa: Address overfitting with regularization. Compare training and test MSE.

from sklearn.linear_model import Ridge, Lasso

# 1. Train the original model (for comparison)
original_train_mse = mean_squared_error(y_train, model.predict(X_train))
original_test_mse = mean_squared_error(y_test, y_pred)

print("Original Model:")
print(f"Training MSE: {original_train_mse:.2f}")
print(f"Test MSE: {original_test_mse:.2f}")

# 2. Ridge Regularization (L2)
ridge_model = Ridge(alpha=1.0)  # Alpha is the regularization strength
ridge_model.fit(X_train, y_train)

ridge_train_mse = mean_squared_error(y_train, ridge_model.predict(X_train))
ridge_test_mse = mean_squared_error(y_test, ridge_model.predict(X_test))

print("\nRidge Regularization:")
print(f"Training MSE: {ridge_train_mse:.2f}")
print(f"Test MSE: {ridge_test_mse:.2f}")

# 3. Lasso Regularization (L1)
lasso_model = Lasso(alpha=0.1)  # Alpha is the regularization strength
lasso_model.fit(X_train, y_train)

lasso_train_mse = mean_squared_error(y_train, lasso_model.predict(X_train))
lasso_test_mse = mean_squared_error(y_test, lasso_model.predict(X_test))

print("\nLasso Regularization:")
print(f"Training MSE: {lasso_train_mse:.2f}")
print(f"Test MSE: {lasso_test_mse:.2f}")

# Explanation:
# - Look for reduced gap between training and test MSE after each regularization.
# - Choose the regularization (if any) that optimizes generalization (i.e., reduces overfitting).
# - Alpha can be tuned further for better performance.
```

Julia Lenc

3 and 4. Pre-processing?

Objectives

Check residual for randomness. If residuals are not random, go back to pre-processing and check whether model assumptions are met (next slide).

```
import matplotlib.pyplot as plt
import scipy.stats as stats
import numpy as np

# Step 3ab: Check residuals for randomness
# Residuals are calculated as the difference between true and predicted values
residuals = y_test - model.predict(X_test)

# 1. Visualization: Residual Plot
plt.figure(figsize=(8, 6))
plt.scatter(model.predict(X_test), residuals, alpha=0.7, edgecolor='k')
plt.axhline(0, color='red', linestyle='--', linewidth=1)
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()

# 2. Statistical Test: Test for randomness using the Durbin-Watson statistic
from statsmodels.stats.stattools import durbin_watson

dw_stat = durbin_watson(residuals)
print(f"Durbin-Watson Statistic: {dw_stat:.2f}")

# Interpretation:
# - Residuals should ideally be scattered randomly around the horizontal line at zero.
# - Durbin-Watson Statistic ≈ 2 indicates no autocorrelation (random residuals).
# - Stat < 2: Positive autocorrelation (residuals are not independent).
# - Stat > 2: Negative autocorrelation.
```

Julia Lenc

3 and 4. Assumption check

Objectives

Check assumptions for linear regression: linearity, no outliers, heteroscedasticity, no multicollinearity.

```python
# Visualize the relationship between predictors and the target
for feature in X.columns:
    plt.figure(figsize=(8, 6))
    plt.scatter(X[feature], y, alpha=0.7, edgecolor='k')
    plt.xlabel(feature)
    plt.ylabel("Target")
    plt.title(f"Scatter Plot: {feature} vs Target")
    plt.show()
```

```python
from sklearn.ensemble import IsolationForest

# Detect outliers using Isolation Forest (alternative: Z-score or IQR)
iso_forest = IsolationForest(contamination=0.05, random_state=42)
outlier_flags = iso_forest.fit_predict(X)  # -1 indicates an outlier

# Count detected outliers
print(f"Number of suspected outliers: {sum(outlier_flags == -1)}")
```

```python
# Breusch-Pagan test for heteroscedasticity
from statsmodels.stats.diagnostic import het_breuschpagan
import statsmodels.api as sm

lm_test = sm.OLS(residuals, sm.add_constant(model.predict(X_test))).fit()
bp_test = het_breuschpagan(lm_test.resid, lm_test.model.exog)
print(f"Breusch-Pagan p-value: {bp_test[1]:.3f}")

# Interpretation:
# - If p-value < 0.05, heteroscedasticity is present (non-constant residual variance).
```

```python
import seaborn as sns

# Plot a correlation matrix
cor_matrix = X.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(cor_matrix, annot=True, cmap="coolwarm")
plt.title("Feature Correlation Matrix")
plt.show()

# Calculate Variance Inflation Factor (VIF) for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif_data)

# Interpretation:
# - VIF > 5 (or 10) indicates significant multicollinearity.
```

Julia Lenc

3 and 4. Feature engineering

Examples

Log transformation. Polynomial transformation. Feature removal and addition.

```
from sklearn.preprocessing import PolynomialFeatures
from numpy import log1p  # log1p(x) = log(1 + x), better for skewed data and values close to 0

# Log transformation of a skewed feature
X['log_feature'] = log1p(X['feature_to_transform'])

# Polynomial feature generation
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X[['feature_to_transform']])  # Generates x, x²
print(f"Polynomial Features Shape: {X_poly.shape}")
```

```
# Drop irrelevant or problematic features
X = X.drop(columns=['feature_to_remove'])
print("Removed feature: feature_to_remove")
```

```
# Replace 'absolute price' with 'price index'
X['price_index_vs_brand_A'] = X['price'] / X['competitor_price']
X = X.drop(columns=['price'])  # Drop the original price column
print("Added 'price_index_vs_brand_A' and removed 'price'")
```

Julia Lenc

5. Final model evaluation

Examples

Check whether changes have improved evaluation scores and whether the model meets evaluation criteria now. If not - consider increasing its complexity.

```
# Refit the transformed model
refined_model = LinearRegression()
refined_model.fit(X_train, y_train)

# Predictions and performance
original_y_pred = model.predict(X_test)
refined_y_pred = refined_model.predict(X_test)

# Calculate and compare metrics
original_mse = mean_squared_error(y_test, original_y_pred)
refined_mse = mean_squared_error(y_test, refined_y_pred)

print("Comparison of Model Performance:")
print(f"Original Model MSE: {original_mse:.2f}")
print(f"Refined Model MSE: {refined_mse:.2f}")

# Interpretation:
# - A lower MSE in the refined model suggests that the transformations improved performance.
```

Julia Lenc

Did you find it useful?

Save

Share

Follow

Analytics, Market Research,
Machine Learning and AI

Julia Lenc