Supervised Learning Classification

Regression vs Classification
Business applications
Models: Logistic Regression, Naive Bayes, k-NNs
Terminology: from variable to early stopping
Mathemical explanation for each of core models
Model choice based on on business questions and data
Initiation - evaluation - fitting - reprocessing - fine-tuning
Python code for each stage of the process

Julia Lenc

Analytics Journey

Business Analytics (BA)

1. Intro: Business and Revenue models. KPIs
2. Business models translated into analytics
3. Techniques: Descriptive, Diagnostic, Predictive, Prescriptive

Diagnostic Techniques

1. Inference: hypotheses testing
2. Unsupervised Learning: clustering, dimensionality reduction, anomalies

Predictive Techniques

1. Supervised learning: overview
2. Preparation: data pre-processing
3. Foundations: model choice and evaluation
4. Regression: linear and non-linear
5. Classification: logistic regression, Naive Bayes, k-NNs (this presentation!)
6. Time series: ARIMA, SARIMA, Exponential Smoothing
7. Advanced: Decision Trees, SVM
8. Ensemble: bagging, boosting, stacking
9. Neural Networks: FFNN, CNN, RNN, Transformers

Prescriptive Techniques

1. Optimization: Linear, Non-linear and Dynamic programming
2. Simulation: Monte Carlo, Discreet Events, System Dynamics
3. Probabilistic Sequence: Markov Chains, Markov Decision Processes
4. Reinforcement Learning: Q-Learning, Deep RL, Policy Gradient

Julia Lenc

Classification

Intro

1. Classification vs regression
2. Core models: logistic regression, Naive Bayes, k-NNs
3. Business applications (by business model elements)

Math

1. Terminology
2. Mathematical explanation

Statistics

1. Model selection: business questions, data requirements
2. Training and validation process
3. Evaluation criteria

Python

Step-by-step process in scikit-learn

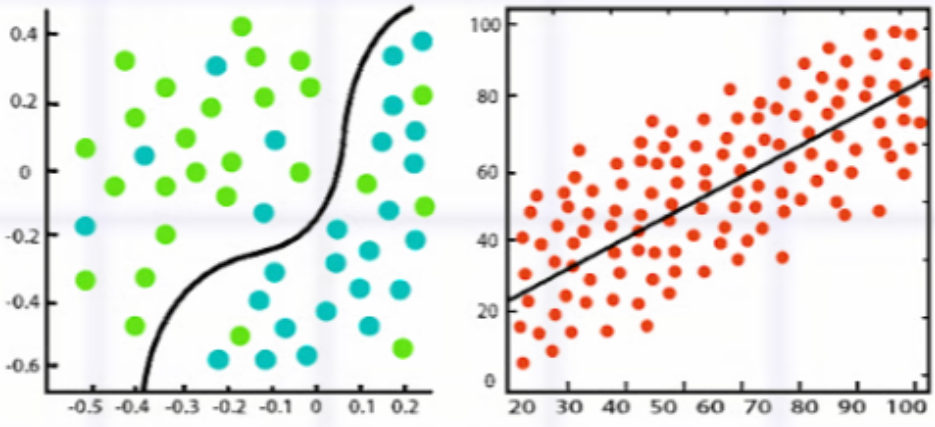
Classification vs Regression

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**

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 Regression


Julia Lenc

Core Classification Models

Logistic Regression

The **likelihood** of belonging to a category. Highly interpretable. Coefficients show the impact of variables on the prediction.
Popular applications: **churn, recommenders**

Naive Bayes

The most likely **class.** Predicts the relationship between the features of an observation and class.
Popular applications: **text analysis** (sentiment, spam/not spam)

k-Nearest Neighbours (k-NNs)

The most likely **class**. No need for a training or assumptions about data distribution.
Popular applications: **targeted marketing, recommenders**

Julia Lenc

Business Applications

By Business Model elements

- **Value Creation**: churn prediction (LR), sentiment analysis (NB), lightweight recommender system by customer segment (k-NNs)
- **Revenue Model**: discount personalization (LR), upselling (k-NNs)
- **Market Opportunity**: likehood of new product adoption by customer segment (LR), customer / user lifestage classifiacion (NB), product-market-fit prediction by customer segment (k-NNs)
- **Go-to-Market**: personalized messaging by likehood to react (LR), predict loalty card adoption (NB), classifiying new and upcoming stores as premium / mid / low tier shoppers (k-NNs)
- **Operations**: classify risk levels (LR), route support tickets - urgent/not urgent (NB), categorize returns (k-NNs)

Legend:

LR - Logistic Regression, NB - Naive Bayes, k-NN - k-Nearest Neighbours

Julia Lenc

Terminology

Foundations of Classification

- **Dependent variable - DV**: the categorical outcome we aim to predict (e.g., "churn" or "no churn").
- **Independent variable - IV**: the input features used to predict the dependent variable (e.g., email content, credit scores).
- **Class label:** each unique category in the DV (e.g., "Yes", "No").
- **Logistic function:** the s-shaped curve that maps predicted values to a probability in the range [0, 1].
- **Probability threshold:** a cutoff value (commonly 0.5) to classify probabilities into one of the class labels.
- **Prior probability:** In Naive Bayes, the proportion of each class in the dataset before observing any features.

Terminology

Evaluating model fit

- **Confusion matrix**: a table summarizing predictions vs actual values: True Positive - TP, True Negative - TN, False Negative - FN, False Positive - FP.
- **Accuracy**: the proportion of correct predictions = (TP + TN) / Total Observations.
- **Precision:** the proportion of correctly predicted positives out of all predicted positives = TP / (TP + FP).
- **Recall:** the proportion of actual positives identified by the model = TP / (TP + FN).
- **F1 score:** the harmonic mean of precision and recall = 2 × (Precision × Recall) / (Precision + Recall).
- **ROC curve (AUC):** a plot of true positive rate vs false positive rate across thresholds. AUC measures overall performance.
- **Log Loss:** a cost function used to evaluate probabilistic models, penalizing incorrect predictions heavily.

Terminology

Key statistical concepts

- **Linerally separable**: when a straight line or hyperplane can separate classes (important for logistic regression).
- **Conditional independence**: Naive Bayes assumes that features are independent given the class label.
- **k in k-NNs:** refers to the number of nearest neighbors used to determine a sample's class.

Terminology

Common challenges

- **Overfitting**: the model is too complex and learns noise or patterns specific to training data.
- **Underfitting**: the model is too simple and fails to capture meaningful patterns in the data.
- **Threshold selection:** the choice of probability cutoff (e.g., 0.5) can affect metrics like precision and recall.
- **Imbalanced classes:** when one class significantly outnumbers the other(s), leading to biased predictions.
- **Feature scaling:** essential for k-NN to ensure that all features contribute equally to distance calculations.

Terminology

Optimization techniques

- **Gradient descent**: used in logistic regression to find the coefficients that minimize log loss.
- **Hyperparameter tuning**: adjusting parameters (e.g., number of neighbors in k-NN, alpha in Logistic Regression L1).
- **Cross-validation:** splitting the dataset into folds to evaluate the model's performance.
- **Grid search:** systematic approach to optimizing hyperparameters.
- **Early stopping:** terminating training when performance on validation data stops improving.

Math: Logistic Regression

$$P(y = 1 \mid X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n)}}$$

P(y=1 | X) : predicted probability that dependent variable y equals 1 ("positive" class).

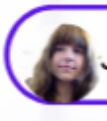
βO : the intercept; the baseline log-odds of the positive class when all features are 0.

β1,β2,...,βn : the coefficients for each feature, describing how strongly each feature contributes to the log-odds of the positive class. A positive coefficient increases the probability of y=1 , while a negative coefficient decreases it.

X1,X2,...,Xn : the independent variables (features) used to predict the dependent variable. These are the input values for the model.

e(...) : the exponential function, which transforms the log-odds into probabilities, ensuring the output probability is between 0 and 1.

1+e−(...) : the normalizing denominator, ensuring that the output probability P(y=1) satisfies $0 \leq P(y=1) \leq 1$.



Julia Lenc

Math: Naive Bayes

$$P(y \mid X) = \frac{P(X \mid y) \cdot P(y)}{P(X)}$$

P(y | X) : Posterior probability (output).

P(X | y) : Likelihood of features for a given class.

P(y) : Prior probability of the class.

P(X) : Evidence, often ignored when comparing classes.

Prediction rule: class with highest posterior probability is chosen as the prediction

$$\hat{y} = \arg\max_{y} \left(P(X \mid y) \cdot P(y) \right)$$

Example: **Sentiment analysis**

"The movie was great but a little long"

X: input (bag of words, words frequencies, encoded features)

P(positive | X)=0.15

P(neutral | X)=0.07

P(negative | X)=0.03

Predicted Sentiment: Positive


Julia Lenc

Math: k-Nearest Neighbours

$$d(X, X_i) = \sqrt{\sum_{j=1}^{n} (X_j - X_{ij})^2}$$

1. The distance is calculated between data point X and each point in the dataset.

   Here: **Euclidean distance** (most popular). Also: Manhattan Distance, Cosine Similarity

   X : Input sample or data point to classify.

   Xi : A training sample in the dataset.

   Xj and Xij : Features of X and Xi at index j .

2. **k closest training points** are selected - those with the smallest distances from X

   Sort all training data points by distance to X. Select the top k neighbors.

   Warning! small k = sensitive to noise, large k = better generalization but may dilute class boundaries.

3. **The class with the most votes = prediction.**

   Example: if 3 neighbors: 2 positive, 1 neutral → Predicted class = Positive.

Julia Lenc

Selecting the model

By business question type:

"Why something is happening?" → Logistic Regression

"Why do customers churn? (Price, promotions, competitive offers...)

Feature weights (coefficients) inform about the magnitude and direction of influence.

"What category does this belong to?" → Naive Bayes

"Is this review positive, negative or neutral?"

"Which insurance plan fits this customer?"

"What is the most similar or closest to this?" → k-NNs

"Which restaurant is physically nearest or most similar to the one a customer likes?

"Which flights are most similar to the one this traveler is looking for?

Remember about data requirements:

	Logistic Regression	Naive Bayes	k-NNs
Independence of errors	🔴	🔴	🟡
Independence of features	🔴	🟡	🟡
No outliers	🟡	🟡	🔴

Legend:
🔴 Must-Have: essential for the method to give valid results.
🟡 Desirable: desirable for better performance or interpretation.


Julia Lenc

The process: Stage 1

1. Understand and formulate the business question as:

Specific: Define the problem clearly (e.g., classification vs. regression tasks).

Measurable: Determine evaluation criteria (e.g., accuracy, precision / recall).

Achievable: Ensure data and resources match the modeling goals.

Relevant: Validate that answering this question will drive action.

Time-bound: Identify stakeholders and key deadlines for delivery.

2. Prepare the data to meet requirements:

Read here about 8 stages of data preparation to get perfect input

3. Choose the model and set evaluation criteria:

Read here about confusion matrix, evaluation criteria and "precision vs recall"

Julia Lenc

The process: Stage 2

1. Initial Model Setup and Run:

Choose appropriate algorithms and their initial hyperparameters:

Logistic Regression: Regularization (L1/L2), learning rate.

Naive Bayes: Type of NB (e.g., Gaussian, Multinomial, Bernoulli).

k-NNs: Number of neighbors (k), distance metric (e.g., Euclidean).

---

2. Initial Model Setup and Run:

Performance evaluated on validation set according to aligned criteria. Key checks:

Logistic Regression: check for underfitting/poor performance due to mismatched features or class imbalance.

Naive Bayes: check assumptions (e.g., independence of features), class imbalance.

k-NNs: Check sensitivity of the number of neighbors and feature normalization.

Julia Lenc

The process: Stage 3

1. Address Overfitting (or Underfitting):

Evaluate if models generalize well:

Logistic Regression: use regularization (L1/L2) to prevent overfitting.

Naive Bayes: likely robust to overfitting; verify appropriate smoothing.

k-NNs: adjust k (up for more smoothing) or distance weightings.

2. Reassess pre-processing, if results are not satisfactory:

If performance is unsatisfactory, revisit:

Feature engineering (e.g., removing irrelevant features, adding interaction terms).

Scaling/encoding for k-NNs and Logistic Regression.

Addressing feature independence violations for Naive Bayes.

3. Fine-tune:

If performance is unsatisfactory, optimize hyperparameters:

Logistic Regression: Grid search for regularization strength (C).

Naive Bayes: Optimize type of NB (e.g., Gaussian for continuous data).

k-NNs: Adjust k, distance metrics, and weighted neighbor calculations.

4. Escalate:

If nothing works: Logistic Regression → SVM or Neural Networks, k-NNs → Ensemble

Import the data

Pre-requisites: *data preprocessing is here, validation criteria are here*

```python
# Step 1: Import the pandas library
# pandas is a powerful library for working with structured data (e.g., CSV files).
import pandas as pd

# Step 2: Define the path to your CSV file
# Replace 'your_file.csv' with the actual name of your file stored locally.
# For example, your file could be named 'data.csv', and it should be in the same directory as this script.
csv_file_path = 'your_file.csv'

# Step 3: Read the CSV file into a pandas DataFrame
# A DataFrame is like a table with rows and columns—it stores your data.
# The `pd.read_csv()` function reads the CSV file and creates this table in Python.
data = pd.read_csv(csv_file_path)

# Step 4: Check if the data was loaded correctly
# Use the `.head()` method to print the first 5 rows of the table (this helps us confirm the data is loaded).
# Output: You should see a small portion of the data as a table in the console.
print("Preview of the dataset:")
print(data.head())

# Step 5: Get basic information about the data
# The `.info()` method provides important details, such as:
# - Number of rows and columns of the data
# - Names of the columns
# - Data types of each column (e.g., numerical, text)
# - Memory usage
print("\nBasic information about the dataset:")
print(data.info())

# Step 6: Display summary statistics for numerical columns
# The `.describe()` method gives a quick overview of numerical data, such as:
# - Mean, min, max, and standard deviation of each numeric column
# This helps us understand the data distribution.
print("\nSummary statistics for numerical columns:")
print(data.describe())
```

Model Run. Evaluation

```python
# Step 1: Import necessary libraries
# pandas: For data manipulation
# numpy: For numerical operations
# train_test_split: To split dataset into training and testing sets
# StandardScaler: To scale features (needed for LR and k-NNs)
# Models: LogisticRegression, GaussianNB, KNeighborsClassifier
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Step 2: Load the preprocessed CSV dataset
csv_file_path = 'your_file.csv'  # Replace with your file path
data = pd.read_csv(csv_file_path)

# Step 3: Prepare feature matrix (X) and target vector (y)
# Assuming the dataset has features in all columns except the last one,
# and the last column is the target variable ("label").
X = data.iloc[:, :-1].values  # Features (everything except the last column)
y = data.iloc[:, -1].values   # Target variable (last column)

# Step 4: Split the dataset into training and testing sets
# 80% for training, 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Standardize Features (Important for LR and k-NNs)
# Logistic Regression and k-NNs require scaled data to perform well.
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Rest of the code is on the next slide :)

Model Run. Evaluation

```python
# --- LOGISTIC REGRESSION ---
print("\nRunning Logistic Regression...")
# Step 6: Initialize and train the Logistic Regression model
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train)  # Train the model using the training set

# Step 7: Make predictions and evaluate accuracy
lr_predictions = lr_model.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_predictions)
print(f"Logistic Regression Accuracy: {lr_accuracy:.2f}")

# --- NAIVE BAYES ---
print("\nRunning Naive Bayes...")
# Step 8: Initialize and train the Naive Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)  # Train the model using the training set

# Step 9: Make predictions and evaluate accuracy
nb_predictions = nb_model.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)
print(f"Naive Bayes Accuracy: {nb_accuracy:.2f}")

# --- k-NEAREST NEIGHBORS ---
print("\nRunning k-Nearest Neighbors (k-NNs)...")
# Step 10: Initialize and train the k-NNs model
knn_model = KNeighborsClassifier(n_neighbors=5)  # Choosing k=5 as default
knn_model.fit(X_train, y_train)  # Train the model using the training set

# Step 11: Make predictions and evaluate accuracy
knn_predictions = knn_model.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
print(f"k-Nearest Neighbors Accuracy: {knn_accuracy:.2f}")
```

This is the rest of the code from previous page
It would not work on its own :)

Model Run. Evaluation

```python
# --- LOGISTIC REGRESSION ---
print("\nRunning Logistic Regression...")
# Step 6: Initialize and train the Logistic Regression model
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train)  # Train the model using the training set

# Step 7: Make predictions and evaluate accuracy
lr_predictions = lr_model.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_predictions)
print(f"Logistic Regression Accuracy: {lr_accuracy:.2f}")

# --- NAIVE BAYES ---
print("\nRunning Naive Bayes...")
# Step 8: Initialize and train the Naive Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)  # Train the model using the training set

# Step 9: Make predictions and evaluate accuracy
nb_predictions = nb_model.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)
print(f"Naive Bayes Accuracy: {nb_accuracy:.2f}")

# --- k-NEAREST NEIGHBORS ---
print("\nRunning k-Nearest Neighbors (k-NNs)...")
# Step 10: Initialize and train the k-NNs model
knn_model = KNeighborsClassifier(n_neighbors=5)  # Choosing k=5 as default
knn_model.fit(X_train, y_train)  # Train the model using the training set

# Step 11: Make predictions and evaluate accuracy
knn_predictions = knn_model.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
print(f"k-Nearest Neighbors Accuracy: {knn_accuracy:.2f}")
```

This is the rest of the code from previous page
It would not work on its own :)
Here we evaluate accuracy. Add other aligned metrics!

Overfitting / Underfitting

```python
# Logistic Regression: Address Overfitting with Regularization
# L2 regularization is default; reduce C to increase regularization.
lr_model = LogisticRegression(C=0.1, random_state=42)  # Stronger regularization

# Naive Bayes: Verify Smoothing to Avoid Underfitting
# Adjust 'var_smoothing' (default: 1e-9) for continuous data in GaussianNB.
nb_model = GaussianNB(var_smoothing=1e-9)

# k-NNs: Address Overfitting/Underfitting by Adjusting 'k' and Distance Weighting
# Increase 'k' to reduce variance, and use weighted distances for neighbors.
knn_model = KNeighborsClassifier(n_neighbors=10, weights='distance')  # Larger k, weighted neighbors
```

Reassess pre-processing

```python
# Step 4: Reassess Pre-processing for Each Model

# --- FEATURE ENGINEERING ---
# Example: Removing irrelevant features (if domain knowledge suggests they don't contribute)
# Let's say we know "ID" and "Timestamp" are irrelevant for our prediction task.
# Drop these columns if they exist in the dataset.
irrelevant_features = ['ID', 'Timestamp']  # Replace with your actual irrelevant columns
X = pd.DataFrame(X, columns=data.columns[:-1])  # Add column names back temporarily for clarity
X = X.drop(columns=irrelevant_features, errors='ignore').values  # Drop and convert to NumPy array

# Example: Adding interaction terms for Logistic Regression (if applicable)
# Interaction terms (e.g., x1 * x2) can help capture relationships between features.
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(interaction_only=True, include_bias=False, degree=2)
X_with_interactions = poly.fit_transform(X)

# --- SCALING & ENCODING ---
# Logistics Regression and k-NNs require properly scaled numeric features for optimal performance.
# Replace original X_train/X_test with scaled versions.
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Naive Bayes: Ensure features are appropriately encoded or normalized to address independence violations.
# Example: Encoding categorical variables for Naive Bayes (if applicable).
# Use one-hot encoding for categorical variables, which GaussianNB handles better.
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
categorical_columns = ['Category']  # Replace with actual categorical column names
X = pd.DataFrame(X, columns=data.columns[:-1])  # Add column names back temporarily
X_categorical = encoder.fit_transform(X[categorical_columns])  # Encode categorical features
X = pd.concat([X.drop(columns=categorical_columns), pd.DataFrame(X_categorical)], axis=1).values
```

Fine-tuning

(last chance before complexity increase)

```python
# Step 5: Fine-Tuning and Complexity Optimization with Grid Search

# Import GridSearchCV for hyperparameter tuning
from sklearn.model_selection import GridSearchCV

# --- LOGISTIC REGRESSION ---
print("Fine-tuning Logistic Regression...")
lr_param_grid = {'C': [0.01, 0.1, 1, 10, 100]}  # Grid for regularization strength
lr_grid_search = GridSearchCV(LogisticRegression(random_state=42), lr_param_grid, cv=5, scoring='accuracy')
lr_grid_search.fit(X_train, y_train)

# Best Logistic Regression model
best_lr_model = lr_grid_search.best_estimator_
print(f"Best Logistic Regression Parameters: {lr_grid_search.best_params_}")
print(f"Logistic Regression Accuracy: {best_lr_model.score(X_test, y_test):.2f}")

# --- NAIVE BAYES ---
print("\nFine-tuning Naive Bayes...")
# Example: Naive Bayes does not require extensive hyperparameter tuning in most cases.
# You may adjust smoothing manually.
nb_gaussian_model = GaussianNB(var_smoothing=1e-9)  # Example showing tuned smoothing
nb_gaussian_model.fit(X_train, y_train)

# Evaluate Naive Bayes model
nb_accuracy = nb_gaussian_model.score(X_test, y_test)
print(f"Naive Bayes Accuracy: {nb_accuracy:.2f}")

# --- k-NEAREST NEIGHBORS ---
print("\nFine-tuning k-Nearest Neighbors (k-NNs)...")
knn_param_grid = {
    'n_neighbors': [3, 5, 10, 15],           # Adjust neighbor count
    'weights': ['uniform', 'distance'],      # Uniform or weighted neighbors
    'metric': ['euclidean', 'manhattan']     # Distance metrics
}
knn_grid_search = GridSearchCV(KNeighborsClassifier(), knn_param_grid, cv=5, scoring='accuracy')
knn_grid_search.fit(X_train, y_train)

# Best k-NNs model
best_knn_model = knn_grid_search.best_estimator_
print(f"Best k-NNs Parameters: {knn_grid_search.best_params_}")
print(f"k-NNs Accuracy: {best_knn_model.score(X_test, y_test):.2f}")

# --- GENERAL ADVICE ---
# If underperformance persists, evaluate more advanced models:
# Logistic Regression can upgrade to SVMs or neural networks.
# k-NNs can evolve into advanced ensemble methods (e.g., Random Forests).
```

Did you find it useful?

Save

Share

Follow

Analytics, Market Research,
Machine Learning and AI

Julia Lenc