Classifiers

Instructor: Proserpio Davide

Why do we Need Classifiers?

- Which customer will churn? (Yes/No)
- Which consumers are more likely to use a coupon?
- Which ad will they click? (Ad A, B, or C)
- What star rating will they leave? (1–5)
- Which reviews are fake and which are organic?
- **Marketing payoff:**
  - Better targeting
  - Less wasted spend
  - Better ads
  - Improved personalization.
  - Fight platform manipulation

What s a Classifier?

- A model that assigns each customer (or observation) to a **class/bucket**.
- Binary classification: 2 classes (e.g., churn vs no churn).
- Multi-class classification: more than 2 (e.g., which product category).
- Ordinal classification: ordered categories (e.g., star ratings).

How Does Classification Work?

- Inputs = **features** (age, income, past purchases, ad impressions).
- Output = **predicted class** (churn = Yes/No).
- Example: Predict coupon usage
  - “If income > $50k and #purchases > 5 → high chance to respond to coupon.”

Common Classifiers

- **Logistic Regression**
  - Interpretable, usually used as the baseline model
  - Odds ratios, coefficients
- **Decision Trees**
  - Intuitive, rules-based (“if age > 30 and income > 50k…”)
- **Random Forests / Gradient Boosting**
  - Ensemble methods, higher accuracy
- **Support Vector Machines (SVM)**
  - Separating hyperplanes
- **Neural Networks**
  - More complex, less interpretable

Logistic Regression (The Workhorse)

- Outputs probability (0 to 1).
- Decision rule: if $p > 0.5$ → predict “Yes.”
- Very interpretable
- **Marketing example:** Probability of clicking an ad = 0.72 → predict click.

Decision Trees

- Split customers using simple rules.
- Easy to explain to managers.
- Very interpretable
- **Marketing example:**
  - “If age < 30 and visited site > 3 times → likely to purchase.”

Ensembles (Random Forest, Boosting)

- Combine many trees → better predictions.
- Tradeoff: higher accuracy, lower interpretability.

	Random Forest	**Boosting** (Gradient Boosting / XGBoost)
How it works	Many trees built **in parallel** on random samples; predictions are **averaged**	Trees built **one after another**; each new tree **fixes previous mistakes**; predictions are **added up**
What it improves	**Stability** (reduces variance)	**Accuracy** (reduces bias)
Tuning	Few parameters; works well "out of the box"	More parameters (learning rate, depth, #trees); needs careful tuning
Overfitting risk	Lower	Higher if not regularized/early-stopped
Interpretability	Medium–Low	Low
When to use	Fast, strong **baseline**; noisy data; want reliability	Aim for **top accuracy** and you can tune a bit

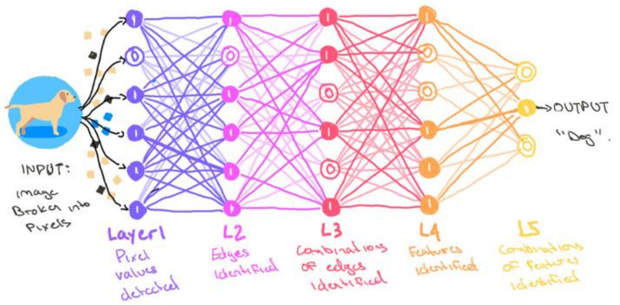
Support Vector Machines (SVM)

- A model that draws the **cleanest possible line** between two groups, leaving the **biggest gap** (margin) so it generalizes well.

- Think of two crowds on a field. SVM puts a rope between them and pulls it so there’s **maximum space** from both crowds. That space helps us make **safer decisions** on new people.

- Interpretability: Not as easy to say which feature matters most.

Neural Network (NN)



- A model that learns **layers of patterns**. Each layer finds **useful combinations** of your inputs to predict the outcome.
- **When it helps:**
    - Behavior likely depends on **interactions** (e.g., tenure × inactivity × price change).
    - You have **enough data** and want a model that can capture **complex patterns**.
- Black-box, i.e., hard to interpret

Model Training

- **The Goal of Prediction**
  - We care about whether the model can predict **new, unseen customers.**
- **The Risk of Overfitting**
  - If we only evaluate on the same data we trained on:
    - The model may **memorize noise or quirks** in that data.
    - Example: tree learns “Customer ID #123 always buys” → useless for future data.
  - Looks great in training (100% accuracy), but fails with new data.

Model Training

- **Train/Test Split** (generally 80%/20%)
  - **Training set**: Used to estimate the model (fit parameters, learn patterns).
  - **Test set**: Held out, never seen by the model → simulate new data.
- Compare performance:
  - If train accuracy >> test accuracy → **overfitting**.
  - If similar → model generalizes well.

Model Training

- Goal: Predict churn.
- Train on historical customers (2019–2023).
- Test on recent customers (2024).
- If the model performs well on the 2024 test set → confident we can use it in 2025.

Model Training: Cross-validation

- It is often helpful to perform cross-validation:
  - A way to estimate **out-of-sample performance** by repeatedly training on part of the data and validating on the rest.
  - Prevents **overfitting** to one lucky split.
  - Allows **tuning model parameters** (e.g., tree depth) and **compare models** fairly.
- Uses all data for both training and validation (rotating).

Model Training: Cross-validation

How it works (standard k-fold)

- Split data into **k** equal folds (e.g., k=5).
- For each fold: train on k–1 folds, validate on the held-out fold.
- **Average** the chosen metric across folds (e.g., Precision).
- Pick the model with the **best average**
- Refit on the full training set.

Evaluation Metrics

- **Accuracy**: overall % correct.
- **Precision**: % of predicted “yes” that were correct.
- **Recall**: % of actual “yes” caught.
- **AUC:** Are under the ROC curve

Computing metrics

- Suppose we predict whether a **customer will churn (Yes/No).**
- Here's the **confusion matrix** from our classifier on the **test set**:

	Predicted: Yes	**Predicted: No**
Actual: Yes	50 (True Positive)	10 (False Negative)
Actual: No	20 (False Positive)	120 (True Negative)

- **True Positive (TP):** predicted Yes, actually Yes → 50
- **False Positive (FP):** predicted Yes, actually No → 20
- **False Negative (FN):** predicted No, actually Yes → 10
- **True Negative (TN):** predicted No, actually No → 120

Computing metrics

	Predicted: Yes	**Predicted: No**
Actual: Yes	50 (True Positive)	10 (False Negative)
Actual: No	20 (False Positive)	120 (True Negative)

- Accuracy =
- Precision =
- Recall =

Accuracy: overall % correct.
Precision: % of predicted “yes” that were correct.
Recall: % of actual “yes” caught.

Computing metrics

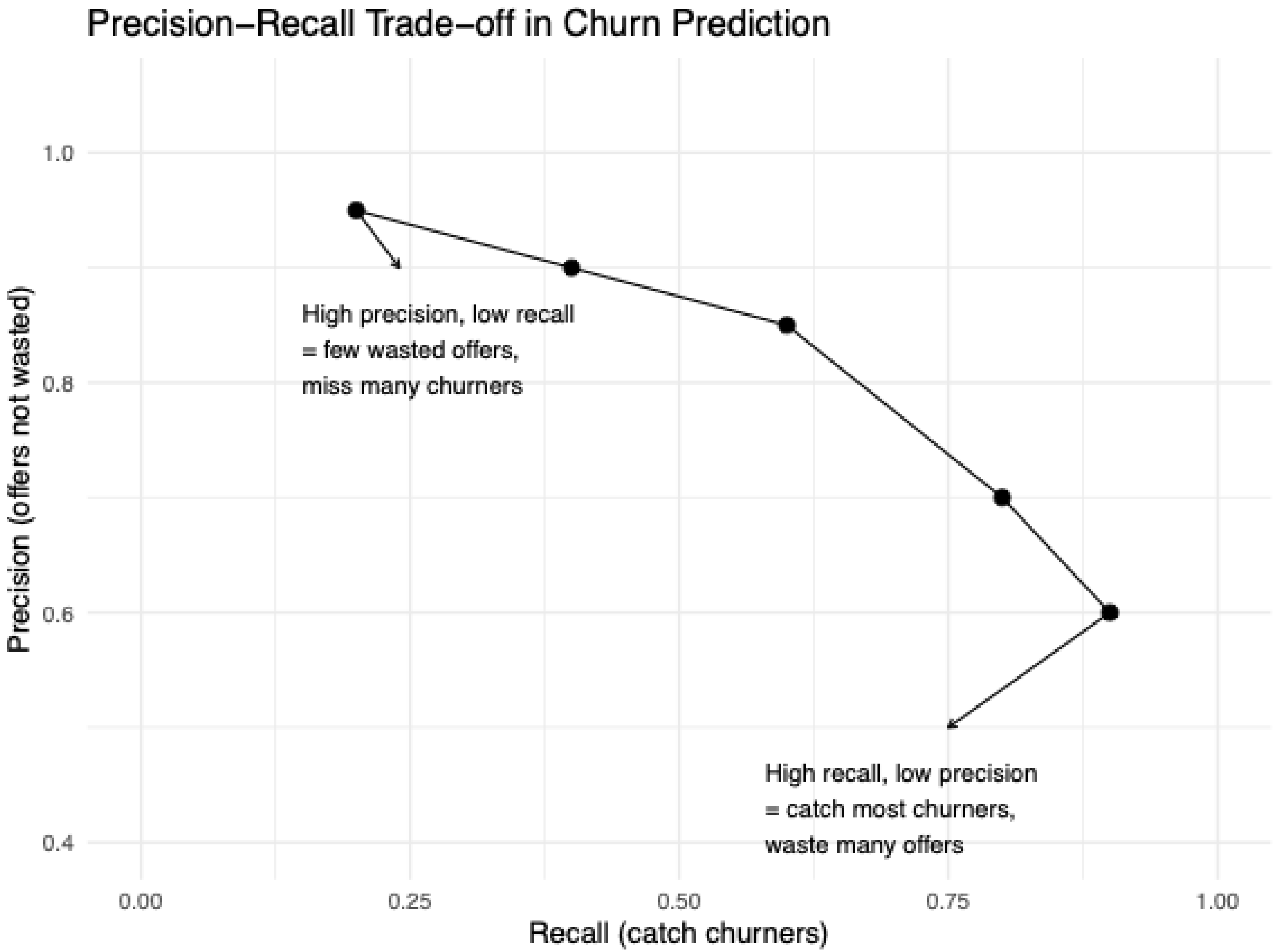
	Predicted: Yes	Predicted: No
Actual: Yes	50 (True Positive)	10 (False Negative)
Actual: No	20 (False Positive)	120 (True Negative)

- Accuracy = (TP + TN)/ (TP + TN + FP + FN) = .85
  - Overall, the model gets 85% of churn/stay predictions correct.

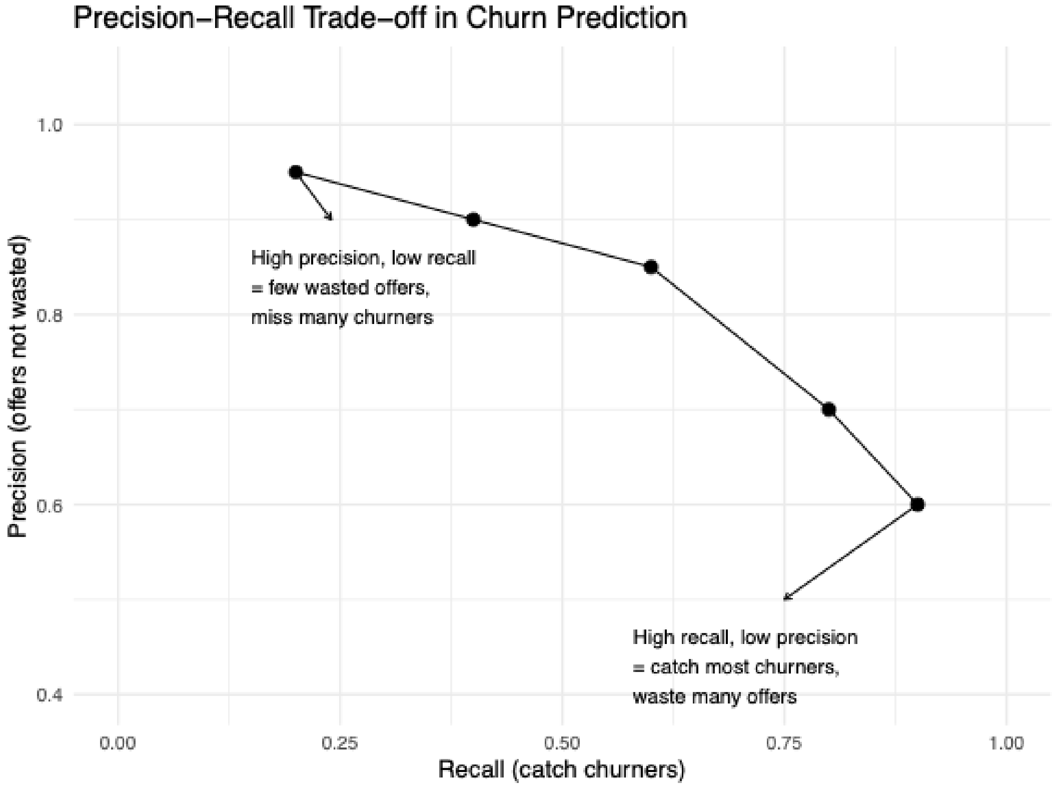
- Precision = TP/(TP + FP) = .71
  - When the model predicts a customer will churn, it’s correct 71% of the time.
  - Marketing implication: If we target “predicted churners” with retention offers, 29% of offers are wasted on customers who weren’t going to churn.

- Recall = TP/(TP + FN) = .83
  - Of all the customers who actually churned, the model successfully identified 83%.
  - Marketing implication: We save most of the at-risk customers, but 17% slipped through and churned without being flagged.

Computing metrics



Computing metrics



Intuition

- Your model gives each customer a **churn score** (estimated probability).
- You predict "churn" when **score ≥ threshold** t.
- **Raise** t→ you only flag the *very high* scores
  - **False positives (FP)** drop → **precision** $= \frac{TP}{TP+FP}$ tends to **increase**.
  - But some **true positives (TP)** also get filtered out → **recall** $= \frac{TP}{TP+FN}$ **decreases**.

Problems with accuracy

- Highly imbalanced classes: say positive class (e.g., churn) is 5%
  - 10,000 customers; **5% churners = 500** positives, 9,500 negatives.

Model	Pred Pos	TP	FP	FN	TN	Accuracy	Precision	Recall
Trivial “always No”	0	0	0	500	9,500	**95%**	—	**0%**

Problems with accuracy

- Highly imbalanced classes: say positive class (e.g., churn) is 5%
  - 10,000 customers; **5% churners = 500** positives, 9,500 negatives.

Model	**Pred Pos**	**TP**	**FP**	**FN**	**TN**	**Accuracy**	**Precision**	**Recall**
Trivial “always No”	0	0	0	500	9,500	**95%**	—	**0%**
Useful model	600	300	300	200	9,200	**95%**	50%	60%

F1

- **F1** is a single number that balances **precision** and **recall:**

$$F1 = \frac{Precison \ \times \ Recall}{Precision + Recall}$$

- When positives are **rare** (e.g., churners), raw **accuracy** can be misleading.
- **F1** rewards models that keep **both** precision (few wasted offers) **and** recall (catch churners) reasonably high.

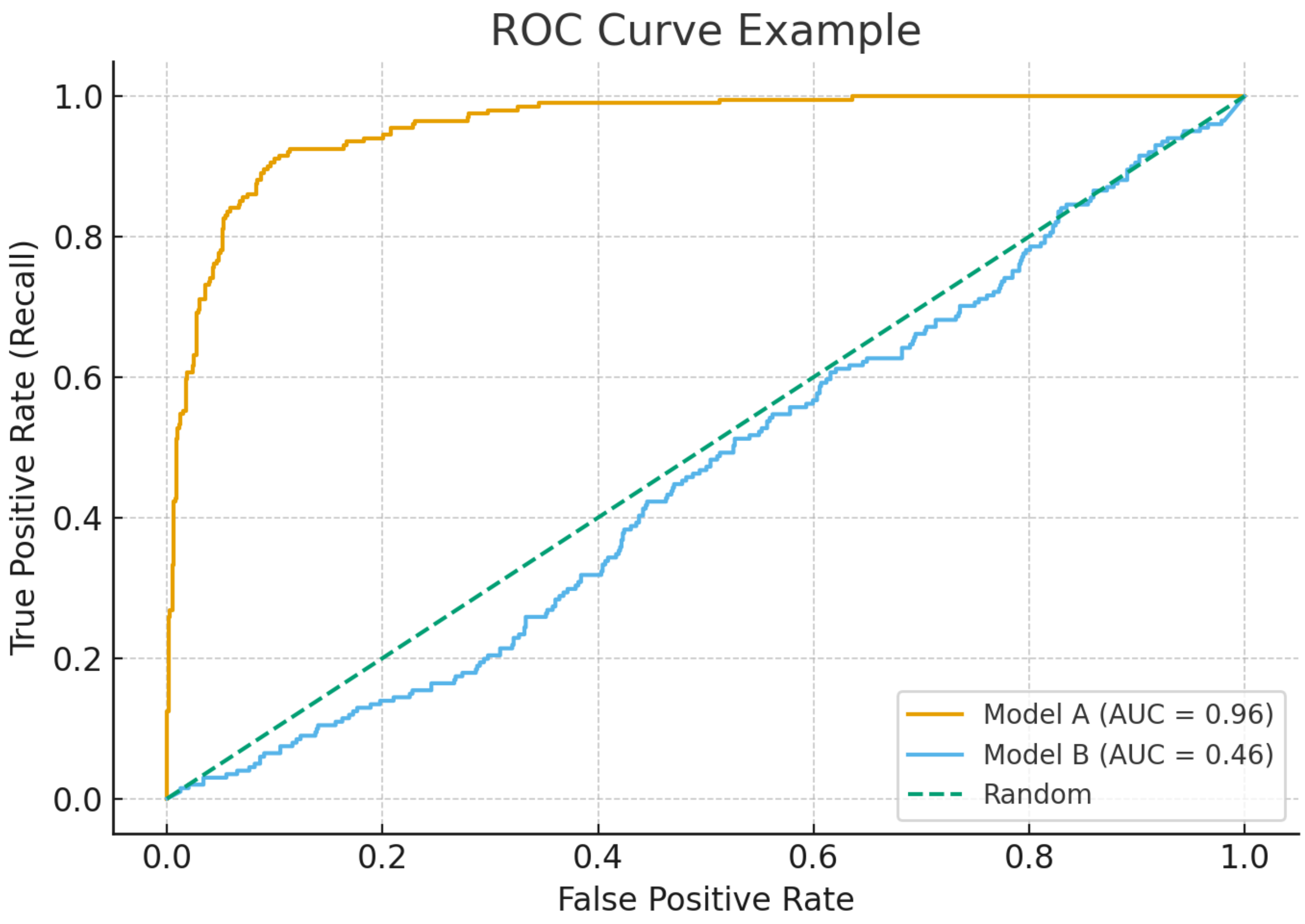
ROC (Receiver Operating Characteristic)

- A curve showing model performance as you **move the decision threshold** from strict → lenient.
- **Axes**
  - **TPR/Recall (y-axis):** % of actual positives correctly flagged.
  - **FPR (x-axis):** % of actual negatives incorrectly flagged.
- **How to read it**
  - Each point = a threshold.
  - **Top-left is best** (high TPR, low FPR).
  - The **diagonal** is random guessing.
- **Why it's useful**
  - You can see the **precision–recall trade-off** indirectly
  - Helps pick a **threshold** that fits your tolerance for false positives.

AUC (Area Under the ROC Curve)

- A single number (0–1) summarizing the **overall ranking power** of your model across all thresholds.
- **AUC =** chance the model gives a **higher score to a random positive** than to a random negative.
    - 0.5 = random; 1.0 = perfect separation.
- **Why it’s useful**
    - **Threshold-free** way to compare models early on.
    - Relatively insensitive to class imbalance

ROC-AUC example

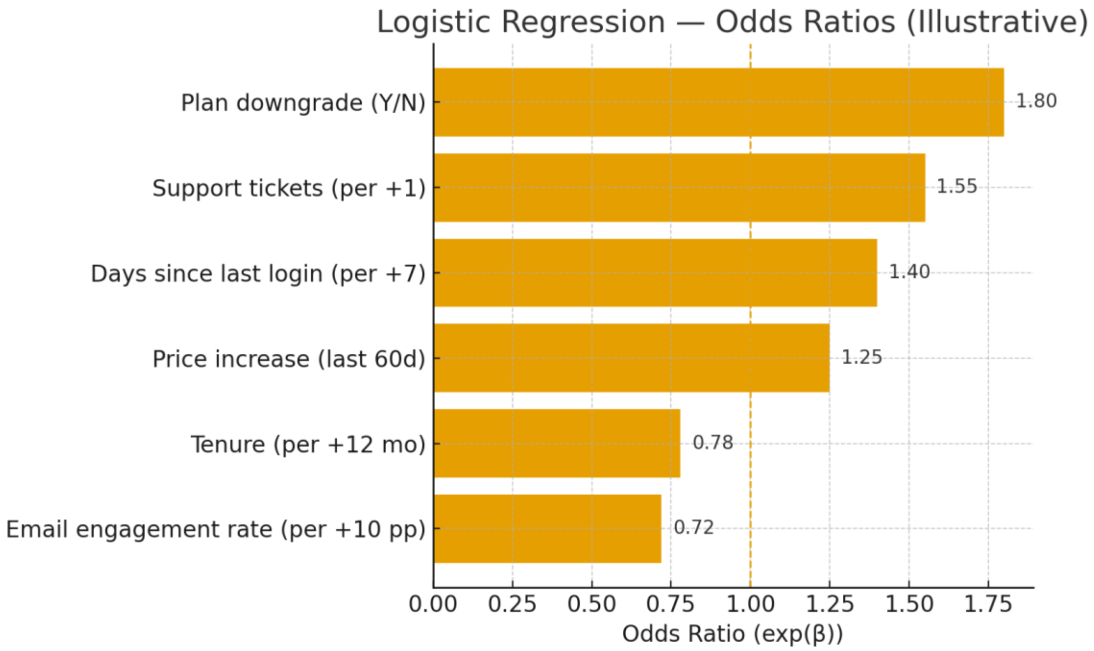


Interpreting Classifiers Output

- Coefficients & odds ratios (logit)
- Feature importance (tree-based models)
- SHAP / LIME for black-box models

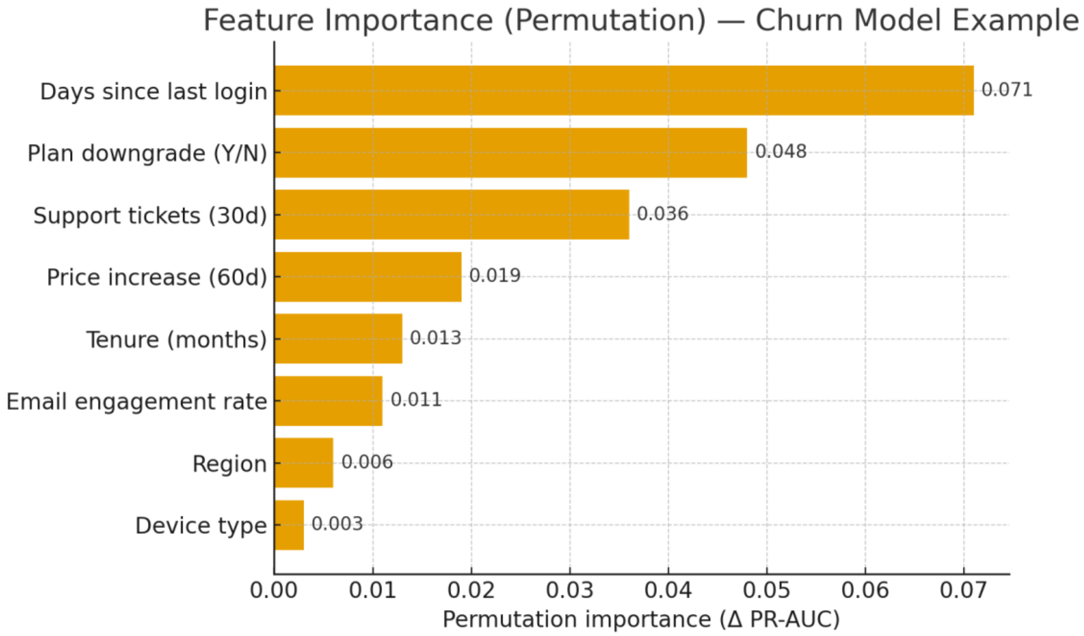
Coefficients & Odds Ratios (Logistic Regression)

- How each feature changes the **odds** of being in the positive class (e.g., churn).
- Turn coefficients into **odds ratios** by $\exp(\beta)$.
  - a 1-unit increase in the feature multiplies the **odds** by $\exp(\beta)$ (holding everything else constant).
- Show a **bar chart ordered by odds**



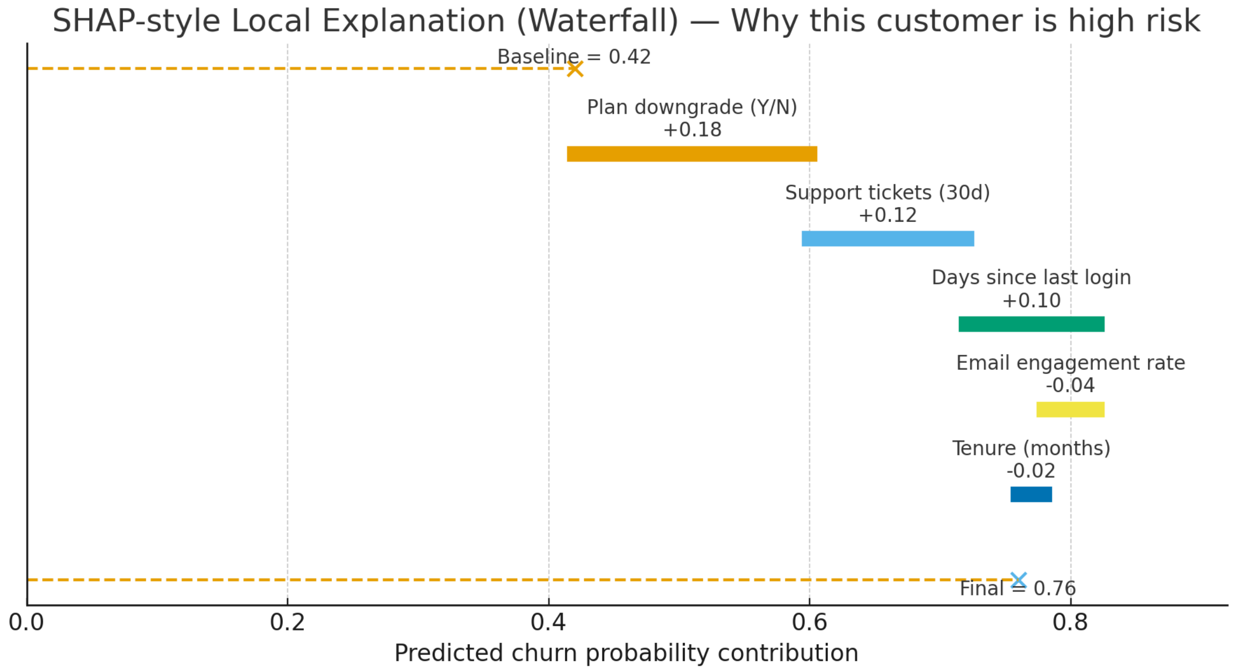
Feature Importance (Tree-Based Models)

- Which features did the model rely on overall to make accurate predictions?
- Identify top predictors to guide **policy levers** and **data collection** (e.g., “tickets” and “inactivity trend” matter most).
- Show a **bar chart** by importance



SHAP / LIME (Explaining Black-Box Models)

- Breaks a model’s score into **feature contributions** for each row of the data.
- **SHAP**: consistent, additive attributions.
  - **Waterfall plot** : baseline risk → add (+) and subtract (–) contributions to final score.


SHAP-style Local Explanation (Waterfall) — Why this customer is high risk
Baseline = 0.42
Plan downgrade (Y/N)
Support tickets (30d)
Days since last login
Email engagement rate
Tenure (months)
Final = 0.76
Predicted churn probability contribution