SEN608

Data Analytics and Statistics

Week 6: Classification

Assoc. Prof. Dr. Onur Polat

Source: Bruce, Bruce, Gedeck (2026). AI-Assisted Statistics for Data Scientists, 3rd Ed., O'Reilly - Chapter 5

OVERVIEW

This Week's Agenda

- The classification problem: binary and multi-class
- Naive Bayes classifier
- Discriminant Analysis (LDA)
- Logistic Regression (GLM, logit, odds ratios)
- Evaluating Classification Models
- Confusion Matrix, Precision, Recall, Specificity
- ROC Curve, AUC and Lift
- Strategies for Imbalanced Data (SMOTE, weighting, cost-based)

What is Classification?

- The target variable Y is categorical (class label)
- Binary classification: Y = 1 (class of interest) or Y = 0
- Examples: Credit default (yes/no), customer churn, phishing email
- Multi-class: Gmail filtering inbox into "primary / social / promotional / forums"
- Supervised learning: train on labeled data, apply to new data with unknown labels
- Most algorithms produce both predict (class) and predict_proba (probability score)
- Cutoff (threshold): probabilities above the cutoff are assigned to the class of interest (default 0.5)

KEY CONCEPTS

Classification - Key Terms

- Propensity score: Probability of belonging to the class of interest
- Cutoff: Threshold value that converts a probability to a class label
- Confusion Matrix: Table showing correct and incorrect classifications
- Precision: Of predicted positives, how many are truly positive
- Recall (sensitivity): Of true positives, how many were captured
- Specificity: Of true negatives, how many were captured
- AUC: Area under the ROC curve - threshold-independent model quality

SECTION I

Naive Bayes

Bayesian Classification

- Goal: estimate P(Y=i | X1,...,Xp) given observed predictor values
- Exact Bayesian solution: Find all records with the SAME X values; assign the most common class
- Problem: As the number of predictors grows, finding EXACT matches becomes impossible
- E.g. 8-variable voter model: even a large sample may have no exact match for a new record
- Naive assumption: predictors are conditionally independent GIVEN Y
- This simplification allows P(Xj | Y=i) to be estimated separately for each predictor
- Note: Naive Bayes requires CATEGORICAL (factor) predictors

The Naive Bayes Formula

$$P(Y=i \mid X1,X2,...,Xp) \text{ prop. } P(Y=i) \text{ prod_j } P(Xj \mid Y=i)$$

- For each class i: multiply prior P(Y=i) by each predictor's conditional probability P(Xj | Y=i)
- Normalize so probabilities across all classes sum to 1
- Assign the class with the highest posterior probability
- The posterior probabilities themselves can be biased
- But when the goal is to RANK records by P(Y=1), Naive Bayes works well
- Laplace smoothing: prevents zero probabilities for categories absent in training

Naive Bayes in Python (loan_data)

```
import pandas as pd
from sklearn.naive_bayes import MultinomialNB

loan_data = pd.read_csv('loan_data.csv.gz')

predictors = ['purpose_', 'home_', 'emp_len_']
outcome = 'outcome'

X = pd.get_dummies(loan_data[predictors],
                   prefix='', prefix_sep='',
                   drop_first=True, dtype=int)
y = loan_data[outcome]

naive_model = MultinomialNB(alpha=0.01, fit_prior=True)
naive_model.fit(X, y)

# Predict a new observation
new_loan = X.iloc[146:147]
print('Class:', naive_model.predict(new_loan)[0])
print('Probabilities:', naive_model.predict_proba(new_loan)[0])
```

Naive Bayes with Numeric Predictors

- Standard Naive Bayes requires CATEGORICAL predictors
- Two approaches for numeric variables:
- 1) Use a probability model: assume a normal distribution to estimate P(Xj | Y=i)
- 2) Bin the numeric predictor into categories and apply the standard algorithm
- In scikit-learn: GaussianNB for numeric variables; MultinomialNB for categorical
- For mixed data: build a pipeline with ColumnTransformer + OneHotEncoder + KBinsDiscretizer
- Example: in loan_data, borrower_score is numeric and purpose_ is categorical - KBinsDiscretizer is needed to use both

SECTION II

Discriminant Analysis (LDA)

LDA - Linear Discriminant Analysis

- The EARLIEST statistical classifier, introduced by Fisher (1936)
- Core idea: find linear combinations that MAXIMIZE between-class separation
- Assumption: predictors are normally distributed within each class (loose in practice)
- LDA assumes the SAME covariance matrix for both classes
- QDA (Quadratic): allows DIFFERENT covariance matrices per class (more flexible)
- Modern classification favors tree models and logistic regression over LDA
- Still valuable: fast training, interpretable, related to PCA

Fisher's Linear Discriminant

- Goal: find the linear combination that MAXIMIZES SSbetween / SSwithin
- SSbetween: between-group sum of squares (squared distance between group means)
- SSwithin: within-group sum of squares (spread around each group mean)
- Covariance matrix: measures how pairs of features vary together
- Mahalanobis distance: multivariate extension of z-score standardization

maximize: SS_between / SS_within

LDA in Python (loan3000 dataset)

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

loan3000 = pd.read_csv('loan3000.csv')
loan3000.outcome = loan3000.outcome.astype('category')

predictors = ['borrower_score', 'payment_inc_ratio']
outcome = 'outcome'

X = loan3000[predictors]
y = loan3000[outcome]

loan_lda = LinearDiscriminantAnalysis()
loan_lda.fit(X, y)

print('Discriminant weights:')
print(pd.DataFrame(loan_lda.scalings_, index=X.columns))

# Probability predictions
pred = pd.DataFrame(loan_lda.predict_proba(X),
                    columns=loan_lda.classes_)
print(pred.head())
```

SECTION III

Logistic Regression

The Logit and Logistic Response Function

- Core problem: a linear model on a binary outcome produces predictions outside (0,1)
- Solution: the LOGIT transform maps probabilities to (-inf, +inf) via log-odds
- Odds: P / (1-P) - the ratio of "successes" to "failures"
- Log-odds (logit): log(P / (1-P)) - now linearly modelable
- The sigmoid function is the INVERSE of logit: maps the real line back to (0, 1)

$$\log(P / (1-P)) = b0 + b1*X1 + \ldots + bq*Xq$$

$$p = 1 / (1 + \exp(-(b0 + b1*X1 + \ldots + bq*Xq)))$$

Logistic Regression in Python

```
from sklearn.linear_model import LogisticRegression

loan_data = pd.read_csv('loan_data.csv.gz')

predictors = ['payment_inc_ratio', 'purpose_', 'home_',
              'emp_len_', 'borrower_score']
outcome = 'outcome'

X = pd.get_dummies(loan_data[predictors],
                   prefix='', prefix_sep='',
                   drop_first=True, dtype=int)
y = loan_data[outcome]

# C=np.inf -> no regularization (pure MLE)
import numpy as np
logit_reg = LogisticRegression(C=np.inf, max_iter=1000)
logit_reg.fit(X, y)

print('Coefficients:', dict(zip(X.columns, logit_reg.coef_[0])))
print('Classes:', logit_reg.classes_)
```

Generalized Linear Models (GLM)

- A GLM is defined by two components:
- 1) A probability family: Binomial (logistic), Poisson (counts), Gamma, etc.
- 2) A link function: maps the response to the predictors (logit, log, identity, probit...)
- Logistic regression = Binomial family + logit link
- Linear regression = Gaussian family + identity link
- MLE (Maximum Likelihood Estimation): no closed form; iterative optimization (Fisher scoring)
- Deviance: model-quality measure - minus 2 times log-likelihood

Interpreting Coefficients - Odds Ratios

- In linear regression: bj = 1-unit increase in Xj -> bj-unit change in Y
- In logistic regression: bj = 1-unit increase in Xj -> bj-unit change in log(odds)
- Practical interpretation: exp(bj) is the ODDS RATIO
- Example (loan_data): purpose_small_business coefficient = 1.21526
- -> exp(1.21526) = 3.4 -> small-business loans are 3.4 TIMES riskier than credit-card payoff loans
- borrower_score coefficient = -4.6 -> best borrowers' default odds are ~100x LOWER
- Nonlinear mapping: coefficients are linear in log-odds, but S-shaped in probability

Detailed Analysis with statsmodels

```
import statsmodels.api as sm

y_numbers = [1 if yi == 'default' else 0 for yi in y]
logit_reg_sm = sm.GLM(y_numbers,
                      X.assign(const=1),
                      family=sm.families.Binomial())
logit_result = logit_reg_sm.fit()
print(logit_result.summary())

# Output includes:
# - Std.Error, z-value, p-value per coefficient
# - Null deviance, Residual deviance, AIC
# - Number of Fisher Scoring iterations

# Odds ratios and confidence intervals
import numpy as np
odds = pd.DataFrame({
    'coef': logit_result.params,
    'odds_ratio': np.exp(logit_result.params),
    'p_value': logit_result.pvalues
}).round(4)
print(odds)
```

SECTION IV

Model Evaluation

Accuracy and Its Limits

- Accuracy = (TP + TN) / Total observations
- The simplest and most commonly used metric
- IMPORTANT PITFALL: deeply misleading on imbalanced data!
- Example: in a %99 healthy / %1 sick screening task...
- A model that labels everyone "healthy" achieves 99% accuracy - yet detects ZERO patients
- Financial fraud: with 0.1% fraud rate, a 99.9%-accurate model can be useless
- Solution: use more nuanced metrics - Precision, Recall, F1, AUC

Confusion Matrix

- A 2x2 table (in the binary case) showing correct and incorrect classifications BY TRUTH

	Predicted: 1	Predicted: 0
Actual: 1	TP (True Positive)	FN (False Negative)
Actual: 0	FP (False Positive)	TN (True Negative)

Note: the class of interest is typically labeled Y=1 (default, fraud, disease)

Precision, Recall, Specificity

- Precision = TP / (TP + FP) -> "When I predict positive, how often am I right?"
- Recall (Sensitivity) = TP / (TP + FN) -> "Of true positives, what fraction did I catch?"
- Specificity = TN / (TN + FP) -> "Of true negatives, what fraction did I catch?"
- F1 Score = 2 * (P * R) / (P + R) -> harmonic mean of Precision and Recall
- Business impact: credit card fraud detection -> high Recall (minimize losses)
- Marketing campaign -> high Precision (do not waste budget)
- Medical screening -> balance both; false negative = missed patient, false positive = unnecessary test

The Rare Class Problem

- In many applications the class of interest is RARE: fraud, default, disease, purchase
- The rare class (1) is usually more IMPORTANT - misclassification cost is higher
- Example: 0.1% of web visitors purchase -> "no purchase for all" model gets 99.9% accuracy
- But this model is USELESS - it detects none of the real buyers
- Solution strategies (detailed later): resampling, weighting, cost-based classification, threshold tuning
- Evaluation: use Precision, Recall, F1, AUC instead of accuracy

ROC Curve and AUC

- ROC = Receiver Operating Characteristic
- A threshold-INDEPENDENT tool for evaluating model quality
- Y-axis: Recall (TPR, True Positive Rate)
- X-axis: 1 - Specificity (FPR, False Positive Rate)
- Sweeping the cutoff from 0 to 1 traces the Recall-vs-FPR curve
- AUC = Area Under the ROC Curve (0.5 = random, 1.0 = perfect)
- AUC 0.5-0.6 poor, 0.7-0.8 good, 0.9+ excellent
- Under heavy imbalance the Precision-Recall curve is more informative

EVALUATION

Computing ROC and AUC in Python

```
from sklearn.metrics import (roc_curve, roc_auc_score,
                             confusion_matrix,
                             classification_report)
import matplotlib.pyplot as plt

# Probability predictions
y_proba = logit_reg.predict_proba(X)[:, 0]  # 'default' class

# ROC curve
fpr, tpr, thresholds = roc_curve(
    [1 if yi=='default' else 0 for yi in y], y_proba)
auc = roc_auc_score(
    [1 if yi=='default' else 0 for yi in y], y_proba)

plt.figure(figsize=(6,6))
plt.plot(fpr, tpr, label=f'AUC = {auc:.3f}')
plt.plot([0,1], [0,1], 'k--', alpha=0.5)
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Recall)')
plt.title('ROC Curve - Logistic Regression (loan_data)')
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

Lift and Gains Chart

- Problem: for a rare class, cutoff <0.5 is needed - but which value?
- Lift helps answer this
- Sort records by predicted probability and split into deciles
- Divide the positive rate in the top decile by the overall positive rate -> Lift
- Example: overall positive rate = 0.1%, top-decile rate = 0.3% -> Lift = 3
- Very common in marketing: direct-mail campaigns, best-candidate ranking
- Uplift: A/B test + model combination; the INDIVIDUAL treatment-A-vs-B effect per subject

SECTION V

Strategies for Imbalanced Data

Undersampling (Downsampling)

- Problem: loan_data has 19% default, 81% paid off -> model overfits paid off
- Full-data training: model predicts only 0.4% default (50x below real rate)
- Solution 1: SHRINK the majority class (downsample)
- Method: draw N random samples from the majority (N = minority class size)
- Result: balanced training set; both classes have equal information weight
- Advantage: faster training due to smaller data
- Disadvantage: majority-class information is LOST - acceptable only on very large datasets

Weighting - Balance Without Losing Data

```
# Weighting: give higher weight to the rare class
import numpy as np
from sklearn.linear_model import LogisticRegression

# Default = rare class
default_wt = 1 / np.mean(y == 'default')  # ~5.3
wt = [default_wt if yi == 'default' else 1 for yi in y]

# sample_weight enables weighted training
full_model = LogisticRegression(C=np.inf, max_iter=10000)
full_model.fit(X, y, sample_weight=wt)

# Prediction rates are now balanced
pred_rate = 100 * np.mean(full_model.predict(X) == 'default')
print(f'Default prediction rate: {pred_rate:.2f}%')
# Expected output: ~58% (vs 0.4% without weighting)

# scikit-learn shortcut: class_weight='balanced'
balanced_model = LogisticRegression(
    class_weight='balanced', C=np.inf, max_iter=10000)
balanced_model.fit(X, y)
```

SMOTE - Synthetic Oversampling

```
# SMOTE = Synthetic Minority Oversampling Technique
# Chawla et al. (2002) - creates synthetic samples of the rare class
# Idea: generate new records based on NEIGHBORS of each rare example

from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline

# SMOTE pipeline (applied ONLY to training data)
smote_pipe = Pipeline([
    ('smote', SMOTE(sampling_strategy=0.8, random_state=42,
                    k_neighbors=5)),
    ('clf', LogisticRegression(max_iter=5000))
])

smote_pipe.fit(X_train, y_train)
y_pred = smote_pipe.predict(X_test)

# IMPORTANT: SMOTE is applied ONLY to the training set
# The test set must retain its ORIGINAL imbalanced distribution
# (it must mirror the real world)
```

Cost-Based Classification

- In practice, accuracy and AUC are "poor man's" ways of choosing a cutoff
- More business-focused approach: estimate the costs of False Positives and False Negatives
- Example (credit): expected return = P(Y=0) x R - P(Y=1) x C
- R = profit from a paid-off loan, C = cost of a default
- Approve the loan? Yes if the expected return is POSITIVE
- Advantage: model probabilities + business costs combine into the optimal decision
- Marketing: when per-offer costs (mail, time) and expected revenue are known, let the model choose the cutoff ITSELF

SUMMARY

What We Learned This Week

- Classification: predicting a categorical Y - binary or multi-class
- Naive Bayes: conditional-independence assumption, good for ranking, biased probabilities
- LDA: linear combination that maximizes group separation (Fisher 1936)
- Logistic Regression: logit transform + MLE, interpretable via odds ratios
- Model evaluation: Confusion matrix, Precision, Recall, F1, AUC-ROC, Lift
- Accuracy misleads on imbalanced data -> use more nuanced metrics
- Imbalanced data: undersampling, SMOTE, weighting, cost-based classification

References and Further Reading

- Bruce, Bruce, Gedeck (2026). AI-Assisted Statistics for Data Scientists, 3rd Ed., O'Reilly - Chapter 5
- Hosmer, Lemeshow, Sturdivant (2013). Applied Logistic Regression, 3rd ed., Wiley
- Hastie, Tibshirani, Friedman (2009). The Elements of Statistical Learning, 2nd ed., Springer
- James, Witten, Hastie, Tibshirani (2021/2023). An Introduction to Statistical Learning (R/Python)
- Chawla et al. (2002). SMOTE: Synthetic Minority Over-sampling Technique, J. of AI Research
- scikit-learn docs: scikit-learn.org/stable/modules/linear_model.html#logistic-regression
- imbalanced-learn docs: imbalanced-learn.org

Next Week: Statistical Machine Learning

- K-Nearest Neighbors (KNN) - distance-based classification
- Decision Trees - prediction via recursive partitioning
- Random Forest - bootstrap aggregation (bagging)
- XGBoost - gradient boosting
- Feature importance and SHAP values

Questions?