Introduction to Classification Models

Machine Learning Guide

July 3, 2025

1 Introduction

1.1 What is Classification?

Classification is the task of predicting categories (like spam/not spam) instead of continuous numerical values. It represents one of the fundamental machine learning tasks that has widespread applications across various domains.

1.2 Real-World Examples

- **Email spam detection:** Automatically filtering unwanted emails
- **Medical diagnosis:** Classifying patients as healthy or sick
- **Image recognition:** Distinguishing between cats and dogs in photos
- **Fraud detection:** Identifying fraudulent versus legitimate transactions

Today's Goal: Learn 5 simple but powerful classification models!

2 Preparing Our Data

2.1 Why Split Data?

Data splitting is crucial for honest model evaluation:

- **Training set (70-80%):** Used to teach the model patterns
- **Test set (20-30%):** Used to evaluate real-world performance
- Prevents overfitting by avoiding "memorization" of answers

2.2 Data Scaling

Some models require features to be on the same scale to perform optimally. This ensures we're comparing "apples to apples" across different feature dimensions.

```
# Set up our practice data
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Create synthetic dataset (1000 examples, 10 features)
X, y = make_classification(n_samples=1000, n_features=10,
                           n_classes=2, random_state=42)

# Split data (70% train, 30% test)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=42)

# Scale features for models that need it
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Listing 1: Setting up practice data

3 Classification Models

3.1 Model 1: Logistic Regression

How it works: Logistic regression draws a "best fit" decision boundary between categories using a sigmoid function to output probabilities.

Best for: Linear relationships, quick baseline models

Limitations: Struggles with complex, non-linear patterns

Example use: Email spam detection

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
accuracy = log_reg.score(X_test_scaled, y_test)
print(f"Accuracy: {accuracy:.2f}")
```

Listing 2: Logistic Regression Implementation

3.2 Model 2: k-Nearest Neighbors (k-NN)

How it works: Following the principle "birds of a feather flock together," k-NN classifies data points based on the majority class of their k nearest neighbors.

Best for: Simple, intuitive problems with clear groupings

Limitations: Slow with large datasets, sensitive to k selection

Example use: Handwriting recognition

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)  # Try k=5 neighbors
knn.fit(X_train_scaled, y_train)
accuracy = knn.score(X_test_scaled, y_test)
print(f"Accuracy: {accuracy:.2f}")
```

Listing 3: k-Nearest Neighbors Implementation

3.3 Model 3: Decision Trees

How it works: Decision trees create a series of yes/no questions in a flowchart structure, building a tree by finding the best feature splits at each node.

Best for: Problems requiring interpretability, non-linear relationships

Limitations: Prone to overfitting, unstable with small data changes

Example use: Customer segmentation

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=3)  # Limit tree depth
tree.fit(X_train, y_train)
accuracy = tree.score(X_test, y_test)
print(f"Accuracy: {accuracy:.2f}")
```

Listing 4: Decision Tree Implementation

3.4 Model 4: Random Forests

How it works: Random forests combine many decision trees through ensemble voting, where multiple trees contribute to the final prediction.

Best for: Complex problems, robust performance

Limitations: Less interpretable, slower than single trees

Example use: Fraud detection

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators=100)  # 100 trees
forest.fit(X_train, y_train)
accuracy = forest.score(X_test, y_test)
print(f"Accuracy: {accuracy:.2f}")
```

Listing 5: Random Forest Implementation

3.5 Model 5: Naive Bayes

How it works: Naive Bayes applies Bayes' theorem with the "naive" assumption that features are independent of each other.

Best for: Text classification, very fast training

Limitations: Independence assumption often unrealistic

Example use: Sentiment analysis (positive/negative reviews)

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train, y_train)
accuracy = nb.score(X_test, y_test)
print(f"Accuracy: {accuracy:.2f}")
```

Listing 6: Naive Bayes Implementation

4 Model Comparison

Model	Pros	Cons	Best For
Logistic Regression	Simple, fast	Linear relationships only	Baseline models
k-NN	Intuitive, no training needed	Slow prediction	Small datasets with clear groupings
Decision Tree	Easy to interpret	Overfits easily	Business rule extraction
Random Forest	Powerful, robust	Complex, slower	Complex problems
Naive Bayes	Very fast, good for text	Strong independence assumptions	Text classification

Table 1: Comparison of Classification Models

5 Conclusion

Each classification model has its strengths and ideal use cases. Start with simpler models like logistic regression for baseline performance, then experiment with more complex approaches like random forests for improved accuracy on challenging problems. The key is understanding your data characteristics and requirements to select the most appropriate model.