Classification

Created	@September 29, 2024 6:01 PM
Class	Supervised Learning with scikit-learn

Introduction to Supervised Learning with scikit-learn

Machine Learning Overview

Definition

- Process where computers learn to make decisions from data without explicit programming
- Examples:
  - Predicting email spam
  - Clustering books into categories

Types of Machine Learning

Unsupervised Learning

- Uncovers hidden patterns from unlabeled data
- Example: Grouping customers based on purchasing behavior

Supervised Learning

- Uses known values to build predictive models
- Predicts values for unseen data using features
- Example: Predicting basketball player's position based on points per game

Supervised Learning

Types

1. Classification
    - Predicts labels or categories
    - Example: Binary classification of bank transactions as fraudulent or non-fraudulent
2. Regression
    - Predicts continuous values
    - Example: Predicting property prices based on features like number of bedrooms and size

Terminology

- Features: Also known as predictor variables or independent variables
- Target variable: Also called dependent variable or response variable

Requirements

- Data must not have missing values
- Data must be in numeric format
- Data should be stored as pandas DataFrames, Series, or NumPy arrays

scikit-learn Workflow

General Syntax

```
# Import the model
from sklearn.module import Model

# Instantiate the model
model = Model()

# Fit the model
model.fit(X, y)

# Make predictions
predictions = model.predict(X_new)
```

Example: k-Nearest Neighbors

- Uses distance between observations to predict labels or values

Workflow Steps

1. Import the model from sklearn module
2. Instantiate the model
3. Fit the model to training data (X: features, y: target variable)
4. Use the model to make predictions on new data

```
# Example of classification prediction (spam detection)
predictions = model.predict(X_new)
# Returns an array of 0s (not spam) and 1s (spam)
```

Key Takeaways

- Supervised learning uses labeled data to build predictive models
- Two main types: classification and regression
- scikit-learn provides a consistent workflow for different models

- Exploratory data analysis is crucial before applying supervised learning techniques
- Proper data format and handling of missing values are essential prerequisites

Building a Classification Model with k-Nearest Neighbors (KNN)

Prerequisites

- We'll be using `telecom_churn` dataset.

Key Details

- Focus: Building a classifier for predicting labels of unseen data
- Algorithm: k-Nearest Neighbors (KNN)

Classification Model Building Process

1. Build a classifier using labeled data (training data)
2. Pass unlabeled data as input
3. Predict labels for unseen data
4. Evaluate the model's performance

k-Nearest Neighbors (KNN) Algorithm

Concept

- Predicts labels based on the k closest labeled data points
- Uses majority voting for classification

Mathematical Formulation

1. Distance Calculation:
   For two points
   p and q in n-dimensional space, the Euclidean distance is:
   $d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$
   Where p_i and q_i are the values of the i^{th} feature for points p and q respectively.

2. K-Nearest Neighbors:
   For a new point
   x, find the k training samples x_i closest to x according to the distance metric.

3. Majority Voting:
   For classification, the predicted class
   $\hat{y}$ for point x is:
   $\hat{y} = \text{mode}(y_i)\ for\ i = 1, 2, ..., k$
   Where y_i is the class label of the i^{th} nearest neighbor.

Algorithmic Description

1. Store all training samples (X, y) where X are feature vectors and y are labels.

2. For a new sample x:
   a. Calculate distances
   $d(x, x_i)$ to all training samples x_i.
   b. Select the k samples with the smallest distances to
   x.
   c. Assign the label based on majority vote of the k-nearest neighbors.

Example

- If k = 3, classify based on the majority label of 3 nearest neighbors
- If k = 5, classify based on the majority label of 5 nearest neighbors

```
# Visualization example (pseudo-code)
plt.scatter(x, y, c=labels)
```

```
plt.scatter(new_point_x, new_point_y, c='black')
plt.show()
```

KNN Implementation with scikit-learn

Data Preparation

```
# Split data into features and target
X = df[['total_day_charge', 'total_evening_charge']].values
y = df['churn'].values

# Check shapes
print(X.shape)  # (3333, 2)
print(y.shape)  # (3333,)
```

Model Creation and Fitting

```
from sklearn.neighbors import KNeighborsClassifier

# Instantiate the model
knn = KNeighborsClassifier(n_neighbors=15)

# Fit the model
knn.fit(X, y)
```

Making Predictions

```
# New data for prediction
X_new = np.array([[30, 10], [40, 20], [50, 30]])

# Make predictions
predictions = knn.predict(X_new)
print(predictions)  # [1, 0, 0]
```

Visualizing KNN Decision Boundary

- Example: Telecom company customer churn prediction
- Features: Total evening charge vs. Total day charge
- Labels: Churned (blue) vs. Not churned (red)
- Decision boundary created with k=15 neighbors

```
# Pseudo-code for visualization
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.contourf(xx, yy, Z, alpha=0.3)
plt.xlabel('Total Day Charge')
plt.ylabel('Total Evening Charge')
plt.show()
```

Key Takeaways

- KNN is a simple yet effective classification algorithm
- It predicts based on the majority vote of k nearest neighbors
- The choice of k and the distance metric can significantly affect performance
- scikit-learn provides an easy-to-use implementation of KNN
- Proper data formatting is crucial (2D array for features, 1D array for target)
- Visualizing the decision boundary helps in understanding the model's behavior

Evaluating Classification Model Performance

Prerequisites

- We'll be using `telecom_churn` dataset.

Key Details

- Focus: Measuring and interpreting model accuracy
- Technique: Train-test split for unbiased evaluation

Model Evaluation Process

Accuracy Metric

- Definition: $\frac{\text{Number of correct predictions}}{\text{Total number of observations}}$
- Formula: $Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Observations}}$

Train-Test Split

1. Split data into training and test sets
2. Fit classifier on training set
3. Calculate accuracy on test set

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, tes
t_size=0.3, random_state=42, stratify=y)

knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X_train, y_train)

accuracy = knn.score(X_test, y_test)
print(f"Model accuracy: {accuracy:.2f}")
```

Interpreting Model Complexity

Underfitting vs. Overfitting

- Underfitting: Simple models unable to detect relationships in the dataset
- Overfitting: Complex models sensitive to noise in training data

Effect of k in KNN

- Larger k: Simpler model, smoother decision boundary
- Smaller k: More complex model, more irregular decision boundary

Model Complexity Curve

```
neighbors = range(1, 51)
train_accuracy = {}
test_accuracy = {}

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_accuracy[k] = knn.score(X_train, y_train)
    test_accuracy[k] = knn.score(X_test, y_test)

plt.plot(neighbors, list(train_accuracy.values()), label="Training Accuracy")
plt.plot(neighbors, list(test_accuracy.values()), label="Test Accuracy")
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

Mathematical Interpretation of KNN Complexity

As k increases, the decision function $f(x)$ for a point x becomes smoother:

$f(x) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(x)} y_i$

Where $\mathcal{N}_k(x)$ is the set of k nearest neighbors of x, and y_i are their labels.

Key Takeaways

- Use train-test split for unbiased performance evaluation
- Accuracy is a common metric for classification tasks

- Stratify the split to maintain label proportions
- Model complexity affects performance:
    - Too simple (high k) leads to underfitting
    - Too complex (low k) can lead to overfitting
- Use model complexity curves to find optimal k
- Best performance often occurs at moderate k values