TechNeeds
ML Upskill
Week 4

Naive Bayes

Naive Bayes is a probabilistic machine learning algorithm used for classification tasks. It is based on Bayes' Theorem, which describes the probability of an event based on prior knowledge of conditions related to the event. Despite its simplicity, Naive Bayes can be very effective, particularly for text classification.

Bayes' Theorem:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

- P(A∣B): Posterior probability of class AAA given predictor BBB
- P(B∣A): Likelihood of predictor BBB given class AAA
- P(A): Prior probability of class AAA
- P(B): Prior probability of predictor BBB

Naive Assumption: The algorithm assumes that the predictors (features) are independent given the class. This is rarely true in real-world applications, hence the term "naive". Despite this, the classifier performs well in many scenarios.

Steps in Naive Bayes Classification

1. **Prepare the Data**:
    - Clean and preprocess the data (e.g., handle missing values, encode categorical variables).
    - Split the data into training and testing sets.
2. **Calculate Priors**:

    - Compute the prior probability for each class based on the training data.
3. **Calculate Likelihoods**:
    - For Gaussian Naive Bayes, calculate the mean and variance of each feature for each class.
    - For Multinomial and Bernoulli Naive Bayes, calculate the likelihood of each feature given each class.
4. **Apply Bayes' Theorem**:
    - Use the priors and likelihoods to compute the posterior probability for each class given a new instance.
    - Assign the class with the highest posterior probability to the instance.

Mathematical Formulation

For a given class Ck and feature vector x=(x1,x2,…,xn), the posterior probability is:

$$P(C_k|\mathbf{x}) = \frac{P(C_k) \cdot P(x_1|C_k) \cdot P(x_2|C_k) \cdot \ldots \cdot P(x_n|C_k)}{P(\mathbf{x})}$$

$$P(C_k|\mathbf{x}) \propto P(C_k) \cdot \prod_{i=1}^{n} P(x_i|C_k)$$

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline

# Sample data
texts = ["I love programming", "Python is great", "I hate bugs", "Debugging is fun"]
labels = ["positive", "positive", "negative", "positive"]

# Create a pipeline with a vectorizer and a Naive Bayes classifier
model = make_pipeline(CountVectorizer(), MultinomialNB())

# Train the model
model.fit(texts, labels)

# Predict the category of a new text
new_text = ["I love debugging"]
predicted_label = model.predict(new_text)
print(predicted_label)
```

```
['positive']
```

Support Vector Machines (SVM)

Support Vector Machines (SVM) are supervised machine learning models used for classification and regression tasks. SVM aims to find the optimal hyperplane that best separates different classes in the feature space.

Hyperplane:

- In an nnn-dimensional space, a hyperplane is a flat affine subspace of dimension n−1n-1n−1 that separates data points into different classes.
- For a two-dimensional space, the hyperplane is a line; for a three-dimensional space, it is a plane.

Margin:

- The margin is the distance between the hyperplane and the closest data points from either class, known as support vectors.
- SVM aims to maximize this margin to improve the model's generalization ability.

Support Vectors:

- Support vectors are the data points that lie closest to the hyperplane and influence its position and orientation.
- These points are critical in defining the optimal hyperplane.

Kernel Trick:

- The kernel trick allows SVM to efficiently perform a non-linear classification by transforming the original data into a higher-dimensional space using a kernel function.
- Common kernels include linear, polynomial, radial basis function (RBF), and sigmoid.

Mathematical Formulation

1. **Objective**:
    - For linearly separable data, SVM aims to find a hyperplane that separates the classes with the maximum margin.
    - The decision function for classification is: f(x)=w⋅x+bf(x) = w \cdot x + b

$$f(x) = w \cdot x + b$$

The goal is to minimize $\frac{1}{2}\|w\|^2$ subject to the constraint that all data points are correctly classified:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

Soft Margin SVM:

- For non-linearly separable data, SVM introduces slack variables ξi to allow some misclassifications.
- The objective becomes:

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

Subject to:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \forall i$$

C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error.

Steps in SVM Classification

1. **Prepare the Data**:
    - Clean and preprocess the data.
    - Encode categorical variables and scale the features.
2. **Choose a Kernel**:
    - Select an appropriate kernel function based on the data and problem.
3. **Train the Model**:
    - Fit the SVM model to the training data.
4. **Tune Hyperparameters**:
    - Use cross-validation to tune hyperparameters such as C, γ, and kernel-specific parameters.
5. **Evaluate the Model**:
    - Assess the model's performance on the testing set using metrics like accuracy, precision, recall, and F1-score.


```python
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Load dataset
iris = datasets.load_iris()
X, y = iris.data, iris.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train SVM model with RBF kernel
model = SVC(kernel='rbf', gamma='scale')
model.fit(X_train, y_train)

# Predict labels for the test set
y_pred = model.predict(X_test)
```

```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```