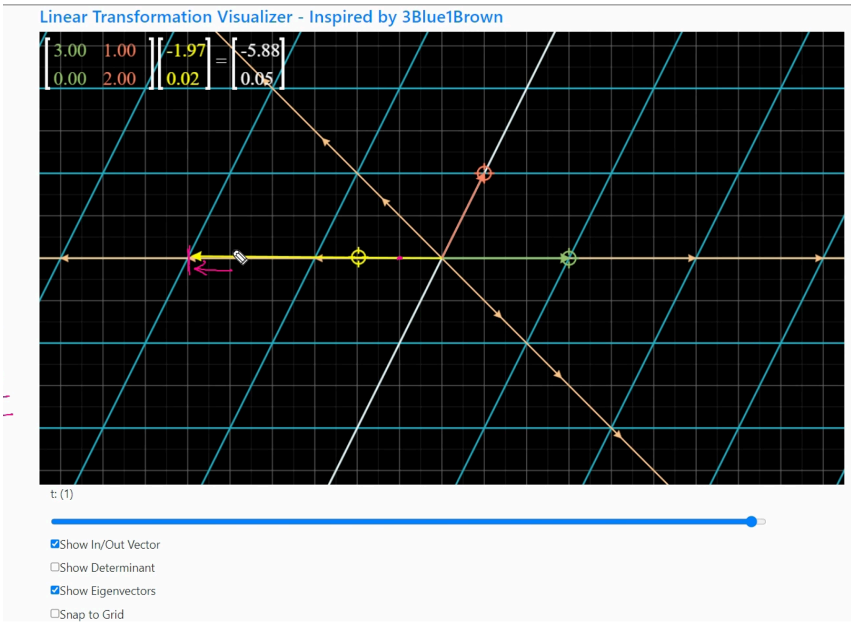
Maths in ML
Sunday, 17 November 2024 12:09 AM

Machine Learning Mathematics: Practical Examples & Intuitive Explanations

1. Eigenvectors & PCA: A Visual Guide

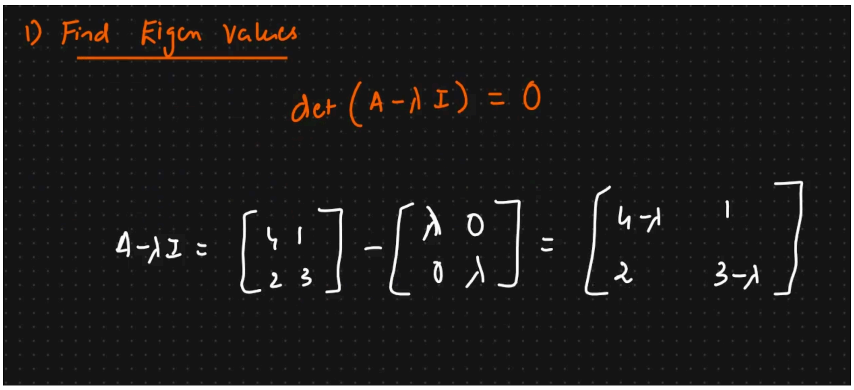
Understanding Eigenvectors Intuitively
Imagine you're stretching a rubber sheet:
- Regular vectors change both direction and magnitude when stretched
- Eigenvectors only change in magnitude (scaled by eigenvalue)
- They represent the "natural" directions of transformation

In the below diagram if we place the new vector(yellow) in any one of 2 lines(orange mixed brown on x=0 and diagonal) then it will only scale, there will not be direction changes in any other points if we keep it will change the direction as well.



Eigen value : A scalar that indicates how much an eigen vector is streched or compressed during linear transformation
Eigen Vector : A non zero vector that only changes in scale not direction when a linear transformation is applied
Av = lambda v

How to find Eigen value of a matrix



= (4-L)*(3-L)-2 = L^2 -7L +10
L = -b +- sqrt(b^2 - 4ac) / 2a

L = 5 L =2

```python
import numpy as np
import matplotlib.pyplot as plt

# Example: Finding eigenvectors of a 2x2 matrix
matrix = np.array([[4, 1],
                  [1, 4]])
eigenvalues, eigenvectors = np.linalg.eig(matrix)

print("Eigenvalues:", eigenvalues)
# Output: [5. 3.]
print("Eigenvectors:", eigenvectors)
# Output: [[ 0.70710678  0.70710678]
#         [-0.70710678  0.70710678]]
```

PCA (Principal Component Analysis) by Example

Let's say we have student data with multiple features:
```python
from sklearn.decomposition import PCA
import pandas as pd

# Example dataset
student_data = pd.DataFrame({
    'height': [170, 175, 160, 180, 165, 172],
    'weight': [70, 75, 60, 85, 65, 71],
    'study_hours': [3, 4, 2, 3, 2, 4],
    'sleep_hours': [7, 8, 6, 7, 6, 8],
    'test_score': [85, 90, 75, 88, 78, 89]
})

# Apply PCA
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(student_data)

# Check variance explained
print("Variance explained:", pca.explained_variance_ratio_)
# Output: [0.72, 0.21] - First two components explain 93% of variance
```

Visual Explanation of PCA:
1. Original data has 5 dimensions (features)
2. PCA finds the directions (eigenvectors) with maximum variance
3. First principal component: Direction of most variation
4. Second principal component: Next best direction orthogonal to first
5. Result: 2D representation preserving most important patterns

2. Gradient Descent: Visual Journey

Simple Example: Finding Minimum of $y = x^2$
```python
def gradient_descent_example():
    x = 10  # Starting point
    learning_rate = 0.1

    for i in range(10):
        # Gradient of x² is 2x
        gradient = 2 * x
        # Update step
```

        x = x - learning_rate * gradient
        print(f"Step {i}: x = {x:.2f}, y = {x**2:.2f}")

Output shows x converging to 0 (minimum point)
```

### Real ML Example: Linear Regression
```python
# Simple linear regression using gradient descent
def linear_regression_gd():
    # Sample data
    X = np.array([1, 2, 3, 4, 5])
    y = np.array([2, 4, 6, 8, 10])

    # Initialize parameters
    w = 0  # weight
    b = 0  # bias
    learning_rate = 0.01

    for epoch in range(100):
        # Forward pass
        y_pred = w * X + b

        # Compute gradients
        dw = -2 * np.mean(X * (y - y_pred))
        db = -2 * np.mean(y - y_pred)

        # Update parameters
        w = w - learning_rate * dw
        b = b - learning_rate * db

        if epoch % 10 == 0:
            print(f"Epoch {epoch}: w = {w:.2f}, b = {b:.2f}")

# Final output shows w ≈ 2, b ≈ 0
```

3. Probability Distributions in Action

Normal Distribution Example
```python
import numpy as np
from scipy.stats import norm

# Generate sample heights (in cm) for a class
heights = np.random.normal(170, 10, 1000)  # mean=170, std=10

# Plot histogram with normal curve
plt.hist(heights, bins=30, density=True, alpha=0.7)
plt.plot(np.linspace(140, 200, 100),
         norm.pdf(np.linspace(140, 200, 100), 170, 10))
plt.title('Student Heights Distribution')
```

Practical ML Example: Weight Initialization
```python
# Neural network weight initialization
def initialize_weights(layer_size):
    # Xavier/Glorot initialization
    limit = np.sqrt(2.0 / layer_size)
    return np.random.normal(0, limit, size=layer_size)

# Example for a layer with 100 neurons

```python
# Example for a layer with 100 neurons
weights = initialize_weights(100)
```

4. Matrix Operations in Neural Networks

Simple Neural Network Layer
```python
def neural_network_layer(input_data, weights, bias):
    """
    input_data: shape (batch_size, input_features)
    weights: shape (input_features, output_features)
    bias: shape (output_features,)
    """
    # Matrix multiplication + bias
    output = np.dot(input_data, weights) + bias

    # Apply ReLU activation
    output = np.maximum(0, output)
    return output

# Example usage
batch_size = 32
input_features = 10
output_features = 5

input_data = np.random.randn(batch_size, input_features)
weights = np.random.randn(input_features, output_features)
bias = np.random.randn(output_features)

layer_output = neural_network_layer(input_data, weights, bias)
```

5. Statistical Measures in Model Evaluation

Practical Example: Model Performance Analysis
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score

def evaluate_model(y_true, y_pred):
    # Basic metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)

    # F1 Score
    f1 = 2 * (precision * recall) / (precision + recall)

    print(f"""
    Model Performance:
    Accuracy: {accuracy:.2f}
    Precision: {precision:.2f}
    Recall: {recall:.2f}
    F1 Score: {f1:.2f}
    """)

# Example with binary classification results
y_true = [1, 0, 1, 1, 0, 1]
y_pred = [1, 0, 1, 0, 0, 1]
evaluate_model(y_true, y_pred)
```

6. Activation Functions Visualized

6. Activation Functions Visualized

Common Activation Functions
```python
def plot_activation_functions():
    x = np.linspace(-5, 5, 100)

    # ReLU
    relu = np.maximum(0, x)

    # Sigmoid
    sigmoid = 1 / (1 + np.exp(-x))

    # Tanh
    tanh = np.tanh(x)

    plt.figure(figsize=(12, 4))
    plt.subplot(131)
    plt.plot(x, relu)
    plt.title('ReLU')

    plt.subplot(132)
    plt.plot(x, sigmoid)
    plt.title('Sigmoid')

    plt.subplot(133)
    plt.plot(x, tanh)
    plt.title('Tanh')
```

Real-world Application Examples

1. Image Dimensionality Reduction
```python
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load sample 28x28 image (784 dimensions)
image = load_sample_image()  # 784 dimensions
pca = PCA(n_components=50)  # Reduce to 50 dimensions

# Reduce and reconstruct
reduced = pca.fit_transform(image.reshape(1, -1))
reconstructed = pca.inverse_transform(reduced)

# Compare original vs reconstructed
plt.subplot(121)
plt.imshow(image)
plt.title('Original')
plt.subplot(122)
plt.imshow(reconstructed.reshape(28, 28))
plt.title('Reconstructed (50 components)')
```

2. Text Classification Example
```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

# Sample text data
texts = ["great product", "bad service", "excellent", "terrible"]
labels = [1, 0, 1, 0]  # 1: positive, 0: negative
```

```
# Convert text to numerical features
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(texts)

# Train classifier
classifier = MultinomialNB()
classifier.fit(X, labels)

# Predict new text
new_text = ["really great service"]
prediction = classifier.predict(vectorizer.transform(new_text))
print("Prediction:", "Positive" if prediction[0] == 1 else "Negative")
```

Tips for Understanding Complex Concepts

1. **Visualization First**:
   - Always try to visualize the concept
   - Start with 2D examples before moving to higher dimensions
   - Use analogies from everyday life

2. **Step-by-Step Learning**:
   - Break down complex operations into simple steps
   - Implement basic versions before adding complexity
   - Test understanding with small examples

3. **Practice with Real Data**:
   - Start with small, clean datasets
   - Gradually increase complexity
   - Compare results with different approaches

Remember: The goal is to build intuition before diving into mathematical complexity. Each concept builds on previous ones, so ensure solid understanding before moving forward.