optimizers

May 22, 2024

1 1. Regularization Visualization

```
[ ]: import streamlit as st
     import numpy as np
     from sklearn.linear_model import LinearRegression, Lasso, Ridge
     import matplotlib.pyplot as plt
     import plotly.graph_objs as go
```

1.0.1 Linear regression

```
[ ]: def Linearregression(X, y):
         model = LinearRegression()
         model.fit(X, y)
         y_pred = model.predict(X)

         fig, ax = plt.subplots()
         ax.scatter(X, y, color='black', label='Data Points')
         ax.plot(X, y_pred, color='blue', linewidth=3, label='Linear Regression')
         ax.set_xlabel('X')
         ax.set_ylabel('y')
         ax.set_title('Linear Regression')
         ax.legend()
         st.pyplot(fig)

         fig_3d = go.Figure()
         fig_3d.add_trace(go.Scatter3d(x=X.squeeze(), y=y, z=y_pred, mode='markers',␣
     ↪name='Actual Data Points'))
         fig_3d.add_trace(go.Scatter3d(x=X.squeeze(), y=y_pred, z=y_pred,␣
     ↪mode='lines', name='Predicted Line'))
         fig_3d.update_layout(scene=dict(xaxis_title='X', yaxis_title='y',␣
     ↪zaxis_title='Predicted y'))
         st.write("## 3D Visualization - Linear Regression")
         st.plotly_chart(fig_3d)
```

1.0.2 lasso regression

[ ]:
```
def lasso(X, y, alpha):
    lasso_model = Lasso(alpha=alpha)
    lasso_model.fit(X, y)
    y_pred_lasso = lasso_model.predict(X)
    coef = lasso_model.coef_[0]
    intercept = lasso_model.intercept_
    formula = f'y = {coef:.2f}X + {intercept:.2f}'
    explanation = f"In Lasso regression, the penalty term (alpha) is added to␣
↪the absolute values of the coefficients (L1 regularization), which can␣
↪result in sparse models with some coefficients being exactly zero."

    fig, ax = plt.subplots()
    ax.scatter(X, y, color='black', label='Data Points')
    ax.plot(X, y_pred_lasso, color='red', linewidth=2, label='Lasso Regression')
    ax.set_xlabel('X')
    ax.set_ylabel('y')
    st.title(f'Lasso Regression')
    st.write(f"**Formula:** {formula}")
    st.write(f"**Explanation:** {explanation}")
    ax.legend()
    st.pyplot(fig)

    fig_3d = go.Figure()
    fig_3d.add_trace(go.Scatter3d(x=X.squeeze(), y=y, z=y_pred_lasso,␣
↪mode='markers', name='Actual Data Points'))
    fig_3d.add_trace(go.Scatter3d(x=X.squeeze(), y=y_pred_lasso,␣
↪z=y_pred_lasso, mode='lines', name='Predicted Line'))
    fig_3d.update_layout(scene=dict(xaxis_title='X', yaxis_title='y',␣
↪zaxis_title='Predicted y'))
    st.write("## 3D Visualization - Lasso Regression")
    st.plotly_chart(fig_3d)
```

1.0.3 ridge

[ ]:
```
def ridge(X, y, alpha):
    ridge_model = Ridge(alpha=alpha)
    ridge_model.fit(X, y)
    y_pred_ridge = ridge_model.predict(X)
    coef = ridge_model.coef_[0]
    intercept = ridge_model.intercept_
    formula = f'y = {coef:.2f}X + {intercept:.2f}'
    explanation = f"In Ridge regression, the penalty term (alpha) is added to␣
↪the square of the coefficients (L2 regularization), which helps in reducing␣
↪the complexity of the model."
```

```
    color = 'green' if alpha < 1 else 'blue'  # Change color based on alpha␣
↪value

    fig, ax = plt.subplots()
    ax.scatter(X, y, color='black', label='Data Points')
    ax.plot(X, y_pred_ridge, color=color, linewidth=2, label='Ridge Regression')
    ax.set_xlabel('X')
    ax.set_ylabel('y')
    st.title(f'Ridge Regression')
    st.write(f"**Formula:** {formula}")
    st.write(f"**Explanation:** {explanation}")
    ax.legend()
    st.pyplot(fig)

    fig_3d = go.Figure()
    fig_3d.add_trace(go.Scatter3d(x=X.squeeze(), y=y, z=y_pred_ridge,␣
↪mode='markers', name='Actual Data Points'))
    fig_3d.add_trace(go.Scatter3d(x=X.squeeze(), y=y_pred_ridge,␣
↪z=y_pred_ridge, mode='lines', name='Predicted Line'))
    fig_3d.update_layout(scene=dict(xaxis_title='X', yaxis_title='y',␣
↪zaxis_title='Predicted y'))
    st.write("## 3D Visualization - Ridge Regression")
    st.plotly_chart(fig_3d)
```

[ ]:
```
np.random.seed(42)
X = np.linspace(1, 10, 200).reshape(-1, 1)
y = 2 * X.squeeze() + np.random.normal(0, 2, 200)
```

1.0.4 Streamlit interface

[ ]:
```
st.sidebar.header('Regression Type')
regression_type = st.sidebar.selectbox('Select Regression Type', ('Linear␣
↪Regression', 'Lasso Regression', 'Ridge Regression'))

if regression_type == 'Linear Regression':
    Linearregression(X, y)
elif regression_type == 'Lasso Regression':
    alpha = st.sidebar.slider('Select Alpha', min_value=0.1, max_value=10.0,␣
↪step=0.1)
    lasso(X, y, alpha)
else: # Ridge Regression
    alpha = st.sidebar.slider('Select Alpha', min_value=0.1, max_value=100.0,␣
↪step=0.5)
    ridge(X, y, alpha)
```

2 2. Gradient Descent Visualization

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import streamlit as st
```

[ ]:
```python
np.random.seed(1234)
num_samples = 100
x1 = np.random.uniform(0, 10, num_samples)
x2 = np.random.uniform(0, 10, num_samples)
intercept = np.ones(num_samples)
error_term = np.random.normal(0, 0.5, num_samples)
```

[ ]:
```python
intercept_true = 1.5
coef1_true = 2
coef2_true = 5
```

[ ]:
```python
y = intercept_true * intercept + coef1_true * x1 + coef2_true * x2 + error_term
feature_matrix = np.vstack([intercept, x1, x2]).T
```

2.0.1 True coefficients

[ ]:
```python
true_coefficients = [intercept_true, coef1_true, coef2_true]
```

2.0.2 Gradient Descent Function

[ ]:
```python
def perform_gradient_descent(learning_rate, iterations, initial_coefficients):
    coefficients = initial_coefficients
    coefficients_history = np.zeros((iterations + 1, 3))
    coefficients_history[0, :] = initial_coefficients

    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = np.dot(feature_matrix, coefficients)
        residuals = y - predictions
        gradient = -2 * np.dot(feature_matrix.T, residuals) / num_samples
        coefficients = coefficients - learning_rate * gradient
        coefficients_history[i + 1, :] = coefficients
        loss_history[i] = np.mean(residuals ** 2)

    return coefficients_history, loss_history
```

2.0.3 Plotting Functions

```
[ ]: def plot_parameters_trajectory(coefficients_history, i, j, axis):
         axis.plot(true_coefficients[i], true_coefficients[j], marker='p',␣
      ↪markersize=10, color='red', label='True Coefficients')
         axis.plot(coefficients_history[:, i], coefficients_history[:, j],␣
      ↪linestyle='--', marker='o', markersize=5, label='Gradient Descent Path')
         axis.set_xlabel(f'Coefficient {i}')
         axis.set_ylabel(f'Coefficient {j}')
         axis.legend()
         axis.grid(True)

     def plot_gradient_descent(coefficients_history, loss_history, learning_rate,␣
      ↪iterations):
         fig, axes = plt.subplots(2, 2, figsize=(12, 12))
         fig.suptitle(f'Gradient Descent Visualization\nLearning Rate:␣
      ↪{learning_rate}, Iterations: {iterations}', fontsize=16)

         plot_parameters_trajectory(coefficients_history, 0, 1, axes[0, 0])
         plot_parameters_trajectory(coefficients_history, 0, 2, axes[0, 1])
         plot_parameters_trajectory(coefficients_history, 1, 2, axes[1, 0])

         axes[1, 1].plot(loss_history, label='Loss Function')
         axes[1, 1].set_xlabel('Iterations')
         axes[1, 1].set_ylabel('Mean Squared Error')
         axes[1, 1].legend()
         axes[1, 1].grid(True)

         st.pyplot(fig)
```

2.0.4 Function to plot the loss function with derivatives and minima/maxima

```
[ ]: def plot_loss_function(loss_history):
         iterations = np.arange(len(loss_history))
         derivatives = np.gradient(loss_history)

         fig, ax = plt.subplots(figsize=(10, 6))
         ax.plot(iterations, loss_history, label='Loss Function', color='blue')
         ax.plot(iterations, derivatives, label='Derivative', color='orange')
         global_min = np.min(loss_history)
         global_min_index = np.argmin(loss_history)
         global_max = np.max(loss_history)
         global_max_index = np.argmax(loss_history)

         ax.plot(global_min_index, global_min, 'go', label='Global Minima')
         ax.plot(global_max_index, global_max, 'ro', label='Global Maxima')
```

```python
    ax.set_xlabel('Iterations')
    ax.set_ylabel('Value')
    ax.legend()
    ax.grid(True)

    st.pyplot(fig)
```

2.0.5 Streamlit Interface

[ ]:
```python
st.title("Interactive Gradient Descent Visualization")

learning_rate = st.sidebar.slider('Learning Rate', min_value=1e-5,␣
↪max_value=1e-1, value=1e-3, step=1e-5, format="%.5f")
iterations = st.sidebar.slider('Number of Iterations', min_value=100,␣
↪max_value=5000, value=1000, step=100)
initial_coefficients = np.array([3.0, 3.0, 3.0])

coefficients_history, loss_history = perform_gradient_descent(learning_rate,␣
↪iterations, initial_coefficients)
plot_gradient_descent(coefficients_history, loss_history, learning_rate,␣
↪iterations)
plot_loss_function(loss_history)
```