Week 5: Linear Regression and Gradient Descent

In Week 5, we delve into the intricacies of linear regression, focusing on efficient methods for large-scale datasets and addressing challenges like overfitting. We explore different optimization algorithms, regularization techniques, and implementation details using the scikit-learn library.

Linear Regression: Model and Loss Function

The linear regression model predicts a continuous target variable (y) based on a linear combination of input features (x_1, x_2, ..., x_m):

[\hat{y} = w_0 + w_1x_1 + w_2x_2 + ... + w_mx_m = w^T x]

where:

- (\hat{y}) is the predicted value.
- (w_0) is the bias (intercept) term.
- (w_i) are the model parameters (weights) associated with each feature (x_i).
- (w) is the weight vector ([w_0, w_1, ..., w_m]^T).
- (x) is the feature vector ([1, x_1, x_2, ..., x_m]^T).

The goal is to learn the optimal weight vector (w) that minimizes the difference between predicted and actual values. We use the mean squared error (MSE) as the loss function:

[J(w) = \frac{1}{2n} \sum_{i=1}^{n} (w^T x_i - y_i)^2]

where (n) is the number of training examples.

Optimization Algorithms

We explore two main approaches for finding the optimal (w):

Normal Equation

The normal equation provides a closed-form solution for (w):

[w = (X^TX)^{-1}X^Ty]

where (X) is the design matrix and (y) is the target vector. While simple, the normal equation's computational complexity is (O(m^3)), making it slow for high-dimensional datasets. It is also computationally expensive for very large datasets that don't fit in memory.

Gradient Descent

Gradient descent is an iterative optimization algorithm that updates the weight vector in the direction of the negative gradient of the loss function:

[w := w - \alpha \nabla J(w)]

where (\alpha) is the learning rate. We discuss three variants:

- **Batch Gradient Descent:** Uses the entire training set to compute the gradient in each iteration. Slow for

large datasets.

- **Mini-batch Gradient Descent:** Uses a small random subset of the training set to compute the gradient. Balances speed and accuracy.
- **Stochastic Gradient Descent (SGD):** Uses only one training example to compute the gradient in each iteration. Fastest but can be noisy.

Regularization

To prevent overfitting, especially in polynomial regression where the number of features increases dramatically, we introduce regularization techniques:

- **L1 Regularization (Lasso):** Adds the L1 norm of the weight vector to the loss function: (J(w) + \lambda ||w||_1). Encourages sparsity (some weights become zero).
- **L2 Regularization (Ridge):** Adds the L2 norm of the weight vector to the loss function: (J(w) + \lambda ||w||_2^2). Shrinks the weights towards zero.
- **Elastic Net:** Combines L1 and L2 regularization.

Implementation with scikit-learn

Scikit-learn provides efficient implementations for linear regression and its variants:

- `LinearRegression`: Implements the normal equation.
- `SGDRegressor`: Implements stochastic gradient descent, supporting various loss functions and penalties (L1, L2, Elastic Net). Well-suited for large datasets.
- `Lasso`: Uses coordinate descent for L1 regularization.
- `Ridge`: Implements L2 regularization.
- `PolynomialFeatures`: Creates polynomial features from existing features.

Code Examples

```
from sklearn.linear_model import LinearRegression, SGDRegressor, Lasso, Ridge
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler

# Linear Regression using Normal Equation
model_normal = LinearRegression()

# Linear Regression using SGD
model_sgd = SGDRegressor(penalty='l2', max_iter=1000)

# Polynomial Regression with Lasso Regularization
poly_lasso = Pipeline([
    ('poly', PolynomialFeatures(degree=2)),
    ('scale', StandardScaler()),
    ('lasso', Lasso(alpha=0.1))
])

# Polynomial Regression with Ridge Regularization
poly_ridge = Pipeline([
    ('poly', PolynomialFeatures(degree=2)),
    ('scale', StandardScaler()),
    ('ridge', Ridge(alpha=0.1))
])
```

Evaluation Metrics

We use the following metrics to evaluate the performance of linear regression models:

- **Mean Squared Error (MSE):** The average squared difference between predicted and actual values.
- **Root Mean Squared Error (RMSE):** The square root of MSE.
- **R-squared (R^2):** Represents the proportion of variance in the dependent variable that is predictable from the independent variables.

Conclusion

In conclusion, Week 5 provided a comprehensive overview of linear regression, encompassing various optimization techniques and regularization methods. We explored the trade-offs between different algorithms, highlighting the efficiency of SGD for large-scale datasets and the importance of regularization to mitigate overfitting. The practical implementation using scikit-learn was demonstrated through code examples, emphasizing the importance of feature scaling and pipeline construction for building robust and accurate models. Prof. Ashish's lectures effectively covered these crucial aspects of linear regression.