Name:- Ujwal Sahu

Branch:- B.VOC(AI&DS)

Division: B

EXPERIMENT NO 8

Title: To understand and implement gradient descent algorithm ,grid search method and model evaluation techniques.

Tools: Vs Code

Theory:

1. Gradient Descent Algorithm

Gradient Descent is an optimization algorithm used to minimize the cost (loss) function in machine learning models. It works by iteratively updating the model parameters in the opposite direction of the gradient (slope) of the loss function with respect to the parameters. The size of the update is controlled by the learning rate.

Key Points:

- Used for optimizing model parameters.
- Involves a learning rate to control step size.
- Goal is to reach the minimum of the loss function.

2. Grid Search Method

Grid Search is a hyperparameter tuning technique that systematically tries all possible combinations of a predefined set of parameters to find the best model performance. It is commonly used with algorithms like SVM, Decision Trees, etc.

Key Points:

- Searches through a “grid” of hyperparameters.
- Evaluates each combination using cross-validation.
- Returns the best parameter set.

3. Model Evaluation Techniques

These techniques are used to assess the performance of a machine learning model to ensure it generalizes well on unseen data. Common metrics and methods include:

- Accuracy, Precision, Recall, F1-Score: For classification tasks.
- Mean Squared Error (MSE), R^2 Score: For regression tasks.
- Confusion Matrix: To visualize classification results.

- Cross-Validation: To test model performance on different data splits.

Key Points:

- Helps determine the effectiveness of the model.
- Prevents overfitting and underfitting.
- Involves both visual and numerical analysis.

2. Code:

1. Gradient Descent

```
import numpy as np import
matplotlib.pyplot as plt
# Generate synthetic data (y = 2x + 1 +
noise) X = 2 * np.random.rand(100, 1) y = 2
* X + 1 + np.random.randn(100, 1) # Add
bias term (X0 = 1)
X_b = np.c_[np.ones((100, 1)), X] #
Cost function (Mean Squared Error)
def compute_cost(theta, X, y):
   return np.mean((X.dot(theta) - y) ** 2) / 2
# Gradient descent def gradient_descent(X, y, theta,
learning_rate, iterations):
   m = len(y)     for _ in
range(iterations):
       gradients = X.T.dot(X.dot(theta) - y) / m
theta -= learning_rate * gradients     return theta #
Initialize theta = np.random.randn(2, 1)  # Random
starting points iterations = 1000 learning_rate = 0.1

# Perform gradient descent theta = gradient_descent(X_b, y,
theta, learning_rate, iterations)
```

Print the result print("Optimal

Parameters:", theta)

Output:

```
temp.py > compute_cost
import numpy as np
import matplotlib.pyplot as plt
# Generate synthetic data (y = 2x + 1 + noise)
X = 2 * np.random.rand(100, 1)
y = 2 * X + 1 + np.random.randn(100, 1)
# Add bias term (X0 = 1)
X_b = np.c_[np.ones((100, 1)), X]
# Cost function (Mean Squared Error)
def compute_cost(theta, X, y):
    return np.mean((X.dot(theta) - y) ** 2) / 2
# Gradient descent
def gradient_descent(X, y, theta, learning_rate, iterations):
    m = len(y)
    for _ in range(iterations):
        gradients = X.T.dot(X.dot(theta) - y) / m
        theta -= learning_rate * gradients
    return theta
# Initialize
theta = np.random.randn(2, 1)  # Random starting points
iterations = 1000
learning_rate = 0.1
# Perform gradient descent
theta = gradient_descent(X_b, y, theta, learning_rate, iterations)
# Print the result
print("Optimal Parameters:", theta)
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\Banty\OneDrive\Desktop\Node_Farm> python -u "c:\Users\Banty\OneDrive\Desktop\Node_Farm\temp.py"
Optimal Parameters: [[1.05260484]
 [1.87689838]]
PS C:\Users\Banty\OneDrive\Desktop\Node_Farm>
```

2. Grid Search Method

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import
LogisticRegression from sklearn.datasets import
load_iris from sklearn.model_selection import
train_test_split
# Load dataset and split into train/test
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3)
# Logistic Regression model model =
LogisticRegression(max_iter=200)
# Hyperparameter grid param_grid = {'C': [0.1, 1, 10]} # Perform grid
search grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train) # Print the best hyperparameters and
score print("Best Hyperparameters:", grid_search.best_params_)
print("Test Accuracy:", grid_search.best_estimator_.score(X_test,
y_test))
```

Output:

3. Model Evaluation (Accuracy, Confusion Matrix and Classification Report)

from sklearn.metrics import confusion_matrix,
classification_report from sklearn.linear_model import
LogisticRegression from sklearn.datasets import load_iris from
sklearn.model_selection import train_test_split
# Load dataset and split
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3)
Train a Logistic Regression model
model =
LogisticRegression(max_iter=200)
model.fit(X_train, y_train) # Predict and
evaluate y_pred = model.predict(X_test)
Print accuracy, confusion matrix, and classification report
print("Accuracy:", model.score(X_test, y_test)) print("Confusion
Matrix:\n", confusion_matrix(y_test, y_pred)) print("Classification
Report:\n", classification_report(y_test, y_pred))

OUTPUT:

temp.py > ...

```python
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
# Load dataset and split
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3)
# Train a Logistic Regression model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
# Predict and evaluate
y_pred = model.predict(X_test)
# Print accuracy, confusion matrix, and classification report
print("Accuracy:", model.score(X_test, y_test))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.87      0.93        15
           2       0.89      1.00      0.94        17

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45

PS C:\Users\Banty\OneDrive\Desktop\Node_Farm>
```

3. Conclusion: In this experiment, we successfully understood and implemented the Gradient Descent algorithm for optimizing model parameters, the Grid Search method for hyperparameter tuning, and various model evaluation techniques to assess model performance. Gradient Descent helped in minimizing the loss function, while Grid Search provided the best combination of hyperparameters. Evaluation metrics such as accuracy, precision, and mean squared error enabled us to measure how well the model performs on unseen data. These techniques together are essential for building accurate and efficient machine learning models.

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				