Machine Learning Report: Regression and Classification

Contents

1 Introduction

Machine Learning (ML) is a subset of artificial intelligence that enables machines to learn patterns from data and make predictions or decisions without being explicitly programmed. This report provides a detailed explanation of the concepts of **Regression** and **Classification**, as covered throughout the course lectures, assignments, and instructional material.

The goal of this report is to explain not only the algorithms themselves, but also the intuition behind how machines learn, how models are trained, how performance is evaluated, and how these ideas fit into a standard machine learning workflow.

2 How Machines Actually Learn

Machine learning is fundamentally an optimization process. The objective of any learning algorithm is to minimize the prediction error by adjusting internal parameters.

2.1 The Optimization Loop

Every machine learning model follows a repeated loop:

1. **Forward Pass**: The model takes input data and produces an output (prediction).
2. **Loss Function**: Measures how far the prediction is from the true value.
3. **Optimizer**: Updates the model parameters to reduce the loss.
4. **Iteration**: This process repeats until the error stops decreasing or reaches a minimum.

2.2 Weights and Biases

The core of a machine learning model consists of numerical parameters called **weights** and **biases**. Learning simply means finding the optimal values for these parameters so that predictions closely match real outcomes.

3 Gradient Descent

Gradient Descent is the most commonly used optimization algorithm in machine learning. It works by moving step-by-step in the direction of the steepest decrease of the error surface.

- The algorithm visualizes error as a surface.

- It moves "downhill" to find the global minimum.
- Each step updates the model parameters.

4 Data Preprocessing

Before training any model, data must be prepared correctly.

4.1 Train-Test Split

To evaluate how well a model generalizes, data is split into:

- Training set (usually 80%)
- Testing set (usually 20%)

```
from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(X, test_size=0.2)
```

4.2 Feature Scaling

Some algorithms are sensitive to feature magnitude. Feature scaling ensures all features are on a similar scale.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

5 Machine Learning Paradigms

5.1 Supervised Learning

Supervised learning uses labeled data, meaning inputs are paired with correct outputs.

- Regression
- Classification

5.2 Unsupervised Learning

Unsupervised learning works with unlabeled data and finds hidden patterns.

- Clustering
- Dimensionality Reduction

6 Regression

6.1 What is Regression?

Regression is used to predict continuous numerical values such as:

- House prices
- Stock prices
- Temperature

6.2 Linear Regression

Linear Regression fits a straight line to the data by learning a slope and intercept.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

The vertical distance between actual points and the predicted line is called the **residual error**.

7 Non-Linear Regression

7.1 Polynomial Regression

When relationships are non-linear, polynomial features are used.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

poly = make_pipeline(
    PolynomialFeatures(3),
    LinearRegression()
)
poly.fit(X_train, y_train)
```

7.2 Overfitting Risk

Higher-degree polynomials increase model complexity and may overfit the training data.

8 Bias-Variance Tradeoff

8.1 Underfitting (High Bias)

Occurs when the model is too simple and fails to capture the trend.

8.2 Overfitting (High Variance)

Occurs when the model is too complex and memorizes noise instead of learning patterns.

8.3 Optimal Model

The ideal model captures the underlying pattern while ignoring random noise.

9 Regression Evaluation Metrics

9.1 Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum |y - \hat{y}|$$

9.2 Mean Squared Error (MSE)

$$MSE = \frac{1}{n}\sum (y - \hat{y})^2$$

9.3 Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{MSE}$$

9.4 R-Squared

Measures how much variance in the data is explained by the model.

```
from sklearn.metrics import mean_absolute_error,
   mean_squared_error
```

10 Classification

10.1 What is Classification?

Classification predicts discrete categories such as:

- Spam vs Not Spam

- Disease detection
- Digit recognition

11 Logistic Regression

Logistic Regression is a classification algorithm that outputs probabilities using the sigmoid function.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
```

12 K-Nearest Neighbors (KNN)

KNN classifies a point based on the majority class among its nearest neighbors.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

12.1 Effect of K

- Low K: Overfitting
- High K: Underfitting

13 Decision Trees

Decision Trees split data using a series of yes/no questions to create pure leaf nodes.

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=5)
dt.fit(X_train, y_train)
```

13.1 Entropy and Gini Impurity

These metrics measure the impurity of a node and guide optimal splits.

14 Support Vector Machines (SVM)

SVM finds the optimal decision boundary that maximizes the margin between classes.

```
from sklearn.svm import SVC
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
```

14.1 Kernel Trick

For non-linear data, kernels like RBF project data into higher dimensions to make separation possible.

15 Classification Evaluation

15.1 Confusion Matrix

- True Positive (TP)
- True Negative (TN)
- False Positive (FP)
- False Negative (FN)

15.2 Precision and Recall

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

15.3 F1-Score

$$F1 = \frac{2PR}{P + R}$$

```
from sklearn.metrics import confusion_matrix,
   classification_report
```

16 Standard Machine Learning Workflow

Regardless of the library used, the workflow remains consistent:

1. Instantiate the model
2. Fit the model on training data
3. Predict on unseen data
4. Evaluate performance

17 Learning Objectives and Key Ideas of Assignment 1

This assignment was designed to introduce fundamentals of Python programming and its core data science libraries.

17.1 Python Programming Fundamentals

The first part of the assignment focuses on core Python concepts such as variables, data types, conditionals, loops, strings, functions, and lists.

17.2 Numerical Computing with NumPy

The NumPy section introduces array-based computation and teaches how to create arrays and matrices, perform mathematical operations, apply indexing and slicing, and work with vectorized data.

17.3 Data Handling with Pandas

In the Pandas section, the assignment emphasizes structured data manipulation using DataFrames and teaches learn how to create tabular data, analyze rows and columns, compute summary statistics, filter data based on conditions, and read external data from CSV files.

17.4 Data Visualization with Matplotlib

The final part introduces data visualization using Matplotlib and teaches plotting line graphs, bar charts, and histograms to visually represent data. This section highlights the importance of visual interpretation of data trends, comparisons, and distributions.

18 Learning Objectives and Key Ideas of the Assignment 2 part 1

This assignment provided an understanding of core machine learning algorithms by implementing them from scratch and comparing them with standard library implementations.

18.1 Understanding Linear Regression from Scratch

The linear regression component focuses on building a regression model without relying on external libraries. It teaches how model parameters such as weights and bias are initialized, updated using gradient descent, and used to make predictions.Then, by applying the model to a real-world house price dataset, the assignment emphasizes data preprocessing, normalization, performance evaluation using Mean Squared Error (MSE) and R-squared, and the importance of hyperparameter tuning.

18.2 Learning Logistic Regression for Binary Classification

The logistic regression section introduces binary classification using a medical dataset. Students learn how probabilities are modeled using the sigmoid function, how gradient descent is applied for classification tasks, and how decision boundaries are learned. It also teaches parameters like accuracy, classification report, and confusion matrix.

18.3 Building Intuition for K-Nearest Neighbors (KNN)

The KNN section focuses on instance-based learning and distance-based classification. By implementing Euclidean distance and testing different values of k.

19 Learning Objectives and Key Ideas of the Assignment 2 part 2

This assignment was helped understanding optimization techniques and classical supervised learning algorithms, combining theoretical reasoning, mathematical intuition, and hands-on experimentation.

19.1 Understanding Stochastic Gradient Descent (SGD)

The first part of the assignment focuses on Stochastic Gradient Descent as an optimization method. The key objective is to understand how the learning rate affects convergence speed, stability, and model performance. Improper learning rates can lead to slow learning

or divergence and learn common strategies for choosing and adapting learning rates during training. This builds intuition about optimization dynamics in machine learning models.

19.2 Comparing Gradient Descent and Stochastic Gradient Descent

By training linear regression models using both Batch Gradient Descent and Stochastic Gradient Descent, the assignment highlights the practical differences between the two approaches. Students learn how batch-based updates lead to smoother and more stable convergence, while stochastic updates introduce noise but improve computational efficiency. This comparison develops an understanding of why SGD is preferred for very large datasets despite its erratic optimization path.

19.3 Decision Trees: Theory and Practice

The decision tree section emphasizes both theoretical foundations and practical implementation. Students learn that decision trees are supervised learning algorithms and understand key concepts such as entropy, information gain, and purity of nodes. Through numerical entropy calculations, students gain insight into how decision trees choose optimal splits. Implementing and visualizing a decision tree using the Iris dataset strengthens interpretability and model understanding.

19.4 Support Vector Machines (SVM)

The SVM section focuses on margin-based classification and geometric intuition. Students learn how SVM constructs optimal decision boundaries by maximizing margins and relying on support vectors. The role of kernel functions is emphasized to explain how SVM handles non-linearly separable data. The assignment also builds understanding of regularization through the parameter C and how it controls the trade-off between margin width and classification errors.

20 Learning Objectives and Key Ideas of the Assignment 3

This assignment is designed to build a deep intuition for **transfer learning**, **model initialization**, and the motivation behind **meta-learning**, particularly Model-Agnostic Meta-Learning (MAML). Rather than focusing only on performance, the assignment emphasizes understanding *why* certain training strategies work better in low-data (few-shot) settings.

20.1 Importance of Initialization in Few-Shot Learning

A key objective of the assignment is to demonstrate that the initial weights of a neural network strongly influence how quickly and effectively it can adapt to new tasks using very few data points. By comparing random initialization with pre-trained models, students learn why good initialization can drastically reduce training time and improve generalization when data is scarce.

20.2 Task-Based Learning Using MNIST

The assignment introduces task-based learning by splitting the MNIST dataset into multiple binary classification tasks. This helps students move away from the traditional single-task mindset and begin thinking in terms of learning across tasks. The use of support and query sets builds intuition for how few-shot learning problems are structured.

20.3 Transfer Learning and Fine-Tuning Strategies

By experimenting with multiple training strategies—training from scratch, fine-tuning from a related task, and fine-tuning from a multi-class classifier—the assignment highlights how knowledge learned from one task can transfer to another. Students develop an understanding of which layers of a neural network capture general features and which are task-specific.

20.4 Comparative Evaluation and Learning Curves

The assignment emphasizes empirical comparison by evaluating different methods using consistent metrics and query sets. Plotting learning curves encourages students to analyze convergence behavior, stability, and adaptation speed rather than relying solely on final accuracy values.

20.5 Building Intuition for Meta-Learning (MAML)

Although full MAML is not explicitly implemented, the assignment builds the conceptual foundation for meta-learning. Students are guided to recognize that MAML seeks a shared initialization that can adapt quickly to many different tasks, even when those tasks are dissimilar. This bridges the gap between standard transfer learning and true meta-learning approaches.

21 Model-Agnostic Meta-Learning (MAML)

Model-Agnostic Meta-Learning (MAML) is a meta-learning framework proposed to enable fast adaptation of machine learning models to new tasks using only a small number of training examples. The key idea behind MAML is to learn a set of model parameters that can be efficiently fine-tuned for a wide range of tasks using standard gradient descent. Unlike many previous meta-learning approaches, MAML does not assume any specific model architecture or learning task, making it applicable to supervised learning, regression, and reinforcement learning.

21.1 Motivation

Human intelligence exhibits the ability to learn new concepts and skills rapidly from limited experience. In contrast, conventional machine learning systems often require large datasets and extensive training. Few-shot learning attempts to bridge this gap by enabling models to generalize from only a handful of examples. Meta-learning, or "learning to learn," addresses this problem by leveraging experience across multiple tasks.

The challenge in few-shot learning lies in integrating prior knowledge while avoiding overfitting to limited new data. MAML addresses this by explicitly optimizing model parameters such that a small number of gradient updates yields strong performance on unseen tasks.

21.2 Meta-Learning Problem Setup

In the meta-learning setting, the objective is to train a model on a distribution of tasks rather than a single task. Each task T_i is defined by:

- A loss function $\mathcal{L}_{T_i}$
- A data distribution $q_i(x)$
- A transition distribution $q_i(x_{t+1}|x_t, a_t)$
- A time horizon H

The model f_θ maps inputs x to outputs a. During meta-training, tasks are sampled from a distribution $p(T)$. For each task, the model is trained on a small support set of K examples and evaluated on a separate query set. The performance on the query set is used to update the model parameters in the meta-learning loop.

21.3 MAML Algorithm

MAML aims to find parameters θ such that a small number of gradient descent steps on a new task leads to good generalization. Given a task T_i, the adapted parameters θ_i' are computed using gradient descent:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(f_\theta)$$

The meta-objective optimizes the performance of the adapted parameters across tasks:

$$\min_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta_i'})$$

The meta-update is performed using stochastic gradient descent:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta_i'})$$

This requires computing gradients through gradients, resulting in second-order derivatives.

21.4 Algorithm Description

22 Applications of MAML

22.1 Supervised Regression and Classification

In supervised learning tasks, the horizon $H = 1$, and each task consists of i.i.d. input-output pairs. For regression tasks, the loss function is mean squared error:

$$\mathcal{L}_{T_i}(f_\phi) = \sum_{(x,y) \sim T_i} \|f_\phi(x) - y\|^2$$

For classification tasks, cross-entropy loss is used:

$$\mathcal{L}_{T_i}(f_\phi) = \sum_{(x,y) \sim T_i} y \log f_\phi(x) + (1-y) \log(1 - f_\phi(x))$$

MAML enables rapid adaptation in both cases by learning task-agnostic representations that are easily fine-tuned.

22.2 Reinforcement Learning

In reinforcement learning, each task corresponds to a Markov Decision Process (MDP). The policy f_θ maps states to action distributions. The task loss is defined as the negative expected return:

$$\mathcal{L}_{T_i}(f_\phi) = -\mathbb{E}_{x_t, a_t \sim f_\phi} \left[\sum_{t=1}^{H} R_i(x_t, a_t) \right]$$

Policy gradient methods are used to estimate gradients since environment dynamics are unknown. MAML significantly accelerates policy adaptation to new environments.

23 Experimental Evaluation

23.1 Regression Experiments

The regression experiment uses sinusoidal functions with varying amplitudes and phases. MAML is trained to adapt to new sine waves using only a few data points. Results demonstrate that MAML learns the underlying structure of sine functions, enabling extrapolation beyond observed data.

23.2 Few-Shot Classification

MAML is evaluated on Omniglot and MiniImagenet datasets under N-way K-shot settings. A convolutional neural network architecture is used. MAML achieves state-of-the-art or comparable performance while using fewer parameters than competing meta-learning methods.

23.3 Reinforcement Learning Experiments

MAML is tested on navigation and locomotion tasks using simulated environments. Tasks vary in goal position, direction, or velocity. The results show that MAML-initialized policies adapt significantly faster than randomly initialized or pretrained policies.

24 First-Order Approximation

To reduce computational cost, a first-order approximation of MAML is evaluated by ignoring second-order derivatives. Surprisingly, this approximation achieves nearly identical performance while reducing computation time by approximately 33%.

25 Related Work

MAML is compared to prior meta-learning methods such as recurrent meta-learners, metric-based approaches, and learned optimizers. Unlike these methods, MAML does not introduce additional parameters or architectural constraints, making it broadly applicable.

26 Discussion and Conclusion

MAML provides a simple yet powerful framework for meta-learning by optimizing model parameters for fast adaptation. Its model-agnostic nature allows it to be applied across supervised learning and reinforcement learning domains. Experimental results demonstrate that MAML enables rapid learning from few examples while maintaining strong generalization.

The ability to reuse prior knowledge efficiently makes MAML a promising approach for scalable and adaptable machine learning systems. Future work may further explore its integration with large-scale models and complex task distributions.