**Summer of Science 2025**  
**Mid-Term Report**

Drishtant Jain (24M1085)  
CS01 – Artificial Intelligence & Machine Learning

June 2025

# 1. Introduction

Artificial Intelligence and Machine Learning have become cornerstones of technological progress in the 21st century. Their influence spans across industries—ranging from healthcare and finance to transportation and entertainment. This report encapsulates my learnings from the first four weeks of the Summer of Science 2025 program under the AI/ML domain. The objective of this phase was to build a solid mathematical and conceptual understanding of supervised learning, regression, classification, and dimensionality reduction methods.

The core resources used during this period include the “Hundred-Page Machine Learning Book” by Andriy Burkov and the NPTEL Machine Learning course by Prof. Balaraman Ravindran. Weekly hands-on implementations were carried out in Python using libraries such as NumPy, pandas, scikit-learn, and matplotlib. This report also includes insights from three mini-projects developed using real-world datasets.

# 2. Week 1: Foundations of Supervised Learning

## 2.1 Overview

Supervised learning is the foundational paradigm in machine learning where the model learns to map inputs to outputs using labeled datasets. In Week 1, I explored the principles behind regression and classification, particularly focusing on statistical decision theory, the bias-variance tradeoff, and optimization concepts.

## 2.2 Statistical Decision Theory

The foundation of supervised learning lies in minimizing expected loss. Let be the input space and the output space. Given a data distribution and a loss function , the goal is to find a function that minimizes the risk:

For regression, the squared error loss is commonly used:

Under this loss, the Bayes optimal predictor is:

For classification with 0-1 loss, the Bayes optimal classifier is:

## 2.3 Bias-Variance Tradeoff

Understanding model generalization requires an analysis of the bias-variance decomposition of prediction error:

- **Bias** measures the error introduced by approximating a real-world problem with a simplified model. - **Variance** measures the model’s sensitivity to fluctuations in the training data.

High bias leads to underfitting; high variance leads to overfitting. Balancing this tradeoff is central to model selection.

## 2.4 Linear Regression

Linear regression is used to model a relationship between a scalar response and one or more explanatory variables. The model is expressed as:

The optimal parameters minimize the mean squared error (MSE):

## 2.5 Gradient Descent Optimization

To minimize , gradient descent iteratively updates parameters:

Where is the learning rate. Proper tuning of is critical: too small slows convergence; too large causes divergence.

## 2.6 Hands-on Implementation

* Implemented Linear Regression on synthetic and real datasets using both analytical and gradient descent methods.
* Visualized error surface and gradient descent path.
* Experimented with varying learning rates and initial weights.

## 2.7 Code Snippet: Gradient Descent for Linear Regression

# Gradient Descent for Linear Regression  
for i in range(num\_iterations):  
 predictions = X.dot(w) + b  
 errors = predictions - y  
 w -= alpha \* (1/n) \* X.T.dot(errors)  
 b -= alpha \* (1/n) \* np.sum(errors)

## 2.8 Observations and Takeaways

- Gradient Descent converges more reliably for convex loss surfaces. - Visualizing cost over iterations aids in debugging. - Model performance improved with feature normalization.

# 3. Week 2: Linear Models and Dimensionality Reduction

## 3.1 Overview

In Week 2, I explored advanced linear modeling techniques and unsupervised methods for reducing dimensionality. The primary topics were:

* Multivariate Linear Regression
* Regularization Techniques: Ridge and Lasso
* Dimensionality Reduction: Principal Component Analysis (PCA)
* Feature Selection vs Feature Extraction

These methods are essential when working with high-dimensional data to prevent overfitting, reduce noise, and improve model interpretability and computational efficiency.

## 3.2 Multivariate Linear Regression

Multivariate Linear Regression extends simple linear regression to multiple input features. The model is:

Where:

* : feature vector
* : weight vector
* : bias term

The parameters are optimized by minimizing the Mean Squared Error:

This is convex and solvable using closed-form (Normal Equation) or numerical optimization.

## 3.3 Regularization: Ridge and Lasso

High-dimensional models are prone to overfitting. Regularization techniques penalize large weights and promote simpler models.

### Ridge Regression (L2 Regularization)

**Effect:** Shrinks coefficients but does not reduce any of them exactly to zero.

### Lasso Regression (L1 Regularization)

**Effect:** Can shrink some coefficients exactly to zero — hence performs variable selection.

### Hyperparameter

Controls trade-off between fit and penalty:

* Small = less regularization
* Large = stronger penalty

**Note:** Ridge is better when all features are useful. Lasso is better when we expect sparsity.

## 3.4 Dimensionality Reduction with PCA

Principal Component Analysis (PCA) is a technique to reduce data dimensions by projecting it onto directions that maximize variance.

### Mathematics of PCA

1. Center the data:
2. Compute covariance matrix:
3. Find eigenvectors of
4. Project data onto top- eigenvectors:

The top principal components correspond to the largest eigenvalues.

**Key Insight:** PCA performs unsupervised feature extraction that preserves variance.

## 3.5 Feature Selection vs Extraction

* **Feature Selection:** Select a subset of existing features (e.g., via correlation, mutual information, RFE)
* **Feature Extraction:** Create new features (e.g., PCA, Autoencoders)

## 3.6 Hands-on Implementation and Experiments

* Implemented Ridge and Lasso using sklearn.linear\_model
* Visualized coefficient shrinkage and compared validation errors
* Performed PCA on the Wine dataset to reduce dimensionality and visualize class separation
* Compared feature selection (SelectKBest) with PCA (extraction)

## 3.7 Code Snippets

### Ridge and Lasso

from sklearn.linear\_model import Ridge, Lasso  
ridge = Ridge(alpha=1.0)  
lasso = Lasso(alpha=0.1)  
ridge.fit(X\_train, y\_train)  
lasso.fit(X\_train, y\_train)

### Principal Component Analysis (PCA)

from sklearn.decomposition import PCA  
pca = PCA(n\_components=2)  
X\_pca = pca.fit\_transform(X\_scaled)

### Feature Selection (Univariate)

from sklearn.feature\_selection import SelectKBest, f\_regression  
selector = SelectKBest(score\_func=f\_regression, k=5)  
X\_new = selector.fit\_transform(X, y)

## 3.8 Observations

* Lasso resulted in sparse models — some coefficients exactly zero.
* Ridge maintained all coefficients but controlled overfitting better.
* PCA helped reduce dimensions without much performance loss.
* Feature selection methods like SelectKBest worked well when some features were noisy.

# 4. Week 3: Classification and Clustering Methods

## 4.1 Overview

In Week 3, the focus shifted to classification algorithms and unsupervised clustering. This week’s curriculum built on previous linear models and introduced:

* Logistic Regression
* K-Nearest Neighbors (KNN)
* K-Means Clustering
* Perceptron Learning Algorithm

I also developed three mini-projects that applied these methods to real-world problems.

## 4.2 Logistic Regression (Binary Classification)

Despite its name, logistic regression is a classification algorithm. It models the probability of a binary outcome using the sigmoid (logistic) function:

The model is trained to minimize the log-loss or binary cross-entropy:

Gradient Descent is used to update weights:

**Key Insight:** The sigmoid function maps inputs to probabilities in the (0,1) interval — allowing thresholding.

### Code Snippet: Logistic Regression

from sklearn.linear\_model import LogisticRegression  
model = LogisticRegression()  
model.fit(X\_train, y\_train)  
preds = model.predict(X\_test)

## 4.3 K-Nearest Neighbors (KNN)

KNN is a non-parametric algorithm that classifies based on proximity in feature space:

**Distance Metric:**

### Hyperparameter:

- Low may overfit (sensitive to noise) - High may underfit (over-smooth)

### Elbow Method

Used to find the optimal by plotting error vs .

### Code Snippet: KNN

from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n\_neighbors=5)  
knn.fit(X\_train, y\_train)  
preds = knn.predict(X\_test)

## 4.4 Perceptron Algorithm

The perceptron is an early binary linear classifier. Given input and label :

**Update Rule (if misclassified):**

**Limitation:** Converges only if data is linearly separable.

### Code Snippet: Perceptron

from sklearn.linear\_model import Perceptron  
clf = Perceptron()  
clf.fit(X\_train, y\_train)

## 4.5 K-Means Clustering

K-Means is an unsupervised learning algorithm for grouping similar data points into clusters.

**Steps:**

1. Initialize centroids randomly
2. Assign each point to nearest centroid (using Euclidean distance)
3. Recalculate centroids as the mean of assigned points
4. Repeat until convergence

**Objective Function:**

where is the cluster assignment of point .

### Code Snippet: K-Means

from sklearn.cluster import KMeans  
kmeans = KMeans(n\_clusters=3)  
clusters = kmeans.fit\_predict(X)

## 4.6 Mini Projects

### 1. Spam Classifier using Logistic Regression

**Objective:** Build a supervised classification model to predict whether an SMS message is spam or legitimate (ham).

**Dataset:** The project uses the well-known SMSSpamCollection.txt dataset from UCI. It contains over 5,500 labeled SMS messages with two columns:

* label: either “ham” or “spam”
* text: the message content

**Preprocessing Steps:**

1. Loaded dataset using pandas with ‘  
   t‘ as the separator.
2. Mapped categorical labels to binary format: spam = 1, ham = 0.
3. Applied TfidfVectorizer to convert messages into numerical vectors:

* Where TF is term frequency, DF is document frequency, and is the total number of documents.

**Modeling:** Used Scikit-learn’s LogisticRegression, suitable for binary classification. The model estimates the probability:

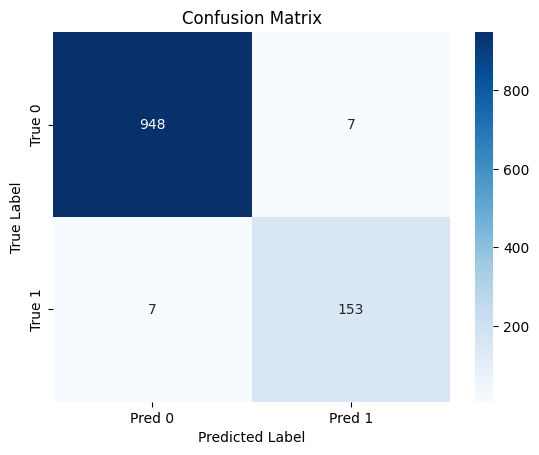
where is the sigmoid function.

**Evaluation:**

* Split the dataset into 80% training and 20% testing.
* Evaluated using precision, recall, F1-score, and confusion matrix.
* Achieved accuracy of over 97%.

**Key Insights:**

* TF-IDF handles sparse word representations efficiently.
* Logistic Regression is interpretable and fast, ideal for text classification.
* The confusion matrix indicated high recall for spam — critical for real-world filtering.



Confusion Matrix for Spam Classifier

### 2. Wine Classifier using K-Nearest Neighbors and Logistic Regression

**Objective:** To classify wine samples into their respective classes based on physicochemical attributes using supervised classification models.

**Dataset:** The project uses the load\_wine() dataset from sklearn.datasets. This is a well-known, structured dataset consisting of:

* **178 samples**
* **13 continuous features**, such as alcohol content, flavanoids, and magnesium levels
* **3 target classes**: class\_0, class\_1, and class\_2 (corresponding to three different cultivars of wine)

**Exploratory Data Analysis (EDA):**

* Created a pandas DataFrame and appended the target class.
* Visualized the distribution of features using seaborn histograms and pairplots.
* Checked for multicollinearity using the correlation matrix:

**Preprocessing:**

* Standardized the feature set using StandardScaler to ensure all variables contribute equally.
* Split the dataset into 80% training and 20% testing using train\_test\_split().

**Models Implemented:**

1. **Logistic Regression:** A multinomial logistic regression classifier was trained using one-vs-rest strategy.
2. **K-Nearest Neighbors (KNN):** A non-parametric model that classifies based on the majority vote of nearest samples in feature space.

**Model Evaluation:**

* Compared classification accuracy, precision, and recall between Logistic Regression and KNN.
* Evaluated models using a confusion matrix and classification report.

**Code Snippet (KNN):**

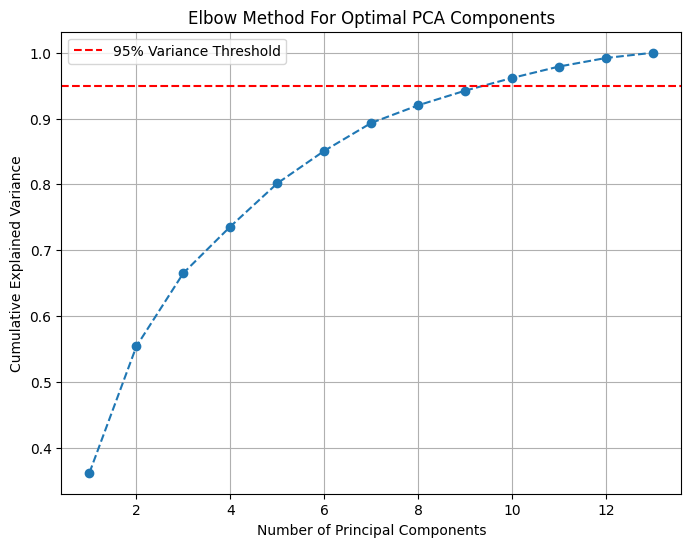
from sklearn.neighbors import KNeighborsClassifier  
model = KNeighborsClassifier(n\_neighbors=5)  
model.fit(X\_train, y\_train)  
model.score(X\_test, y\_test)

**Results:**

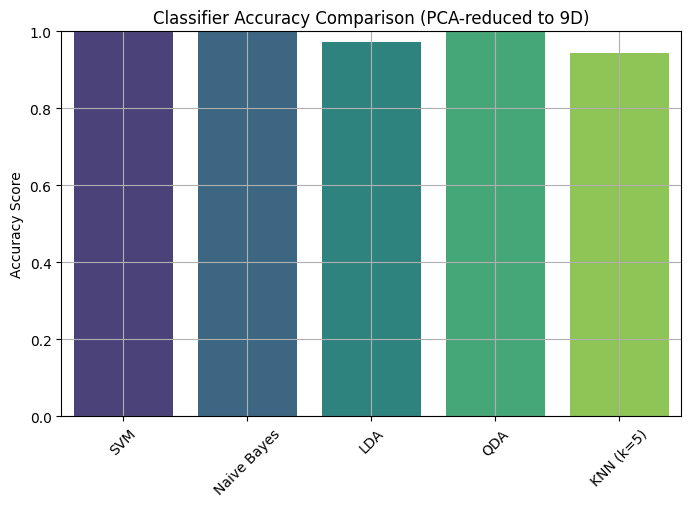
* Both models achieved accuracy above 95% on the test set.
* Logistic regression provided smoother decision boundaries and was faster to train.
* KNN was sensitive to the value of and benefited significantly from feature scaling.

**Key Insights:**

* Feature scaling is critical for distance-based models like KNN.
* Logistic regression can be extended to multiclass problems using softmax.
* Dataset features like alcohol and flavanoids had strong correlation with target classes.



Elbow Method used to determine the optimal number of components using PCA.This was done to reduce the dimensionality.



Classifier Accuracy Comparison for Wine Classification

### 3. Dominant Color Extraction using K-Means Clustering

**Objective:** Extract the most visually dominant colors in an image using unsupervised clustering. This technique is used in design platforms (e.g., Canva) for generating color palettes and themes.

**Approach:**

1. Loaded an image using OpenCV.
2. Converted it from BGR (OpenCV default) to RGB color space.
3. Reshaped the image from to , where is the number of pixels and 3 represents RGB channels.

**Clustering with K-Means:** Each pixel was treated as a point in 3D RGB space. The K-Means algorithm grouped similar colors:

Where:

* are cluster centers (colors)
* is the cluster assignment of pixel

**Implementation:**

* Used sklearn.cluster.KMeans with n\_clusters=5.
* Extracted the RGB values of the cluster centers as the dominant colors.
* Visualized these colors as square patches using matplotlib.

**Applications:**

* Auto-generating color themes from images.
* Simplifying backgrounds in image editing.
* Extracting brand palettes from logos or designs.

**Key Insights:**

* K-Means is highly effective for pixel clustering.
* Initial centroid placement can affect final output — a common caveat.
* This kind of application can be seen on Canva where it gives the dominant colors available when we upload a picture.



Dominant Colors Extracted from an Image. Here K was taken as 10.

## 4.7 Observations and Reflections

* Logistic Regression is interpretable and robust on sparse data.
* KNN’s performance depends heavily on and distance metric.
* Perceptron is conceptually simple but limited to linearly separable data.
* K-Means is efficient but sensitive to initial centroid placement.

# 5. Week 4: Neural Networks and Tree-Based Models

## 5.1 Overview

This week was dedicated to foundational neural network concepts and decision tree-based models. Both paradigms offer powerful yet interpretable solutions for classification and regression. I focused on:

* Perceptron and Multilayer Perceptrons (MLPs)
* Backpropagation and activation functions
* Decision Trees and pruning strategies
* Evaluation metrics like precision, recall, and F1-score

## 5.2 Perceptron Model

The Perceptron is one of the earliest and simplest neural classifiers.

**Model:**

**Update Rule (for binary classification with ):**

Where is the learning rate.

**Limitation:** Converges only for linearly separable datasets. Cannot model XOR-type problems.

## 5.3 Introduction to Neural Networks

A feedforward neural network (also called Multilayer Perceptron or MLP) consists of:

* An input layer
* One or more hidden layers with non-linear activation functions
* An output layer (e.g., sigmoid, softmax)

**Forward Propagation:** For one hidden layer:

Where and are activation functions such as ReLU or sigmoid.

**Loss Function (e.g., Cross-Entropy for binary classification):**

## 5.4 Backpropagation Algorithm

To optimize weights, we use backpropagation and gradient descent:

**Steps:**

1. Perform forward pass to compute
2. Compute loss
3. Compute gradients using chain rule
4. Update weights:

**Key Insight:** The chain rule allows error signals to flow from output back to inputs.

## 5.5 Activation Functions

* Sigmoid: (used in binary classification)
* ReLU: (preferred in deep networks)
* Tanh:

## 5.6 Decision Trees

Decision Trees split data based on feature values to form a tree-like structure of decision rules.

**Splitting Criteria:**

* **Entropy:**
* **Information Gain:**
* **Gini Index:**

**Pruning:** Reduces overfitting by removing branches with low importance or few samples. Can be:

* Pre-pruning: Stop growing based on criteria (e.g., max depth)
* Post-pruning: Prune after full tree is built using validation accuracy

### Code Snippet: Decision Tree

from sklearn.tree import DecisionTreeClassifier  
dtree = DecisionTreeClassifier(criterion='entropy', max\_depth=5)  
dtree.fit(X\_train, y\_train)

## 5.7 Model Evaluation Metrics

* **Accuracy:**
* **Precision:**
* **Recall:**
* **F1 Score:** Harmonic mean of precision and recall

### Code Snippet: Evaluation

from sklearn.metrics import classification\_report, confusion\_matrix  
print(confusion\_matrix(y\_test, y\_pred))  
print(classification\_report(y\_test, y\_pred))

## 5.8 Observations and Reflections

* Neural networks are flexible but require careful tuning (learning rate, layers).
* Activation functions play a crucial role in convergence.
* Decision trees are interpretable but can easily overfit if not pruned.
* Evaluation metrics give a clearer picture than accuracy alone, especially with class imbalance.

# 3. Tools and Technologies Used

* Python (NumPy, pandas, matplotlib, seaborn, scikit-learn)
* Jupyter Notebooks , Google Colab
* OpenCV (for image-based projects)

# 4. Next Steps

As I conclude the midterm phase of the Summer of Science 2025 program, the next four weeks will focus on extending my understanding of machine learning into ensemble techniques, probabilistic models, and deep learning. Additionally, I will begin working on the implementation and refinement of my final project — a collaborative filtering-based movie recommender system.

## Week 5: Ensemble Learning and Evaluation

* Study and implement ensemble methods such as Bagging, Boosting, Random Forests, and Stacking.
* Understand how ensemble methods reduce variance and bias.
* Compare performance of individual models vs ensembles.
* Evaluate models using ROC curve, AUC score, and advanced metrics.

## Week 6: Probabilistic Models and Bayesian Learning

* Explore Naive Bayes classifier and its extensions for multi-class problems.
* Understand Bayes Theorem and its application in classification.
* Learn about probabilistic graphical models like Bayesian Networks.
* Perform feature engineering to enhance performance of probabilistic models.

## Week 7: Neural Networks and Deep Learning Foundations

* Deepen understanding of Perceptron and Multilayer Perceptrons (MLPs).
* Learn the mechanics of gradient descent and backpropagation.
* Study activation functions, weight initialization strategies, and optimization techniques.
* Begin implementing Convolutional Neural Networks (CNNs) for image-based tasks.

## Week 8: Backpropagation and Regularization in Deep Learning

* Implement and experiment with various optimizers (SGD, Adam, RMSProp).
* Study regularization methods including dropout and early stopping to prevent overfitting.
* Complete final implementation of deep learning models and validate them using test datasets.
* Begin finalizing and documenting the collaborative filtering-based movie recommender project.

## Final Project Objective

Build a movie recommendation system using collaborative filtering on the MovieLens dataset. The system will combine:

* Cosine similarity-based nearest neighbor recommendations
* Matrix Factorization using Singular Value Decomposition (SVD)
* Evaluation using ranking metrics like precision@k and RMSE
* Optional UI deployment using Streamlit

# 5. GitHub Repository

Code for all implementations and mini-projects is available at:  
<https://github.com/DrishtantGithub/SummerOfScience2025>

# 6. References

1. Andriy Burkov, *The Hundred-Page Machine Learning Book*, 2019.
2. Balaraman Ravindran, *NPTEL Machine Learning Course*.
3. Machine Learning Essentials — *Udemy Course by Coding Minutes*.
4. GeeksforGeeks: <https://www.geeksforgeeks.org/> For different libraries and quick implementation references.