Assignment 1 – Muhammad Taha

Question 1: Support Vector Machines (SVM) – Decision Boundaries Given a dataset where classes are not linearly separable, explain how SVM can still find an optimal decision boundary. Discuss the role of the Kernel Trick and compare the performance of Linear, Polynomial, and RBF kernels on high-dimensional data.

Bonus: Implement an SVM model using the dataset provided and compare accuracy with different kernels.

Solution:

How SVM Works with Non-Linearly Separable Data

Support Vector Machines (SVMs) aim to find the **optimal decision boundary** (also called a hyperplane) that separates classes with the maximum margin.

- If the dataset is **linearly separable**, a straight hyperplane works.
- But when classes are **not linearly separable**, SVM applies two key ideas:
    1. **Slack Variables (Soft Margin SVM):** Allow some misclassification for flexibility.
    2. **Kernel Trick:** Transform data into a higher-dimensional space where it may become linearly separable.

Kernel Trick

The **Kernel Trick** avoids explicitly computing transformations into high-dimensional space. Instead, it uses a kernel function to compute similarity (dot products) between data points in the transformed space.

This makes SVM powerful because:

- It can separate complex datasets without explicitly computing high-dimensional mappings.
- It only relies on kernel functions, saving computation.

Common Kernels and Their Performance

1. **Linear Kernel**
    - Best for: High-dimensional but *linearly separable* data (e.g., text classification, bag-of-words models).
    - Pros: Fast, interpretable, less risk of overfitting.
    - Cons: Cannot handle non-linear boundaries.
2. **Polynomial Kernel**
    - Best for: Data where relationships are polynomial in nature.
    - Pros: Captures more complex relationships.
    - Cons: Higher degree, overfitting & computationally expensive.

3. **RBF (Radial Basis Function) / Gaussian Kernel**
    - Best for: Most real-world non-linear datasets.
    - Pros: Very flexible, captures non-linear boundaries well.
    - Cons: Requires tuning hyperparameters C and gamma.

Performance Comparison on High-Dimensional Data

- **Linear Kernel**: works well when data is high-dimensional but linearly separable (e.g., text classification).
- **Polynomial Kernel**: can become computationally expensive and may overfit in high-dimensional data.
- **RBF Kernel**: generally, performs the best for complex, non-linear boundaries, but needs careful tuning.

Implementation: In code file task1.ipynb

Question 2: Random Forest – Overfitting vs. Generalization Explain how Random Forest mitigates overfitting compared to a single Decision Tree. Discuss the significance of Bootstrapping and Feature Selection in improving model performance.

Practical Task: Train a Random Forest model on the provided dataset and analyze feature importance. Which features contribute most to loan approval?

Solution:

Random Forest: Overfitting vs. Generalization

Single Decision Tree

- A decision tree learns rules to classify data.
- It keeps splitting until it perfectly classifies the training data, **overfitting** (memorizing noise).
- High variance, performs poorly on unseen data.

Random Forest

A Random Forest is an **ensemble of decision trees** where predictions are made by majority voting (classification) or averaging (regression). It reduces overfitting through two main techniques:

1. **Bootstrapping (Bagging)**
    - Each tree is trained on a different **bootstrap sample** (random sample with replacement) of the dataset.
    - This introduces diversity among trees.
    - Some data points may appear multiple times in one tree's training set, others not at all.
    - Ensures trees are **uncorrelated** and reduces variance.
2. **Feature Selection (Random Subspace Method)**
    - At each split, only a **random subset of features** is considered.
    - Prevents strong predictors from dominating all trees.
    - Increases diversity and helps the model generalize better.

Implementation: In code file task2.ipynb

Question 3: Confusion Matrix and ROC Curve – Model Evaluation A credit scoring model has the following confusion matrix for predicting high default risk:

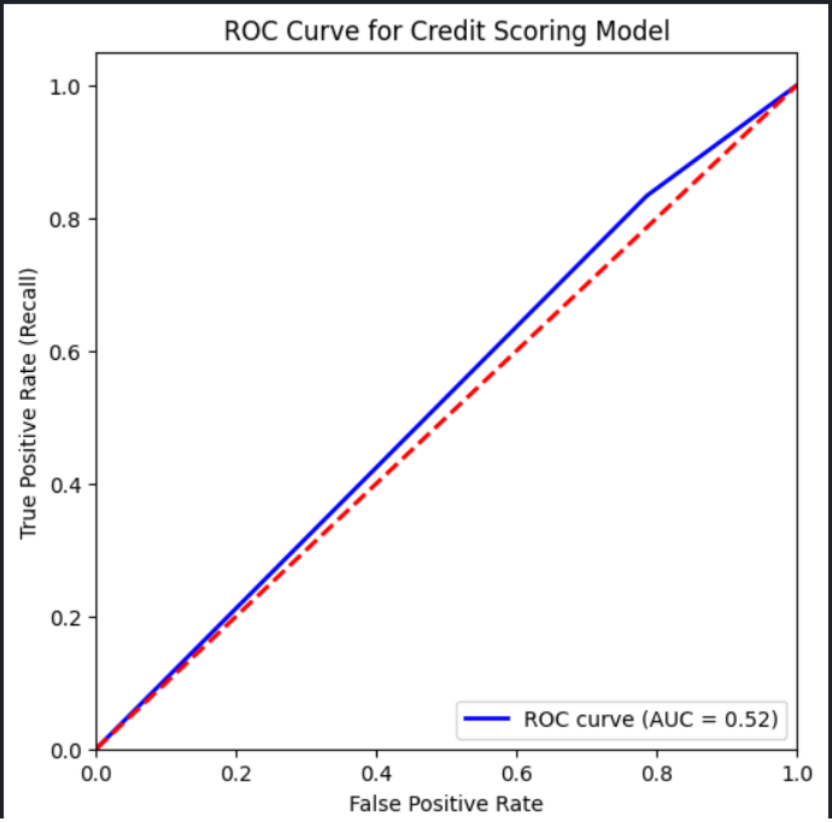
Predicted ↓ / Actual →	Default (1)	No Default (0)
Default (1)	50	30
No Default (0)	10	110

- Calculate Accuracy, Precision, Recall, and F1-Score for this model.
- Draw the ROC Curve for this model assuming the model provides probability scores.
- Interpretation: What trade-offs are observed in Precision vs. Recall? How would you adjust the threshold to reduce false positives?

Solution:

Implementation: In code file task3.ipynb

```
Model Evaluation Metrics:
Accuracy: 0.8000
Precision: 0.6250
Recall: 0.8333
F1-Score: 0.7143
```



Interpretation: Precision vs Recall Trade-off

- **High Recall (83.3%)** → Model is catching most defaults (good for risk prevention).
- **Moderate Precision (62.5%)** → But many predicted defaults are actually non-defaults (false positives).

Trade-off:

- If we **increase threshold** → fewer defaults predicted, reducing **false positives** → Precision ↑ but Recall ↓.
- If we **decrease threshold** → more defaults predicted, catching more true defaults → Recall ↑ but Precision ↓.

Question 4: K-Nearest Neighbors (KNN) – Choosing Optimal K In KNN, selecting the appropriate number of neighbors (K) is crucial for achieving a good balance between bias and variance. Explain the consequences of choosing a very small vs. very large K.

Task: Implement KNN on the given dataset and use cross-validation to determine the optimal K. Analyze how accuracy varies with different values of K.

Solution:

KNN: Choosing Optimal K

1. Very Small K (e.g., K=1 or 2)

- **Low bias, high variance.**
- Model fits training data very closely (almost memorizes it).
- Very sensitive to noise → a single mislabeled point can drastically affect predictions.
- Overfitting risk.

2. Very Large K (e.g., close to dataset size)

- **High bias, low variance.**
- Predictions become overly smooth (majority class dominates).
- Model may ignore local patterns.
- Underfitting risk.

Optimal K balances bias–variance, often found using **cross-validation**.

Implementation: In code file task4.ipynb

Question 5: Bayesian Belief Networks (BBN) – Probabilistic Decision-Making A bank uses a Bayesian Belief Network to assess the probability of loan default based on factors such as income, credit score, and previous defaults.

- Explain how Conditional Independence can simplify the probability calculations in this network.
- Given prior probabilities, construct a simple Bayesian Network with 3 nodes: Loan Approval, Income Level, and Credit Score.

Implementation Challenge: Use pgmpy in Python to model a Bayesian Network for loan approval. Compute the probability of approval given a high income but a low credit score.

Solution:

Conditional Independence in Bayesian Belief Networks

In a Bayesian Belief Network, we often deal with many factors that affect an outcome. For example, a bank may consider **income**, **credit score**, and **previous defaults** to predict whether a person will **default on a loan**.

If we tried to calculate the probability of every possible combination of these factors directly, it would require storing and computing a huge number of probabilities, which quickly becomes too complex as more factors are added.

Conditional Independence is the key idea that makes Bayesian Networks practical. It means that:

Once we know the important "parent" factor, some variables don't directly depend on each other anymore.

For example:

- If we already know whether a loan was approved, then a person's **income** and **credit score** don't directly affect each other in the calculation.
- Instead of storing probabilities for every possible situation, we only need to store smaller, local probability tables (e.g., how income affects approval, how credit score affects approval).

This greatly **simplifies the calculations**, because the network can combine these smaller pieces to answer big questions, like:

- *"What's the probability of default if the person has high income but a poor credit score?"*

Simple Bayesian Network with 3 Nodes

We’ll design:

- **Income Level** → affects Loan Approval.
- **Credit Score** → affects Loan Approval.
- **Loan Approval** → depends on both.

Implementation: In code file task5.ipynb