ALDA Fall 2025
HW4
Due: 10/30/2025

TEAM ID : H24 MEMBER : Yujin Kim, ykim68

HW4 contains 6 questions. Please read and follow the instructions.

- **DUE DATE FOR SUBMISSION: 10/30/2025 11:45 PM**
- **TOTAL NUMBER OF POINTS: 130**
- **NO PARTIAL CREDIT** will be given so provide concise answers.
- **Submissions and updates should be handled by the same person.**
- **You MUST manually add ALL team members in the submission portal when you submit through Gradescope. One submission per group. Team member(s) who are left out will lose 20 points if added after the deadline.**
- Make sure you clearly list **your homework team ID, all team members' names, and Unity IDs, for those who have contributed to the homework contribution** at the top of your submission.
- **[GradeScope and Github]:** Submit a PDF on GradeScope. **You must submit your code in Github, and give the instructors access.**

https://github.com/ykim68ncstate/ncsu-engr-ALDA-Fall2025-H24

https://github.com/ykim68ncstate/ncsu-engr-ALDA-Fall2025-H24

[GradeScope and GitHub Submission Instructions for Coding Questions]: For coding questions, you must submit your written solutions as a PDF on GradeScope and also provide your code in a private GitHub repository with instructor access.

1. **Create a private GitHub repository for your group.** Name your repository using the following format:

   `ncsu-engr-ALDA-Fall2025-HXX`

   Replace `XX` with your homework group number. **Example:** `ncsu-engr-ALDA-Fall2025-H2`. You will use one homework repository for the whole semester: do not create a new repository for each homework.
2. **Set up the repository:**
   - Log in to GitHub and click the green **"New"** button.
   - Use the required naming convention.
   - Set the repository to **private**.
   - Click **"Create repository"**.
3. **Add collaborators:**
   - Navigate to **Settings** > **Collaborators**.
   - Add all group members.
   - **Add the instructors and TAs.**
4. **Organize your code:**
   - Create separate folders for each of five assignments (e.g., `HW1`, `HW2`, etc.).
   - Place all relevant code in the correct folder.
   - Upload your complete `.py` files **before the submission deadline**. Late or missing code will not be graded.
5. **Include your GitHub repository link in the GradeScope PDF.**
6. **Clearly reference your code in the PDF:**
   - Indicate the relevant file for each question.
     Example: "The solution to Question 2 is in `matrix.py`."
   - Code must be executable; outputs must be generated by running the file.
   - **Do not hardcode results.** Submissions without proper computation will receive no credit.
7. **Code excerpts and outputs:** Your PDF must include output for each part and key code snippets (not full scripts) that demonstrate your implementation logic.

TEAM ID : H24 MEMBER : Yujin Kim, ykim68

https://github.com/ykim68ncstate/ncsu-engr-ALDA-Fall2025-H24

1. (15 points) [**BN Inference**] [**Graded By Ian Holmes**] Compute the following probabilities according to the Bayesian net shown in Figure 1.

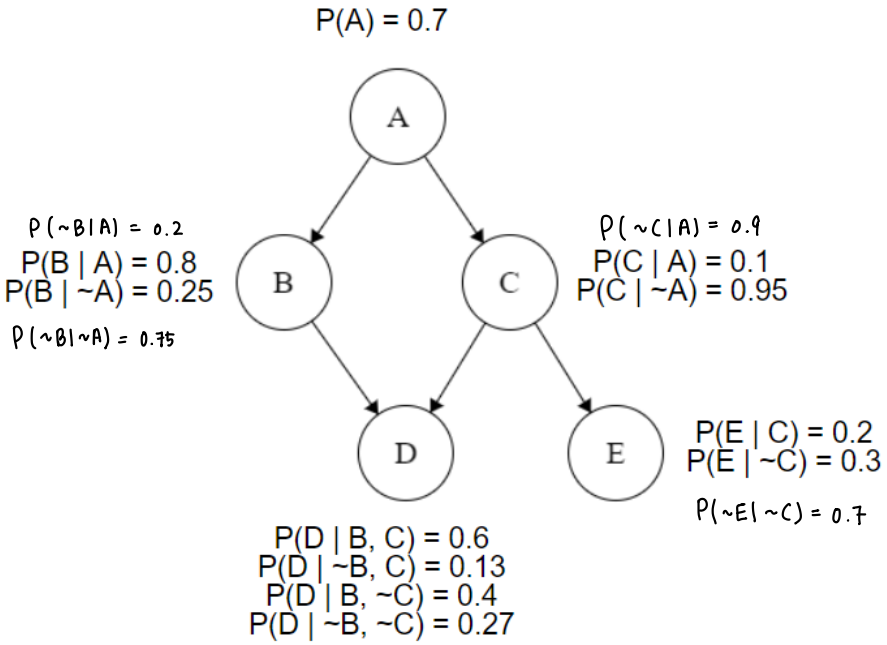


Figure 1: BN Inference

(a) (4 points) Compute $P(A, \neg C, D, \neg E)$. Show your work. 0.164934 →following page

(b) (5 points) Compute $P(A \mid D)$. Show your work. 0.7831 →following page

(c) (6 points) Compute $P(C \mid D)$. Show your work. 0.3061 →following page

$$P(A,B,C,D,E) = P(A)\cdot P(B|A)\cdot P(C|A)\cdot P(D|B,C)\cdot P(E|C)$$

(a) $P(A, \sim C, D, \sim E) = \bigoplus \begin{cases} B=T : P(A)\cdot P(B|A)\cdot P(\sim C|A)\cdot P(D|B,\sim C)\cdot P(\sim E|\sim C) \\ B=F : P(A)\cdot P(\sim B|A)\cdot P(\sim C|A)\cdot P(D|\sim B,\sim C)\cdot P(\sim E|\sim C) \end{cases}$

$= P(A)\cdot P(\sim C|A)\cdot P(\sim E|\sim C)\cdot \left[P(B|A)\cdot P(D|B,\sim C) + P(\sim B|A)\cdot P(D|\sim B,\sim C)\right]$

$= 0.7\cdot 0.9\cdot 0.7\cdot (0.8\cdot 0.4 + 0.2\cdot 0.27) = 0.1644934$

(b) $P(A|D) = \dfrac{P(A,D)}{P(D)}$

1) $P(A,D) = P(A)P(D|A)$

→ $P(A)\cdot P(B|A)\cdot P(C|A)\cdot P(D|B,C) = 0.7\times 0.8\times 0.1\times 0.6 = 0.0336$

$P(A)\cdot P(\sim B|A)\cdot P(C|A)\cdot P(D|\sim B,C) = 0.7\times 0.2\times 0.1\times 0.13 = 0.00182$

$P(A)\cdot P(B|A)\cdot P(\sim C|A)\cdot P(D|B,\sim C) = 0.7\times 0.8\times 0.9\times 0.4 = 0.2016$

$P(A)\cdot P(\sim B|A)\cdot P(\sim C|A)\cdot P(D|\sim B,\sim C) = 0.7\times 0.2\times 0.9\times 0.27 = 0.03402$

$\bigoplus = 0.27104$

2) $P(D) = P(A)\cdot P(D|A) + P(\sim A)\cdot P(D|\sim A)$

→ $P(\sim A)\cdot P(B|\sim A)\cdot P(C|\sim A)\cdot P(D|B,C) = 0.3\times 0.25\times 0.95\times 0.6 = 0.04275$

$P(\sim A)\cdot P(\sim B|\sim A)\cdot P(C|\sim A)\cdot P(D|\sim B,C) = 0.3\times 0.75\times 0.95\times 0.13 = 0.0277875$

$P(\sim A)\cdot P(B|\sim A)\cdot P(\sim C|\sim A)\cdot P(D|B,\sim C) = 0.3\times 0.25\times 0.05\times 0.4 = 0.0015$

$P(\sim A)\cdot P(\sim B|\sim A)\cdot P(\sim C|\sim A)\cdot P(D|\sim B,\sim C) = 0.3\times 0.75\times 0.05\times 0.27 = 0.0030375$

$\bigoplus = 0.075075$

$= 0.27104 + 0.075075 = 0.346115$

$\dfrac{P(A|D)}{P(D)} = \dfrac{0.27104}{0.346115} = 0.7831$

(c) $P(C|D) = \dfrac{P(C,D)}{P(D)}$

1) $P(C,D) = P(C)\cdot P(D|C)$

→ $P(A)\cdot P(B|A)\cdot P(C|A)\cdot P(D|B,C) = 0.7\times 0.8\times 0.1\times 0.6 = 0.0336$

$P(\sim A)\cdot P(B|\sim A)\cdot P(C|\sim A)\cdot P(D|B,C) = 0.3\times 0.25\times 0.95\times 0.6 = 0.04275$

$P(A)\cdot P(\sim B|A)\cdot P(C|A)\cdot P(D|\sim B,C) = 0.7\times 0.2\times 0.1\times 0.13 = 0.00182$

$P(\sim A)\cdot P(\sim B|\sim A)\cdot P(C|\sim A)\cdot P(D|\sim B,C) = 0.3\times 0.75\times 0.95\times 0.13 = 0.0277875$

$\bigoplus = 0.1059575$

$= \dfrac{0.1059575}{0.346115}$

$= 0.3061$

2. (20 points) [**D-Separation**] [**Graded By Ian Holmes**] Conditional independence is a key concept in Bayesian Belief Network (BBN). Please answer the following conditional independence and d-separation questions using the graphs below.
   (a) (5 points) In Figure 2, is A d-separated by $\emptyset$ from I? If so, how? Justify your answer. d - seperated
   (b) (5 points) In Figure 2, is A d-separated by $\{I, K\}$ from L? If so, how? Justify your answer. d-seperated
   (c) (5 points) In Figure 2, is B d-separated by $\{C, E\}$ from K? If so, how? Justify your answer. Not blocking
   (d) (5 points) In Figure 2, is $\{A, B, D\}$ d-separated by $\{H\}$ from $\{F, L, M\}$? If so, how? Justify your answer. Not blocking

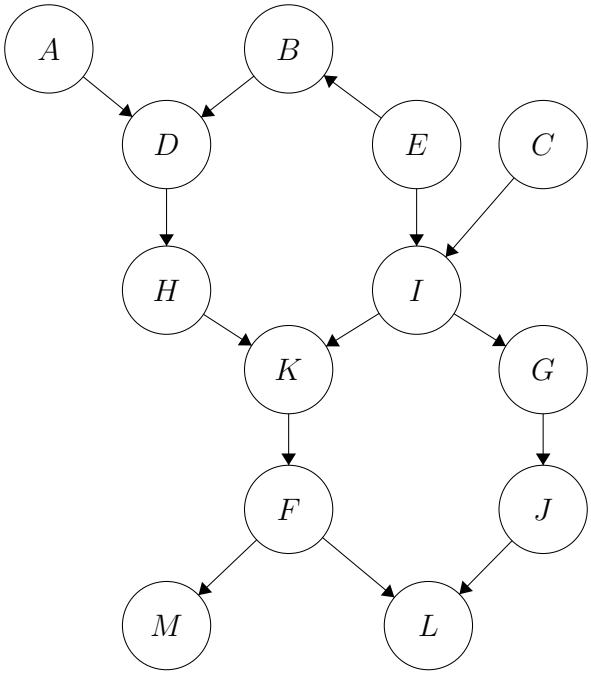


Figure 2: Bayesian Belief Network

(a) PATH1 : A → D → H → K ← I : blocked by K
PATH2 : A → D ← B ← E → I : blocked by D
Serial, Converging → no evidence (∅), Converging, Diverging, → no evidence (∅)
→ d-seperated

(b) A and I ← {I, K}
PATH1 : A → D → H → K → F → L : blocked by K
PATH2 : A → D ← B ← E → I → G → J → L : blocked by I
serial – condition on K, serial – condition on I
d-seperated

(c) B and K ← {C, E}
PATH1 : B → D → H → K : Serial PATH, No evidence → Not blocking
PATH2 : B ← E → I → K : blocked by E
diverging → condition on E

(d) {A, B, D} and {F, L, M} ← H
PATH1 : D → H → K → F : blocked by H
serial → condition on H
PATH2 : D ← B ← E → I → K → F : Not blocking
Converging, diverging → no evidence → not blocking
↳ condition on H
↳ Not blocking

3. (20 points) [**Linear Regression**] [**Graded By Tural Mehtiyev**]
   (a) (5 points) Given the following three data points of (x_1, x_2, y): $(-1, 2, 1)$, $(0, 1, -2)$, $(1, 2, 4)$, try to use a linear regression $y = \beta_1 x_1^2 + \beta_2 x_1 x_2 + \beta_0$ to predict y. Determine the values of β_1, β_2 and β_0 by **manual calculation** and show each step of your work.
   (b) (15 points) [Programming Task] Apply the following three linear regressions:
   (1) $y = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_0$
   (2) $y = \beta_1 x_1^2 + \beta_2 x_2^2 + \beta_3 x_3^2 + \beta_0$
   (3) $y = \gamma_1 x_1^3 + \gamma_2 x_2^3 + \gamma_3 x_3^3 + \gamma_0$

   to the provided data file "real_estate.csv".

   The objective of the linear regression model is to predict the house price of unit area. Write code in Python to perform following tasks. Please *report your output and relevant code* in the document file and also include your code (ends with .py) in the .zip file.

   You are allowed to use `numpy`, `pandas`, `sklearn`'s LOOCV (Leave One Out Cross Validation), but you are expected to write the linear regression algorithm.

   i. (10 points) Load the data. Fit the whole dataset to the three linear regression models, respectively. *Report* the coefficients (α_s, β_s, γ_s) of the three models.
   ii. (5 points) Use **leave-one-out** cross validation to determine the RMSE (root mean square error) for the three models. Specifically, in each fold, fit the training data to the model to determine the coefficients, then apply the coefficients to get predicted label for testing data (You don't need to report the coefficients in each fold). *Report* RMSE for the three models. Based on the RMSE, which model is the best for fitting the given data?

You are NOT allowed to use: sklearn.linear_model.LinearRegression, numpy.linalg.lstsq, or any other functions that are similar in nature.

(a)
$$\begin{cases} 1 = \beta_1(-1)^2 + \beta_2(-1)(2) + \beta_0 \\ -2 = \beta_1(0)^2 + \beta_2(0)(1) + \beta_0 \\ 4 = \beta_1(1)^2 + \beta_2(1)(2) + \beta_0 \end{cases} \leftrightarrow \begin{cases} 1 = \beta_1 - 2\beta_2 + \beta_0 \\ -2 = \beta_0 \\ 4 = \beta_1 + 2\beta_2 + \beta_0 \end{cases} \leftrightarrow \begin{cases} 1 = \beta_1 - 2\beta_2 - 2 \\ 4 = \beta_1 + 2\beta_2 - 2 \end{cases} \leftrightarrow \begin{cases} 3 = \beta_1 - 2\beta_2 \\ 6 = \beta_1 + 2\beta_2 \end{cases}$$

$-3 = -2\beta_2 - 2\beta_2$

$4\beta_2 = 3$

$\beta_2 = 3/4$

$3 = \beta_1 - \frac{3}{2}$

$\beta_1 = 3 + \frac{3}{2} = \frac{9}{2}$

$\beta_0 = -2$, $\beta_1 = 9/2$, $\beta_2 = 3/4$

(b) i $\alpha_1 = -0.233104$, $\alpha_2 = -0.005512$, $\alpha_3 = 1.298058$, $\alpha_0 = 42.748076$

$\beta_1 = -0.003425$, $\beta_2 = -0.000001$, $\beta_3 = 0.174033$, $\beta_0 = 37.498945$

$\gamma_1 = -0.000029$, $\gamma_2 = -0.000000$, $\gamma_3 = 0.017768$, $\gamma_0 = 36.714452$

ii. Model1 RMSE = 10.089057
Model2 RMSE = 11.176234
Model3 RMSE = 12.046067
Best Model based on LOOCV RMSE is Model1.

```
def build_model1(x1, x2, x3, verbose=False):
    if verbose:
        print("\n[Model 1]")
        print("y = alpha1*x1 + alpha2*x2 + alpha3*x3 + alpha0")
    return np.column_stack([x1, x2, x3, np.ones_like(x1)])

def build_model2(x1, x2, x3, verbose=False):
    if verbose:
        print("\n[Model 2]")
        print("y = beta1*(x1^2) + beta2*(x2^2) + beta3*(x3^2) + beta0")
    return np.column_stack([x1**2, x2**2, x3**2, np.ones_like(x1)])

def build_model3(x1, x2, x3, verbose=False):
    if verbose:
        print("\n[Model 3]")
        print("y = gamma1*(x1^3) + gamma2*(x2^3) + gamma3*(x3^3) + gamma0")
    return np.column_stack([x1**3, x2**3, x3**3, np.ones_like(x1)])
```

```
def OrdinaryLeastSquares_fit(X, y, ridge_eps=0.0):
    XtX = X.T @ X
    if ridge_eps > 0:
        XtX = XtX + ridge_eps * np.eye(XtX.shape[0])
    Xty = X.T @ y
    w = np.linalg.solve(XtX, Xty)
    return w

def rmse(y_true, y_pred):
    return float(np.sqrt(np.mean((y_true - y_pred)**2)))
```

i. Load the data

```
def load_data(csv_path, x_cols, y_col):
    df = pd.read_csv(csv_path)
    df.columns = df.columns.str.strip()
    missing = [c for c in [*x_cols, y_col] if c not in df.columns]
    if missing:
        raise ValueError(f"Columns not found: {missing}")
    df = df[[*x_cols, y_col]].dropna().copy()
    x1, x2, x3 = df[x_cols[0]].to_numpy(), df[x_cols[1]].to_numpy(), df[x_cols[2]].to_numpy()
    y = df[y_col].to_numpy()
    return x1, x2, x3, y
```

i. Fit the dataset

```
def fit_full_dataset(x1, x2, x3, y, x_cols):
    models = [
        ("Model 1", build_model1, "alpha", "y = alpha1*x1 + alpha2*x2 + alpha3*x3 + alpha0"),
        ("Model 2", build_model2, "beta", "y = beta1*(x1^2) + beta2*(x2^2) + beta3*(x3^2) + beta0"),
        ("Model 3", build_model3, "gamma", "y = gamma*(x1^3) + gamma2*(x2^3) + gamma3*(x3^3) + gamma0")
    ]
    results = []
    for name, builder, sym, form in models:
        X = builder(x1, x2, x3)
        w = OrdinaryLeastSquares_fit(X, y)
        y_pred = X @ w
        train_rmse = rmse(y, y_pred)

        print(f"\n{name} {form}")
        for label, val in zip(
            [f"{sym}1 ({x_cols[0]})", f"{sym}2 ({x_cols[1]})", f"{sym}3 ({x_cols[2]})", f"{sym}0 (bias)"],
            w.flatten()
        ):
            print(f" {label}: {float(val):.6f}")
        print(f"Training RMSE: {train_rmse:.6f}")
        results.append((name, builder, X, w))
    return results
```

ii. LOOCV

```
def loocv_evaluate(x1, x2, x3, y, results):
    rmse_scores = []
    for name, builder, _, _ in results:
        X = builder(x1, x2, x3)
        loo = LeaveOneOut()
        preds = np.zeros_like(y, dtype=float)
        for train_idx, test_idx in loo.split(X):
            X_train, y_train = X[train_idx], y[train_idx]
            X_test = X[test_idx]
            w = OrdinaryLeastSquares_fit(X_train, y_train)
            preds[test_idx] = (X_test @ w).ravel()
        score = rmse(y, preds)
        rmse_scores.append((name, score))
        print(f"LOOCV RMSE ({name}): {score:.6f}")
    best = min(rmse_scores, key=lambda t: t[1])
    print(f"\nBest Model based on LOOCV RMSE: {best[0]}")
    return rmse_scores
```

Run and Print Reports

```python
def main():
    csv_path = "HW4_data/real_estate.csv"
    x_cols = ["X1 house age", "X2 distance to the nearest MRT station", "X3 number of convenience stores"]
    y_col = "Y house price of unit area"

    x1, x2, x3, y = load_data(csv_path, x_cols, y_col)
    print("Dataset Info")
    print(f"X columns: {x_cols}")
    print(f"y column: {y_col}")
    print(f"Samples: {len(y)}")

    results = fit_full_dataset(x1, x2, x3, y, x_cols)

    loocv_evaluate(x1, x2, x3, y, results)

if __name__ == "__main__":
    main()
```

```
Dataset Info
X columns: ['X1 house age', 'X2 distance to the nearest MRT station', 'X3 number of convenience stores']
y column: Y house price of unit area
Samples: 414

Model 1 y = alpha1*x1 + alpha2*x2 + alpha3*x3 + alpha0
 alpha1 (X1 house age): -0.233104
 alpha2 (X2 distance to the nearest MRT station): -0.005512
 alpha3 (X3 number of convenience stores): 1.298058
 alpha0 (bias): 42.748076
Training RMSE: 10.087057

Model 2 y = beta1*(x1^2) + beta2*(x2^2) + beta3*(x3^2) + beta0
 beta1 (X1 house age): -0.003425
 beta2 (X2 distance to the nearest MRT station): -0.000001
 beta3 (X3 number of convenience stores): 0.179033
 beta0 (bias): 37.498945
Training RMSE: 11.176234

Model 3 y = gamma*(x1^3) + gamma2*(x2^3) + gamma3*(x3^3) + gamma0
 gamma1 (X1 house age): -0.000029
 gamma2 (X2 distance to the nearest MRT station): -0.000000
 gamma3 (X3 number of convenience stores): 0.017768
 gamma0 (bias): 36.714452
Training RMSE: 12.026067
...
LOOCV RMSE (Model 2): 11.305209
LOOCV RMSE (Model 3): 12.190789

Best Model based on LOOCV RMSE: Model 1
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

4. (25 points) [**Coding - ANN**] [**Graded by Ian Holmes**]
Follow the instructions under "GradeScope and GitHub Submission Instructions for Coding Questions" (see page 2).
Train, validate, and test a neural network model using the dataset in "ann_2025.zip", which contains training data, validation data, and test data. The goal here is to train the artificial neural network for a binary classification task. Use the Python neural networks package, Keras (i.e. `import tensorflow.keras`), for this problem. (All the output should be included in your report. Otherwise, your points will be deducted.)
   (a) (3 points) Please briefly describe how to construct your working environments (e.g. language, package version, backend for neural networks, installation, etc.) in your report, and write how to execute your code in a 'README.md' file.
   (b) (2 points) Please apply a fixed random seed of 2025 in order to generate the same result every time. To do this, you will need to override the 'PYTHONHASHSEED' environment variable accessible via the `import os`. Then, write the following: `os.environ['PYTHONHASHSEED']='2025'`. You should also override the random seed generators found in the 'random', 'numpy', and 'tensorflow' libraries with the value of 2025. Finally, set `os.environ['TF_DETERMINISTIC_OPS'] = '1'`.
   (c) (8 points) Create a body of code that iterates over a range of possible numbers of hidden neurons $X = (4, 16, 32, 64, 128)$ that will be used to define a neural network. For each number of hidden neurons, $x \in X$, do the following:
      i. Define a neural network with its parameters as follows: activation function for hidden layer = 'relu', activation for output layer = 'softmax', loss function = 'sparse_categorical_crossentropy', optimizer = 'adam', metrics = 'accuracy', epochs=5, batch_size=10. The model should have a single hidden layer, with the current number of hidden neurons in the loop, x, being assigned to define the size of the single hidden layer.
      ii. Fit the neural network with the provided training data (this is where you will need the epochs and batch_size parameters provided in the previous step). Set `shuffle=False` in your `.fit()` call.
      iii. Validate the neural network that has a hidden layer of size x using the given validation data. The validation accuracy is used to determine how many numbers of hidden neurons are optimal for this problem. You will want to save this value for later use.
      iv. Provide the essential code for the "neural network learning" and include descriptive comments in your report.
   (d) (5 points) Plot a figure, where the horizontal x-axis is the number of hidden neurons, and the vertical y-axis is the accuracy. Please plot both training and validation accuracy in your figure. (Note that the exact accuracy could be slightly different according to your working environments, however, you can analyze the trend.)

(e) (5 points) Provide a simple analysis of your results and choose the optimal number of hidden neurons from the analysis.

(f) (2 points) Report the test accuracy using the given test data on the neural network with the optimal number of hidden neurons.

Accuracy (hidden = 16) : 0.820 / loss = 0.391

(e) Optimal number of hidden neurons : 16

As shown in the figure and table, increasing the number of hidden neurons from 4 to 16 led to a clear improvement in both training and validation accuracy (Train: 0.847 → 0.883 / Val: 0.796 → 0.844). However, when the number of hidden neurons exceeded 16, the validation accuracy slightly decreased (e.g., 0.820 → 0.840) even though the training accuracy continued to increase. This trend indicates that the model began to overfit the training data, learning patterns that do not generalize well to unseen data. Therefore, the model with 16 hidden neurons provides the best balance between accuracy and generalization, achieving the highest validation accuracy (0.844).

(a)

```
path = "README.md"
with open(path, "w", encoding="utf-8") as f:
    f.write("\n".join(lines))

print("CWD:", os.getcwd())
print("Wrote:", os.path.abspath(path))
print("Bytes:", os.path.getsize(path))
print("Preview")
with open (path, "r", encoding="utf-8") as f:
    print("\n".join(f.read().splitlines()[:30]))
print("----------")
```

```
CWD: /Users/kim-yujin/Desktop/ncsu-engr-ALDA-Fall2025-H24_HW1/HW4
Wrote: /Users/kim-yujin/Desktop/ncsu-engr-ALDA-Fall2025-H24_HW1/HW4/README.md
Bytes: 736
Preview
# ANN Binary Classification

## (a) Environment Setup
- **Language:** Python 3.12.7
- **Libraries:**
 - TensorFlow 2.19.0
 - NumPy 2.3.2
 -Pandas 2.3.2
 - Matplotlib 3.10.6
- **Backend:** TensorFlow 2.x (CPU or GPU)
```

(b)

```
# Python has seed
os.environ['PYTHONHASHSEED'] = '2025'

# TensorFlow deterministic operation
os.environ['TF_DETERMINISTIC_OPS'] = '1'

# Define seed value
SEED = 2025

# Fix seed
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)

# Enable determinism in TensorFlow
try:
    tf.config.experimental.enable_op_determinism()
except Exception:
    pass

print("All random seed fixed to", SEED)
```

(c)-i

```python
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Input

def build_model(n_features, hidden_units):
    """
    Build a single-hidden-layer neural network model.

    Parameters:
        n_features (int): Number of input features (columns in X).
        hidden_units (int): Number of neurons in the hidden layer.

    Returns:
        model (Sequential): Compiled Keras model ready for training.
    """
    model = Sequential([
        Input(shape=(n_features,)),             # input layer
        Dense(hidden_units, activation='relu'), # hidden layer
        Dense(2, activation='softmax')           # output layer
    ])

    # Compile the model
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

print("build_model() function defined successfully.")
```

(c)-ii – iv

(c)-ii

```python
# hyperparameters per the assignment
H_LIST = [4, 16, 32, 64, 128]
EPOCHS = 5
BATCH_SIZE = 10

# Containers for last epoch acc
train_accs = []
val_accs = []

# Keep best by validation acc
best_val = -np.inf
best_h = None
best_model = None

n_features = X_train.shape[1]

for h in H_LIST:
    model = build_model(n_features, hidden_units=h)
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=EPOCHS,
        batch_size=BATCH_SIZE,
        shuffle=False,
        verbose=0
    )
```

(c)-iii

```
    # (c)-iii Validation step: use last epoch acc
    train_acc = float(history.history['accuracy'][-1])
    val_acc   = float(history.history['val_accuracy'][-1])
    train_accs.append(train_acc)
    val_accs.append(val_acc)
```

```
    val_acc   = float(history.history['val_accuracy'][-1])
    train_accs.append(train_acc)
    val_accs.append(val_acc)

    # Track the best model
    if (val_acc > best_val) or (np.isclose(val_acc, best_val) and (best_h is None or h < best_h)):
        best_val = val_acc
        best_h = h
        best_model = model

    print(f"[Info] Hidden={h:>3} | Train acc={train_acc:.4f} | Val acc={val_acc:.4f}")

print("\nSummary (last epoch acc):")
for h, ta, va in zip(H_LIST, train_accs, val_accs):
    print(f"Hidden={h:>3} | Train={ta:.4f} | Val={va:.4f}")

print(f"\nSelected best hidden size by validation accuracy: {best_h} (val_acc={best_val:.4f})")
```

```
[Info] Hidden=  4 | Train acc=0.8470 | Val acc=0.7960
[Info] Hidden= 16 | Train acc=0.8830 | Val acc=0.8440
[Info] Hidden= 32 | Train acc=0.8950 | Val acc=0.8200
[Info] Hidden= 64 | Train acc=0.9070 | Val acc=0.8360
[Info] Hidden=128 | Train acc=0.9170 | Val acc=0.8400

Summary (last epoch acc):
Hidden=  4 | Train=0.8470 | Val=0.7960
Hidden= 16 | Train=0.8830 | Val=0.8440
Hidden= 32 | Train=0.8950 | Val=0.8200
Hidden= 64 | Train=0.9070 | Val=0.8360
Hidden=128 | Train=0.9170 | Val=0.8400

Selected best hidden size by validation accuracy: 16 (val_acc=0.8440)
```

(c) - iv.

The code below implements the neural network learning process. Each model has one hidden layer, where the number of neurons varies among 4, 16, 32, 64, and 128.
The model is trained for 5 epochs with a batch size of 10, using the same optimizer (Adam) and loss function (Sparse categorical cross-entropy).
Training and validation accuracies are recorded after each run, and the model with the highest validation accuracy is selected as the optimal network.

```
# hyperparameters per the assignment
H_LIST = [4, 16, 32, 64, 128]
EPOCHS = 5
BATCH_SIZE = 10

# Containers for last epoch acc
train_accs = []
val_accs = []

# Keep best by validation acc
best_val = -np.inf
best_h = None
best_model = None

n_features = X_train.shape[1]

for h in H_LIST:
    model = build_model(n_features, hidden_units=h)
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=EPOCHS,
        batch_size=BATCH_SIZE,
        shuffle=False,
        verbose=0
    )

    # (c)-iii Validation step: use last epoch acc
    train_acc = float(history.history['accuracy'][-1])
    val_acc   = float(history.history['val_accuracy'][-1])
    train_accs.append(train_acc)
    val_accs.append(val_acc)
```

```
    val_acc   = float(history.history['val_accuracy'][-1])
    train_accs.append(train_acc)
    val_accs.append(val_acc)

    # Track the best model
    if (val_acc > best_val) or (np.isclose(val_acc, best_val) and (best_h is None or h < best_h)):
        best_val = val_acc
        best_h = h
        best_model = model

    print(f"[Info] Hidden={h:>3} | Train acc={train_acc:.4f} | Val acc={val_acc:.4f}")
```

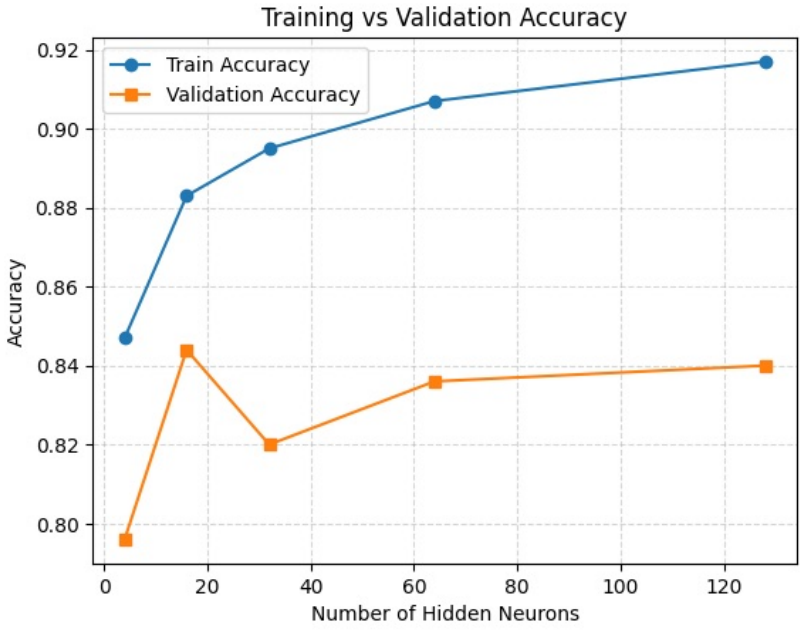
(d)

```
import matplotlib.pyplot as plt

os.makedirs("outputs", exist_ok=True)

plt.figure()
plt.plot(H_LIST, train_accs, marker='o', label='Train Accuracy')
plt.plot(H_LIST, val_accs, marker='s', label='Validation Accuracy')
plt.xlabel('Number of Hidden Neurons')
plt.ylabel('Accuracy')
plt.title('Training vs Validation Accuracy')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.5)
plt.savefig("outputs/accuracy_plot.png", dpi=150, bbox_inches='tight')
plt.show()

print("Saved figure to outputs/accuracy_plot.png")
```



(e)

```
df_results = pd.DataFrame({
    "hidden": H_LIST,
    "train_acc": [float(x) for x in train_accs],
    "val_acc":   [float(x) for x in val_accs],
})
df_results.to_csv("outputs/results.csv", index=False)
df_results
```

	hidden	train_acc	val_acc
0	4	0.847	0.796
1	16	0.883	0.844
2	32	0.895	0.820
3	64	0.907	0.836
4	128	0.917	0.840

(f)

```
assert best_model is not None and best_h is not None, "best_model/best_h not set. Re-run the traini

test_loss, test_acc = best_model.evaluate(X_test, y_test, verbose=0)
print(f"[Test] Accuracy (hidden={best_h}): {test_acc:.4f} | loss={test_loss:.4f}")
```

```
[Test] Accuracy (hidden=16): 0.8200 | loss=0.3911
2025-10-29 19:51:48.494836: E tensorflow/core/framework/node_def_util.cc:680] NodeDef mentions attribut
2025-10-29 19:51:48.495756: E tensorflow/core/framework/node_def_util.cc:680] NodeDef mentions attribut
```

5. (20 points) [**SVM Programming**] [**Graded By Tural Mehtiyev**] In this question, you will employ SVM to solve a classification problem for the provided "svm_2025.csv". This data consists of 60 features and a label for each row that could be 0 or 1. Write code in Python to perform the following tasks. Please *report your output and relevant code* in the document file and also include your code (ends with .py) in the .zip file.
   (a) (1 point) Load data. Report the size of the positive (class 1) and negative (class 0) samples in dataset.
   (b) (2 points) Use *stratified random sampling* to divide the dataset into training data (75%) and testing data (25%) [use random_state= 42]. *Report* the number of positive and negative samples in both training and testing data.
   (c) (7 points) Using the data given in "svm_2025.zip" in that "train_data_2025.csv" as training data, "test_data_2025.csv" as testing data, take SVM with linear kernel as classifier (*third-party packages are allowed to be used*) and set the regularization parameter C as: [0.1, 0.2, 0.3, 0.5, 1, 2, 3, 5, 10], respectively. For each value of C, train a SVM classifier with the training data and get the number of support vectors (SVs). Generate a plot with C as the horizontal axis and number of SVs as the vertical axis. Give a brief analysis on the plot by explaining: 1) as C increases, how the number of SVs changes, and 2) why.
   (d) (10 points) Using "train_data_2025.csv" as training and "test_data_2025.csv" as testing data, compare the performance of four different kernel functions: linear, polynomial, radial basic function (Gaussian kernel), and Sigmoid. For each type of kernel functions, train your SVM classifiers using the training data and evaluate the resulted SVM classifer using testing data by making a table to record *accuracy*, *precision*, *recall* and *f-measure* of the corresponding classification results.
   For different kernel functions, try to tune its parameters such that:
   - for the linear kernel, try to tune C.
   - for the polynomial kernel, try to tune C, degree, and coef0.
   - for the RBF kernel, try to tune C and γ.
   - for the Sigmoid kernel, try to tune C, coef0, and γ.

   More specifically, you should explore each of parameters within these ranges:
   $C \in$ [0.1, 0.2, 0.3, 1, 5, 10, 20, 100, 200, 1000]
   degree $\in$ [1, 2, 3, 4, 5].
   coef0 $\in$ [0.0001, 0.001, 0.002, 0.01, 0.02, 0.1, 0.2, 0.3, 1, 2, 5, 10]
   $\gamma \in$ [0.0001, 0.001, 0.002, 0.01, 0.02, 0.03, 0.1, 0.2, 1, 2, 3]
   You are strongly encourage to use `sklearn`'s GridSearchCV() function; if you decide to use this library feature, please set the cross validation parameter to be 5 (argument of "None" is 5-fold cross validation by default). For each of the four types of kernel functions, please report the best result (using F-measure) and its corresponding optimal parameters. For this dataset, which kernel function will you choose?

(a)

```
import pandas as pd

df = pd.read_csv("HW4_data/SVM_2025/svm_2025.csv")

display(df.head())
print(df.shape)

print(df["Class"].value_counts())
```

✓ 0.0s

```
(156, 61)
Class
0    83
1    73
Name: count, dtype: int64
```

(b)

```
from sklearn.model_selection import train_test_split

label_col = 'Class'

# Split X, y
X = df.drop(columns=[label_col])
y = df[label_col].astype(int)

# Stratified random sampling
X_traing, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=42
)

print("(b) Train/Test Class distribution: 1 positive sample, 0 negative sample")
print(" Train:", y_train.value_counts().sort_index().to_dict())
print(" Test :", y_test.value_counts().sort_index().to_dict())
```

✓ 0.0s

```
(b) Train/Test Class distribution: 1 positive sample, 0 negative sample
 Train: {0: 62, 1: 55}
 Test : {0: 21, 1: 18}
```

(c)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

# load train and test data
train_path = "HW4_data/SVM_2025/train_data_2025.csv"
test_path = "HW4_data/SVM_2025/test_data_2025.csv"

train_df = pd.read_csv(train_path)
test_df  = pd.read_csv(test_path)


def detect_label_column(df):
    candidates = ["label", "Label", "LABEL", "y", "Y", "target", "Target", "class", "Class", "CLASS"]
    cols_lower = {c.lower(): c for c in df.columns}
    for c in candidates:
        if c.lower() in cols_lower:
            return cols_lower[c.lower()]
    return df.columns[-1]

label_col = detect_label_column(train_df)
```

```
def to_binary(y):
    uniq = sorted(pd.unique(y))
    if set(uniq) in ({0,1}, {0.0,1.0}):
        return y.astype(int)
    if len(uniq) == 2:
        mapping = {uniq[0]: 0, uniq[1]: 1}
        return y.map(mapping).astype(int)
    yn = pd.to_numeric(y, errors="coerce")
    if yn.isna().any():
        raise ValueError("Cannot convert labels to {0,1}.")
    return (yn >= 0.5).astype(int)

X_train = train_df.drop(columns=[label_col])
y_train = to_binary(train_df[label_col])
X_test = test_df.drop(columns=[label_col])
y_test = to_binary(test_df[label_col])
```

✓ 0.0s

```
# Train a linear SVM for each value of C
C_values = [0.1, 0.2, 0.3, 0.5, 1, 2, 3, 5, 10]
sv_counts=[]

for C in C_values:
    clf = SVC(kernel="linear", C=C)
    clf.fit(X_train, y_train)
    sv_counts.append(int(clf.support_.shape[0]))
```

✓ 0.0s

```
sv_table = pd.DataFrame({"C": C_values, "num_support_vectors": sv_counts})
display(sv_table)
```

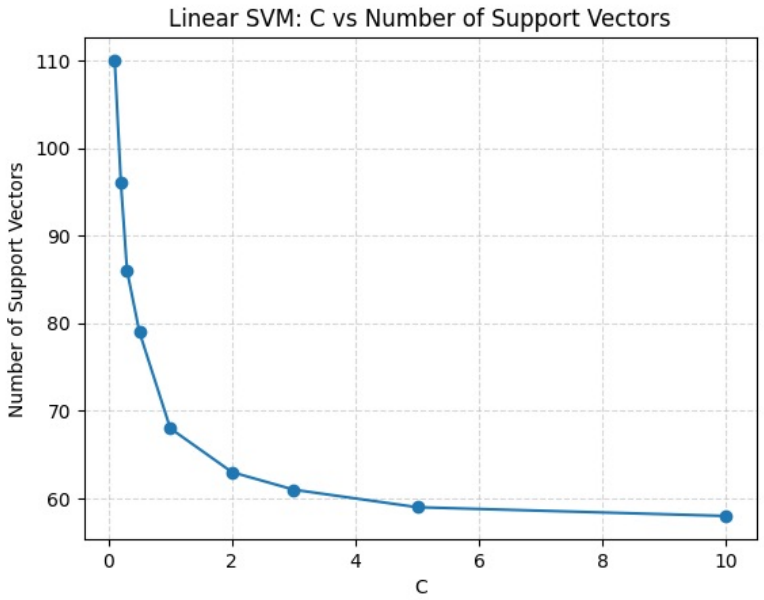
✓ 0.0s

	C	num_support_vectors
0	0.1	110
1	0.2	96
2	0.3	86
3	0.5	79
4	1.0	68
5	2.0	63
6	3.0	61
7	5.0	59
8	10.0	58

```
plt.figure()
plt.plot(C_values, sv_counts, marker='o')
plt.xlabel("C")
plt.ylabel("Number of Support Vectors")
plt.title("Linear SVM: C vs Number of Support Vectors")
plt.grid(True, linestyle='--', alpha=0.5)
plt.savefig("svm_support_vectors_plot.png", bbox_inches="tight", dpi=160)
plt.show()

print("save graph file: svm_support_vectors_plot.png")
```

✓ 0.0s



As the regularization parameter C increases, the number of support vectors decreases. This happens because a large C places a stronger penalty on misclassified points, forcing the SVM to creat a decision boundary that fits the training data more strictly, Consequently, fewer data point lie withing or violate the margin, resulting in a smaller set of support vectors. In contrast, when C is small, the model allows a wider margin with more tolerance for misclassification, so more data points become support vectors.

(d)

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def evaluate(model, X_test, y_test):
    y_pred = model.predict(X_test)
    return {
        "accuracy": accuracy_score(y_test, y_pred),
        "precision": precision_score(y_test, y_pred, zero_division=0),
        "recall": recall_score(y_test, y_pred, zero_division=0),
        "f1": f1_score(y_test, y_pred, zero_division=0)
    }
```

✓ 0.0s

```
# range of hyperprameter exploration
C_list = [0.1, 0.2, 0.3, 1, 5, 10, 20, 100, 200, 1000]
degree_ls = [1, 2, 3, 4, 5]
coef0_ls = [0.0001, 0.001, 0.002, 0.01, 0.02, 0.1, 0.2, 0.3, 1, 2, 5, 10]
gamma_ls = [0.0001, 0.001, 0.002, 0.01, 0.02, 0.03, 0.1, 0.2, 1, 2, 3]

results = []
best_params_report = []
```

✓ 0.0s

```python
# Kernal functions: Linear, Polynomial, RBF, Sigmoid
gs_linear = GridSearchCV(SVC(), {"kernel":["linear"], "C":C_list}, scoring="f1", cv=5, n_jobs=-1)
gs_linear.fit(X_train, y_train)
results.append({"kernel":"linear", **evaluate(gs_linear.best_estimator_, X_test, y_test)})
best_params_report.append(("linear", gs_linear.best_params_, gs_linear.best_score_))

gs_poly = GridSearchCV(SVC(), {"kernel":["poly"], "C":C_list, "degree":degree_ls, "coef0":coef0_ls}, scoring="f1",
gs_poly.fit(X_train, y_train)
results.append({"kernel":"poly", **evaluate(gs_poly.best_estimator_, X_test, y_test)})
best_params_report.append(("poly", gs_poly.best_params_, gs_poly.best_score_))

gs_rbf = GridSearchCV(SVC(), {"kernel":["rbf"], "C":C_list, "gamma":gamma_ls}, scoring="f1", cv=5, n_jobs=-1)
gs_rbf.fit(X_train, y_train)
results.append({"kernel":"rbf", **evaluate(gs_rbf.best_estimator_, X_test, y_test)})
best_params_report.append(("rbf", gs_rbf.best_params_, gs_rbf.best_score_))

gs_sig = GridSearchCV(SVC(), {"kernel":["sigmoid"], "C":C_list, "coef0":coef0_ls, "gamma":gamma_ls}, scoring="f1",
gs_sig.fit(X_train, y_train)
results.append({"kernel":"sigmoid", **evaluate(gs_sig.best_estimator_, X_test, y_test)})
best_params_report.append(("sigmoid", gs_sig.best_params_, gs_sig.best_score_))


print("\nBest params per kernel (CV=5, scoring=F1):")
for name, params, cvf1 in best_params_report:
    print(f"  - {name:7s} | best_pramas={params} | cv_best_f1={cvf1:.4f}")
```

✓ 3.9s

```
Best params per kernel (CV=5, scoring=F1):
  - linear  | best_pramas={'C': 10, 'kernel': 'linear'} | cv_best_f1=0.8449
  - poly    | best_pramas={'C': 0.1, 'coef0': 0.3, 'degree': 2, 'kernel': 'poly'} | cv_best_f1=0.8530
  - rbf     | best_pramas={'C': 0.2, 'gamma': 0.2, 'kernel': 'rbf'} | cv_best_f1=0.8600
  - sigmoid | best_pramas={'C': 200, 'coef0': 1, 'gamma': 0.03, 'kernel': 'sigmoid'} | cv_best_f1=0.8601
```

Both RBF and Sigmoid kernels achieved the highest F1-score (0.86) during cross-validation, slightly outperforming the linear and Polynomial kernels. This indicates that nonlinear decision boundaries fit the data better than a purely linear model. The RBG kernel, with moderate values of C (0.2) and gamma (0.2), effectively balances bias and variance, capturing complex relationships without overfitting. The Sigmoid kernel performed comparably, but RBF is generally preferred because it is more stable and less sensitive to parameter scaling.

6. (30 points) [**SVM Theory**] [**Graded By Rajesh Debnath**] Given 2-dimensional data points $\langle \mathbf{X}^i, Y \rangle$, $i \in [1, 2, 3, 4]$ as shown in Table 1. In this question, you will employ the kernel function for SVM trained with these four data points. Let α_1; α_2; α_3, and α_4 be the Lagrangian multipliers associated with them as: α_i is associated with $\langle \mathbf{X^i}; y_i \rangle$.

Data ID	x_1	x_2	y
$\mathbf{X}^1$	1	1	-1
$\mathbf{X}^2$	-1	1	-1
$\mathbf{X}^3$	1	-1	1
$\mathbf{X}^4$	-1	-1	1

Table 1: Q5(b)

(a) (4 points) Suppose the kernel function is: $\mathbf{K}(\mathbf{X}^i, \mathbf{X}^j) = 2 + ((\mathbf{X}^i \cdot \mathbf{X}^j) + 4 * (\mathbf{X}^i \cdot \mathbf{X}^j)^2)$, where $\mathbf{X}^i$ and $\mathbf{X}^j$ indicate two data points. This kernel is equal to an inner product $\phi(\mathbf{X}^i) \cdot \phi(\mathbf{X}^j)$ with a certain function of ϕ. What is the function of ϕ?

(b) (3 points) Transform the four given data points $\mathbf{X}^i, i \in [1, 2, 3, 4]$ to the higher dimensional space via the function ϕ get from (i). Report your results.

→ kernel Trick

(c) (10 points) Assume the four transformed data points get from (ii) are all support vectors. Apply Lagrange multipliers to determine the *maximum margin linear decision boundary* in the transformed higher dimensional space

(d) (3 points) to classify the point (2,2).

(a) $K(x_i, x_j) = 2 + ((x_i, x_j) + 4 x (x_i, x_j)^2)$

$x_i \cdot x_j = x_{i1} x_{j1} + x_{i2} x_{j2}$

$(x_i \cdot x_j)^2 = (x_{i1} \cdot x_{j1} + x_{i2} x_{j2})^2 = x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2$

$K(x_i, x_j) = 2 + (x_{i1} x_{j1} + x_{i2} x_{j2}) + 4(x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2)$

$\phi(x) = [\sqrt{2}, x_1, x_2, 2x_1^2, 2\sqrt{2} x_1 x_2, 2x_2^2]^T$

(b) $\max_\alpha \; \Sigma_i \alpha_i - \frac{1}{2} \Sigma_{ij} \alpha_i \alpha_j y_i y_j x_i x_j$ $\quad K(x_i, x_j) = 2 + ((x_i, x_j) + 4 x (x_i, x_j)^2)$

$x_i x_j$	x_1	x_2	x_3	x_4
x_1	2	0	0	-2
x_2	0	2	-2	0
x_3	0	-2	2	0
x_4	-2	0	0	2

$k(x_i x_j)$	x_1	x_2	x_3	x_4
x_1	20	2	2	16
x_2	2	20	16	2
x_3	2	16	20	2
x_4	16	2	2	20

$\Rightarrow K = \begin{bmatrix} 20 & 2 & 2 & 16 \\ 2 & 20 & 16 & 2 \\ 2 & 16 & 20 & 2 \\ 16 & 2 & 2 & 20 \end{bmatrix}$

(c) $\max_{\alpha} \sum_i \alpha_i - \frac{1}{2}\sum_{ij} \alpha_i\alpha_j y_i y_j x_i x_j$

$= \sum \alpha_i - \frac{1}{2}\sum \alpha_i \alpha_j y_i y_j k(x_i, x_j)$

$$\begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 20 & 2 & 2 & 16 \\ 2 & 20 & 16 & 2 \\ 2 & 16 & 20 & 2 \\ 16 & 2 & 2 & 20 \end{bmatrix}$$

$$= \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2}\Big(\alpha_1^2 \cdot 1 \cdot 20 + \alpha_1\alpha_2 \cdot 1 \cdot 2\times 2 + \alpha_1\alpha_3 \cdot (-1)\cdot 2\times 2 + \alpha_1\alpha_4 \cdot (-1)\cdot 16 \times 2$$
$$+ \alpha_2^2 \cdot 1 \cdot 20 + \alpha_2\alpha_3 \cdot (-1)\cdot 16 \times 2 + \alpha_2\alpha_4 \cdot (-1)\cdot 2\times 2$$
$$+ \alpha_3^2 \cdot 1 \cdot 20 + \alpha_3\alpha_4 \cdot 1 \cdot 2\times 2 + \alpha_4^2 \cdot 1 \cdot 20 \Big)$$

$$= \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2}\Big(20\alpha_1^2 + 4\alpha_1\alpha_2 - 4\alpha_1\alpha_3 - 32\alpha_1\alpha_4$$
$$+ 20\alpha_2^2 - 32\alpha_2\alpha_3 - 4\alpha_2\alpha_4 + 20\alpha_3^2 + 4\alpha_3\alpha_4 + 20\alpha_4^2 \Big)$$

$$\frac{\partial L}{\partial \alpha_1} = 1 - \frac{1}{2}(2\times 20\alpha_1 + 4\alpha_2 - 4\alpha_3 - 32\alpha_4) = 0 \quad \leftrightarrow 1 - 20\alpha_1 - 2\alpha_2 + 2\alpha_3 + 16\alpha_4 = 0 \quad (1)$$

$$\frac{\partial L}{\partial \alpha_2} = 1 - \frac{1}{2}(4\alpha_1 + 2\times 20\alpha_2 - 32\alpha_3 - 4\alpha_4) = 0 \quad \leftrightarrow 1 - 2\alpha_1 - 20\alpha_2 + 16\alpha_3 + 2\alpha_4 = 0 \quad (2)$$

$$\frac{\partial L}{\partial \alpha_3} = 1 - \frac{1}{2}(-4\alpha_1 - 32\alpha_2 + 2\times 20\alpha_3 + 4\alpha_4) = 0 \quad \leftrightarrow 1 + 2\alpha_1 + 16\alpha_2 - 20\alpha_3 - 2\alpha_4 = 0 \quad (3)$$

$$\frac{\partial L}{\partial \alpha_4} = 1 - \frac{1}{2}(-32\alpha_1 - 4\alpha_2 + 4\alpha_3 + 2\times 20\alpha_4) = 0 \quad \leftrightarrow 1 + 16\alpha_1 + 2\alpha_2 - 2\alpha_3 - 20\alpha_4 = 0 \quad (4)$$

$$20\alpha_1 + 2\alpha_2 - 2\alpha_3 - 16\alpha_4 = 1 \quad (1)$$
$$2\alpha_1 + 20\alpha_2 - 16\alpha_3 - 2\alpha_4 = 1 \quad (2)$$
$$-2\alpha_1 - 16\alpha_2 + 20\alpha_3 + 2\alpha_4 = 1 \quad (3)$$
$$-16\alpha_1 - 2\alpha_2 + 2\alpha_3 + 20\alpha_4 = 1 \quad (4)$$

(1) − (2)

$$-\begin{array}{l} 20\alpha_1 + 2\alpha_2 - 2\alpha_3 - 16\alpha_4 = 1 \\ 2\alpha_1 + 20\alpha_2 - 16\alpha_3 - 2\alpha_4 = 1 \\ \hline 18\alpha_1 - 18\alpha_2 + 14\alpha_3 - 14\alpha_4 = 0 \end{array}$$

$$18(\alpha_1 - \alpha_2) + 14(\alpha_3 - \alpha_4) = 0$$

(3) − (4)

$$-\begin{array}{l} -2\alpha_1 - 16\alpha_2 + 20\alpha_3 + 2\alpha_4 = 1 \\ -16\alpha_1 - 2\alpha_2 + 2\alpha_3 + 20\alpha_4 = 1 \\ \hline 14\alpha_1 - 14\alpha_2 + 18\alpha_3 - 18\alpha_4 = 0 \end{array}$$

$$14(\alpha_1 - \alpha_2) + 18(\alpha_3 - \alpha_4) = 0$$

$\ominus$

$$4(\alpha_1 - \alpha_2) - 4(\alpha_3 - \alpha_4) = 0$$
$$\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4 = 0$$

i) $\alpha_1 = \alpha_2$, $\alpha_3 = \alpha_4$

ii) $\alpha_1 - \alpha_2 = \alpha_3 - \alpha_4$

$\Rightarrow \alpha_1 = \alpha_2 = \alpha_3 = \alpha_4$

(1) $\rightarrow$ $20\alpha + 2\alpha - 2\alpha - 16\alpha = 1 \leftrightarrow 4\alpha = 1$ $\therefore \alpha = \frac{1}{4}$

$$\frac{\partial L}{\partial w_0} \Rightarrow \Sigma \alpha_i y_i = 0$$

$w \cdot \phi(x_1) + b = y_1$

$w = \sum_i \alpha_i y_i \phi(x_i)$

$w = \alpha_1 y_1 \phi(x_1) + \alpha_2 y_2 \phi(x_2) + \alpha_3 y_3 \phi(x_3) + \alpha_4 y_4 \phi(x_4)$

$= \frac{1}{4} \cdot (-1) \cdot \phi(x_1) + \frac{1}{4}(-1)\phi(x_2) + \frac{1}{4}\phi(x_3) + \frac{1}{4}\phi(x_4)$

$$K = \begin{bmatrix} 20 & 2 & 2 & 16 \\ 2 & 20 & 16 & 2 \\ 2 & 16 & 20 & 2 \\ 16 & 2 & 2 & 20 \end{bmatrix}$$

$-\frac{1}{4}k(x_1,x_1) - \frac{1}{4}k(x_1,x_2) + \frac{1}{4}k(x_1,x_3) + \frac{1}{4}k(x_1,x_4)$

$= \frac{1}{4}(-20 - 2 + 2 + 16) + b = \frac{1}{4} \times (-4) + b = -1 + b = \underset{(-1)}{y}$

$-\frac{1}{4}k(x_2,x_1) - \frac{1}{4}k(x_2,x_2) + \frac{1}{4}k(x_2,x_3) + \frac{1}{4}k(x_2,x_4)$

$= \frac{1}{4}(-2 - 20 + 16 + 2) + b = \frac{-4}{4} + b = -1 + b = \underset{(-1)}{y}$

$-\frac{1}{4}k(x_3,x_1) - \frac{1}{4}k(x_3,x_2) + \frac{1}{4}k(x_3,x_3) + \frac{1}{4}k(x_3,x_4)$

$= \frac{1}{4}(-2 - 16 + 20 + 2) + b = \frac{4}{4} + b = 1 + b = \underset{(1)}{y}$

$-\frac{1}{4}k(x_4,x_1) - \frac{1}{4}k(x_4,x_2) + \frac{1}{4}k(x_4,x_3) + \frac{1}{4}k(x_4,x_4)$

$= \frac{1}{4}(-16 - 2 + 2 + 20) + b = \frac{4}{4} + b = 1 + b = \underset{(1)}{y}$

$\therefore b = 0$

Decision function

$f(x) = \sum_{i=1}^{4} \alpha_i y_i k(x_i, x) + b = \frac{1}{4}[-k(x_1,x) - k(x_2,x) + k(x_3,x) + k(x_4,x)]$

$\hat{y} = \text{sign } f(x)$

$\alpha_i = 1/4 > 0 \rightarrow x_1, x_2, x_3, x_4 \Rightarrow SV \quad , b = 0$

$r = \frac{1}{\|w\|} = 1$

(d) $x = (2, 2)$

$K(x_i, x_j) = 2 + ((x_i, x_j) + 4 \times (x_i, x_j)^2)$

$K(x_1, (2,2)) = 2 + (4 + 4 \times 4^2) = 6 + 64 = 70$

$K(x_2, (2,2)) = 2 + (0 + 4 \times 0^2) = 2$

$K(x_3, (2,2)) = 2 + (0 + 4 \times 0^2) = 2$

$K(x_4, (2,2)) = 2 + (-4 + 4 \times 4^2) = -2 + 64 = 62$

$W \cdot \phi(x) + 0 \quad > 0 \quad +1$

$\quad\quad\quad\quad\quad\quad < 0 \quad -1$

$W = \frac{1}{4} \cdot (-1) \cdot \phi(x_1) + \frac{1}{4}(-1) \cdot \phi(x_2) + \frac{1}{4}\phi(x_3) + \frac{1}{4}\phi(x_4)$

$\frac{1}{4} \cdot (-1) K(x_1, x) + \frac{1}{4}(-1) \cdot K(x_2, x) + \frac{1}{4} K(x_3, x) + \frac{1}{4} K(x_4, x)$

$= -\frac{1}{4} \times 70 - \cancel{\frac{1}{4}} \times 2 + \cancel{\frac{1}{4}} \times 2 + \frac{1}{4} \times 62$

$= -\frac{8}{4} = -2$

$f(2,2) = -2 < 0 \quad y = -1$

Class = -1

Data ID	x_1	x_2	y
$\mathbf{X}^1$	1	1	-1
$\mathbf{X}^2$	-1	1	-1
$\mathbf{X}^3$	1	-1	1
$\mathbf{X}^4$	-1	-1	1

Table 1: Q5(b)