MAIE 5421 Computer Vision: Assignment-1

September 26, 2025

Github URL: Rezela/Computer-Vision-Assignment-1

1 Simple Linear Regression (15 Points)

1. $SSR = \sum_{i=1}^{n} \left(y_i - (\beta + \alpha x_i) \right)^2$

① $$\frac{\partial SSR}{\partial \beta} = 2 \cdot \sum_{i=1}^{n} \left(y_i - (\beta + \alpha x_i) \right) \cdot (-1)$$

let $\frac{\partial SSR}{\partial \beta} = 0$,

$$n\beta = \sum_{i=1}^{n} (y_i - \alpha x_i)$$

$$\beta = \bar{y} - \alpha \bar{x}$$

② $$\frac{\partial SSR}{\partial \alpha} = 2 \cdot \sum_{i=1}^{n} \left(y_i - (\beta + \alpha x_i) \right) \cdot (-x_i)$$

introduce $\beta = \bar{y} - \alpha \bar{x}$,

$$\frac{\partial SSR}{\partial \alpha} = -2 \cdot \sum_{i=1}^{n} \left(x_i y_i - x_i \bar{y} + \alpha x_i \bar{x} - \alpha x_i^2 \right)$$

let $\frac{\partial SSR}{\partial \alpha} = 0$,

$$\alpha = \frac{\sum_{i=1}^{n} (x_i y_i - x_i \bar{y})}{\sum_{i=1}^{n} (x_i^2 - x_i \bar{x})}$$

Conclusion: $$\hat{\alpha} = \frac{\sum_{i=1}^{n} (x_i y_i - x_i \bar{y})}{\sum_{i=1}^{n} (x_i^2 - x_i \bar{x})}$$

$$\hat{\beta} = \bar{y} - \bar{x} \cdot \frac{\sum_{i=1}^{n} (x_i y_i - x_i \bar{y})}{\sum_{i=1}^{n} (x_i^2 - x_i \bar{x})}$$

2 Confusion Matrix (10 Points)

2.

		Actual Pos	Ne
Predict	Pos	(TP) 40	5 (FP)
	Ne	(FN) 10	45 (TN)

True Positive Rate (Recall): $TPR = \frac{TP}{TP+FN} = \frac{40}{40+10} = 0.8$

True Negative Rate: $TNR = \frac{TN}{TN+FP} = \frac{45}{45+5} = 0.9$

False Positive Rate: $FPR = \frac{FP}{FP+TN} = \frac{5}{5+45} = 0.1$

False Negative Rate: $FNR = \frac{FN}{FN+TP} = \frac{10}{10+40} = 0.2$

$$\text{Recall} = TPR = \frac{TP}{TP+FN} = 0.8$$

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{40}{40+5} = \frac{8}{9} = 0.889$$

$$\text{Acc} = \frac{TP+TN}{TP+FP+TN+FN} = \frac{85}{100} = 0.85$$

$$F_1\text{-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times \frac{8}{9} \times 0.8}{\frac{8}{9} + 0.8} = 0.842$$

3 K-Nearest Neighbors (10 Points)

KNN.py × NaiveBayes.py KMeans.py DecisionTree.py CNN1.py CNN2.py

```python
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import cosine_distances

data = [
    [1, 1.0, 2.0, 3.0, 'A'],
    [2, 0.5, 1.8, 2.7, 'B'],
    [3, 1.2, 2.2, 3.5, 'B'],
    [4, 4.6, 5.6, 3.7, 'A'],
    [5, 2.4, 4.6, 3.6, 'A'],
    [6, 3.5, 2.0, 4.1, 'B'],
    [7, 3.6, 4.6, 7.1, 'A'],
    [8, 6.2, 4.1, 1.3, 'B'],
    [9, 8.4, 3.5, 1.8, 'A'],
    [10, 5.8, 3.4, 2.7, 'B']
]

df = pd.DataFrame(data, columns=['ID', 'x', 'y', 'z', 'Label'])

target = np.array([3.5, 4.0, 6.0])

# Compute L1 distances
df['L1_Distance'] = df[['x', 'y', 'z']].apply(lambda row: np.sum(np.abs(row - target)), axis=1)

# Compute Cosine distances
features = df[['x', 'y', 'z']].values
cos_dist = cosine_distances(features, target.reshape(1, -1)).flatten()
df['Cosine_Distance'] = cos_dist

# Sort and get top 3 neighbors for each metric
top3_l1 = df.nsmallest(3, 'L1_Distance')
top3_cosine = df.nsmallest(3, 'Cosine_Distance')

# Determine classification by majority vote
l1_result = top3_l1['Label'].mode()[0]
cosine_result = top3_cosine['Label'].mode()[0]
```

运行 KNN ×

```
E:\CV\Assignment1\pythonProject1\.venv\Scripts\python.exe E:\CV\Assignment1\pythonProject1\KNN.py
Full Distance Table:
   ID    x    y    z Label  L1_Distance  Cosine_Distance
0   1  1.0  2.0  3.0     A          7.5         0.016393
1   2  0.5  1.8  2.7     B          8.5         0.044365
2   3  1.2  2.2  3.5     B          6.6         0.014618
3   4  4.6  5.6  3.7     A          5.0         0.069342
4   5  2.4  4.6  3.6     A          4.1         0.043839
5   6  3.5  2.0  4.1     B          3.9         0.026862
6   7  3.6  4.6  7.1     A          1.8         0.001298
7   8  6.2  4.1  1.3     B          7.5         0.241130
8   9  8.4  3.5  1.8     A          9.6         0.271068
9  10  5.8  3.4  2.7     B          6.2         0.137294

Top 3 Neighbors by L1 Distance:
   ID Label  L1_Distance
6   7     A          1.8
5   6     B          3.9
4   5     A          4.1

Top 3 Neighbors by Cosine Distance:
   ID Label  Cosine_Distance
6   7     A         0.001298
2   3     B         0.014618
0   1     A         0.016393

Final Classification by L1 Distance: A
Final Classification by Cosine Distance: A

进程已结束，退出代码为 0
```

ythonProject1 > KNN.py

4 Naive Bayes Classifier (10 Points)

KNN.py NaiveBayes.py × KMeans.py DecisionTree.py CNN1.py CNN2.py

```python
    'Humidity': 'Normal',
    'Windy': False
}

# Prior probabilities
total = len(df)
yes_count = len(df[df['Play Golf'] == 'Yes'])
no_count = len(df[df['Play Golf'] == 'No'])

P_yes = yes_count / total
P_no = no_count / total

# Conditional probabilities
2 个用法  Rezela
def conditional_prob(feature, value, label):
    subset = df[df['Play Golf'] == label]
    count = len(subset[subset[feature] == value])
    return count / len(subset)

features = ['Outlook', 'Temperature', 'Humidity', 'Windy']
P_X_given_yes = np.prod([conditional_prob(f, new_sample[f],  label: 'Yes') for f in features])
P_X_given_no = np.prod([conditional_prob(f, new_sample[f],  label: 'No') for f in features])

# Posterior probabilities
posterior_yes = P_X_given_yes * P_yes
posterior_no = P_X_given_no * P_no

prediction = 'Yes' if posterior_yes > posterior_no else 'No'

print("Prior P(Yes):", round(P_yes, 3))
print("Prior P(No):", round(P_no, 3))
print("Likelihood P(X|Yes):", round(P_X_given_yes, 5))
print("Likelihood P(X|No):", round(P_X_given_no, 5))
print("Posterior P(Yes|X):", round(posterior_yes, 5))
print("Posterior P(No|X):", round(posterior_no, 5))
print("Prediction for new sample:", prediction)
```

行 NaiveBayes ×

```
E:\CV\Assignment1\pythonProject1\.venv
Prior P(Yes): 0.625
Prior P(No): 0.375
Likelihood P(X|Yes): 0.0192
Likelihood P(X|No): 0.04938
Posterior P(Yes|X): 0.012
Posterior P(No|X): 0.01852
Prediction for new sample: No

进程已结束，退出代码为 0
```

5 Decision Tree (15 Points)

```
4 个用法  Rezela
def entropy(class_list):
    total = len(class_list)
    counts = Counter(class_list)
    return -sum((count / total) * math.log2(count / total) for count in counts.values())


1 个用法  Rezela
def info_gain(df, attr, target='Class'):
    total_entropy = entropy(df[target])
    values = df[attr].unique()
    weighted_entropy = 0
    for v in values:
        subset = df[df[attr] == v]
        weighted_entropy += (len(subset) / len(df)) * entropy(subset[target])
    return total_entropy - weighted_entropy
```

KNN.py   NaiveBayes.py ×   KMeans.py   DecisionTree.py ×   CNN1.py   CNN2.py

```
def id3(df, target='Class', attributes=None, depth=0):
    indent = "  " * depth
    classes = Counter(df[target])

    # 如果纯净，返回类别
    if len(classes) == 1:
        return list(classes.keys())[0]

    if attributes is None:
        attributes = [col for col in df.columns if col not in [target, 'ID']]

    # 当前熵
    current_entropy = entropy(df[target])
    print(f"\n{indent}Tree node (Depth {depth})")
    print(f"{indent}Current Entropy: {current_entropy:.4f}")

    # 计算每个属性的信息增益
    gains = {attr: info_gain(df, attr, target) for attr in attributes}
    for attr, g in gains.items():
        print(f"{indent}Information Gain for - {attr}: {g:.4f}")

    # 选择最佳属性
    best_attr = max(gains, key=gains.get)
    print(f"{indent}Choose: {best_attr}")

    tree = {best_attr: {}}
    for v in df[best_attr].unique():
        subset = df[df[best_attr] == v]
        branch_entropy = entropy(subset[target])
        branch_counts = dict(Counter(subset[target]))
        print(f"{indent}  Branch {best_attr}={v}: distribution {branch_counts}, entropy={branch_entropy:.4f}")

        if len(subset) == 0:
            tree[best_attr][v] = classes.most_common(1)[0][0]
        else:
            new_attrs = [a for a in attributes if a != best_attr]
            tree[best_attr][v] = id3(subset, target, new_attrs, depth + 1)
    return tree
```

DecisionTree

```
E:\CV\Assignment1\pythonProject1\.venv\Scripts\python.exe E:\CV\Assignment1\pythonProject1\DecisionTree.py

Tree node (Depth 0)
Current Entropy: 0.7642
Information Gain for - Weather: 0.4581
Information Gain for - Temperature: 0.1409
Information Gain for - Wind: 0.2248
Choose: Weather
  Branch Weather=Sunny: distribution {'No': 2, 'Yes': 1}, entropy=0.9183

  Tree node (Depth 1)
  Current Entropy: 0.9183
  Information Gain for - Temperature: 0.2516
  Information Gain for - Wind: 0.9183
  Choose: Wind
    Branch Wind=High: distribution {'No': 2}, entropy=-0.0000
    Branch Wind=Medium: distribution {'Yes': 1}, entropy=-0.0000
  Branch Weather=Overcast: distribution {'Yes': 3}, entropy=-0.0000
  Branch Weather=Rain: distribution {'Yes': 3}, entropy=-0.0000

Final Decision Tree:
{'Weather': {'Sunny': {'Wind': {'High': 'No', 'Medium': 'Yes'}}, 'Overcast': 'Yes', 'Rain': 'Yes'}}

进程已结束，退出代码为 0
```

6 K-Means (10 Points)

KNN.py NaiveBayes.py KMeans.py DecisionTree.py CNN1.py CNN2.py

```python
import numpy as np
import pandas

data = [
    [2.0, 3.0, 1.0],
    [1.5, 2.5, 1.2],
    [8.0, 7.0, 9.0],
    [7.5, 6.5, 8.5],
    [2.5, 3.5, 1.5],
    [7.0, 6.0, 8.0],
    [1.8, 2.8, 1.1],
    [8.5, 7.5, 9.5]
]

columns = ['x', 'y', 'z']
df = pandas.DataFrame(data, columns=columns)

point1 = df.iloc[0].values
point2 = df.iloc[2].values

for i in range(2):
    print("Iteration: ", i+1)
    clusters = {0: [], 1: []}
    for j in range(len(df)):
        p = df.iloc[j].values
        distance1 = np.linalg.norm(p-point1)
        distance2 = np.linalg.norm(p-point2)
        # clusters[0 if distance1 < distance2 else 1].append(df.iloc[j])  # 保存点的数组到对应的簇
        clusters[0 if distance1 < distance2 else 1].append(j+1)  # 保存点的索引到对应的簇
    print("cluster[0]: ",clusters[0])
    print("cluster[1]: ",clusters[1])
    point1 = np.mean(df.iloc[[i-1 for i in clusters[0]]].values, axis=0)
    point2 = np.mean(df.iloc[[i-1 for i in clusters[1]]].values, axis=0)
    print("centroid 1: ",  point1,
          "\ncentroid 2: ", point2)
```

运行 KMeans

```
E:\CV\Assignment1\pythonProject1\.ven
Iteration:  1
cluster[0]:  [1, 2, 5, 7]
cluster[1]:  [3, 4, 6, 8]
centroid 1:  [1.95 2.95 1.2 ]
centroid 2:  [7.75 6.75 8.75]
Iteration:  2
cluster[0]:  [1, 2, 5, 7]
cluster[1]:  [3, 4, 6, 8]
centroid 1:  [1.95 2.95 1.2 ]
centroid 2:  [7.75 6.75 8.75]

进程已结束，退出代码为 0
```

7 CNN Q1 (15 Points)

KNN.py NaiveBayes.py KMeans.py DecisionTree.py CNN1.py × CNN2.py

```python
import tensorflow as tf

model = tf.keras.Sequential(
    [
        tf.keras.layers.Conv2D(8, (3, 3), strides=2, padding="same", input_shape=(64, 64, 1)),
        tf.keras.layers.Conv2D(16, (3, 3), strides=2, padding="same"),
        tf.keras.layers.Conv2D(32, (3, 3), strides=1, padding="same"),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(10)
    ]
)

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 8)	80
conv2d_1 (Conv2D)	(None, 16, 16, 16)	1,168
conv2d_2 (Conv2D)	(None, 16, 16, 32)	4,640
max_pooling2d (MaxPooling2D)	(None, 8, 8, 32)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 10)	20,490

```
Total params: 26,378 (103.04 KB)
Trainable params: 26,378 (103.04 KB)
Non-trainable params: 0 (0.00 B)

进程已结束，退出代码为 0
```

8 CNN Q2 (15 Points)

KNN.py NaiveBayes.py KMeans.py DecisionTree.py CNN1.py CNN2.py

```python
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(1, (3, 3), strides=1, padding="valid", input_shape=(5, 5, 1), use_bias=False),
    tf.keras.layers.ReLU(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1, use_bias=False)
])

K = np.array([
    [[[1]], [[0]], [[-1]]],
    [[[1]], [[0]], [[-1]]],
    [[[1]], [[0]], [[-1]]]
], dtype=np.float32).reshape(3, 3, 1, 1)
model.layers[0].set_weights([K])
# ValueError: You called `set_weights(weights)` on layer 'conv2d' with a weight list of length 1, but the layer was expecting 2 weights.
# Must set Conv2D layer with use_bias=False

W = np.array([
    [1], [-2], [1], [-1], [-3], [0], [1], [-1], [1]
], dtype=np.float32).reshape(9, 1)
model.layers[-1].set_weights([W])
# ValueError: You called `set_weights(weights)` on layer 'dense' with a weight list of length 1, but the layer was expecting 2 weights.
# Must set Dense layer with use_bias=False

A = np.array([
    [2, 0, 14, 5, 4],
    [11, 12, 16, 4, 8],
    [12, 6, 12, 3, 12],
    [10, 10, 2, 20, 11],
    [4, 11, 8, 12, 4]
], dtype=np.float32).reshape(1, 5, 5, 1)

output = model(A)
print("Output: ", output.numpy().item())
```

```
Output:  4.0

进程已结束，退出代码为 0
```