MATH 4/5388: Machine Learning Methods

Homework 5

Due date: Thursday, April 10

Submission Instruction:

- Submit both the Jupyter notebook file (.ipynb) and a PDF copy of the notebook.
- Ensure that your notebook runs properly before submitting it:
  - Kernel -> Restart & Run All to ensure that there are no errors.
- To generate a PDF of your notebook:
  - File -> Print Preview followed by printing to a PDF from your browser; or:
  - File -> Download as -> PDF via LaTeX.
- If this doesn't work, try first exporting as an HTML file and then converting that to PDF (load it in a web browser and print it to PDF).

Problem 1 (40 points)

In this problem, we explore how decision trees make predictions and evaluate splits for both classification and regression tasks. You'll analyze impurity measures used for classification and think about how tree depth constrains predictions in regression.

Part A: Gini Impurity for Binary Classification

Gini impurity is a common metric for evaluating splits in decision trees for classification. For the binary case (two classes), the Gini impurity is defined as:

$$G(p) = 2p(1-p)$$

where `p` is the probability of one of the two classes (say, class 1), and `1 - p` is the probability of the other class (class 0).

(i) Show that the Gini impurity $G(p)$ is maximized when the two class probabilities are equal.
(ii) What is the maximum value of Gini impurity?

Part B: Entropy for Binary Classification

Another common impurity measure is entropy, defined for the binary case as:

$$H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$$

(i) Plot or analyze the function $H(p)$ for $p \in [0, 1]$.
(ii) Determine whether the entropy is maximized at the same value of p as the Gini impurity

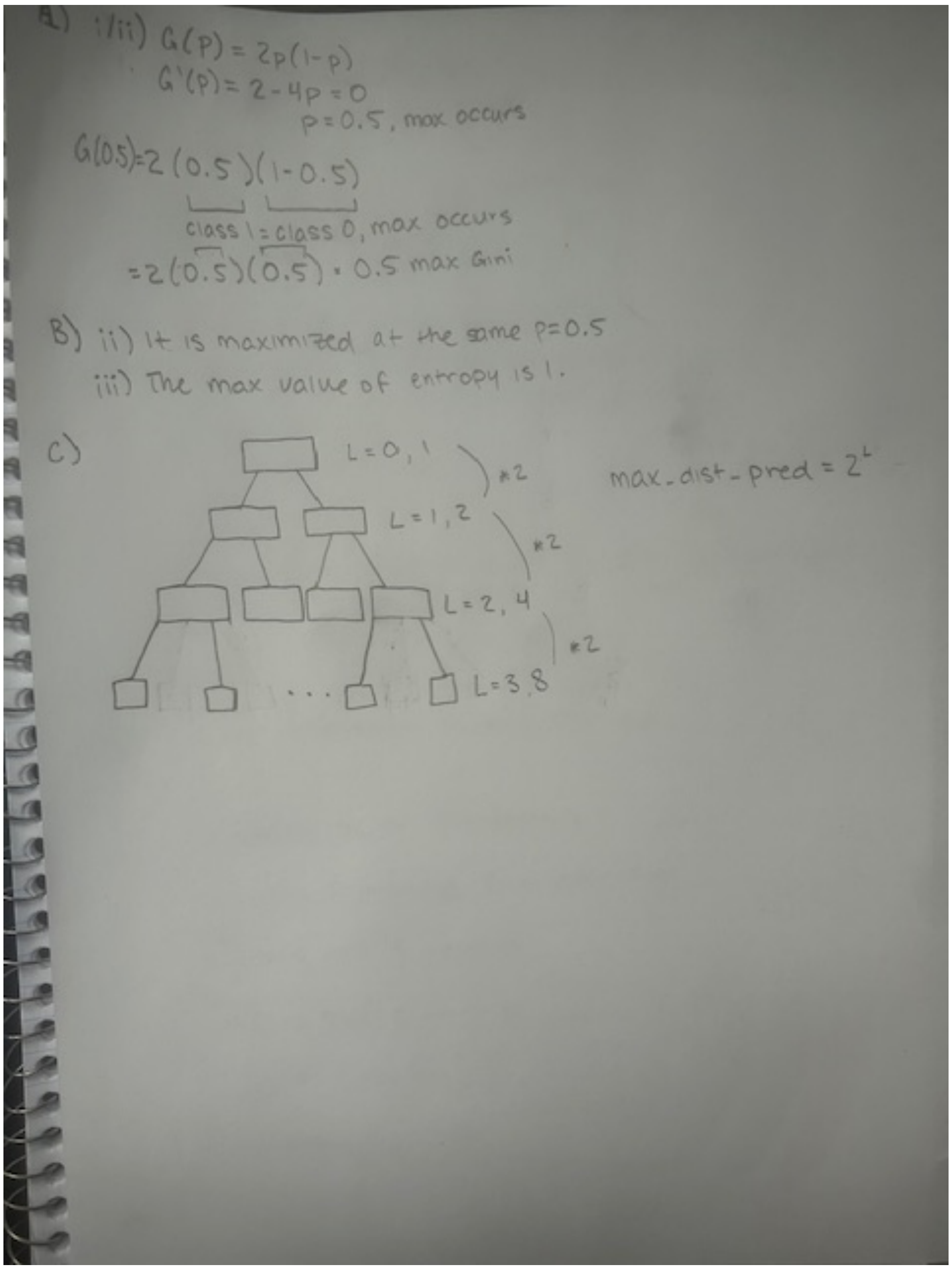
(ii) Determine whether the entropy is maximized at the same value of p as the Gini impurity.
(iii) What is the maximum value of entropy?

Part C: Maximum Number of Leaves in Regression Trees

In decision trees for regression, each leaf predicts a constant value (typically the mean of target values in that region). In lecture, we looked at how tree depth influences the number of regions (and thus predictions).

Suppose we train a regression tree with a maximum depth of L, and splits are binary at each internal node.

What is the maximum number of distinct predicted values the tree can produce? Justify your answer based on the structure of a binary tree of depth L.



A) i/ii) $G(p) = 2p(1-p)$
$G'(p) = 2-4p = 0$
$p = 0.5$, max occurs

$G(0.5) = 2\underbrace{(0.5)}_{}\underbrace{(1-0.5)}_{}$
class 1 = class 0, max occurs
$= 2(0.5)(0.5) = 0.5$ max Gini

B) ii) It is maximized at the same $p=0.5$
iii) The max value of entropy is 1.

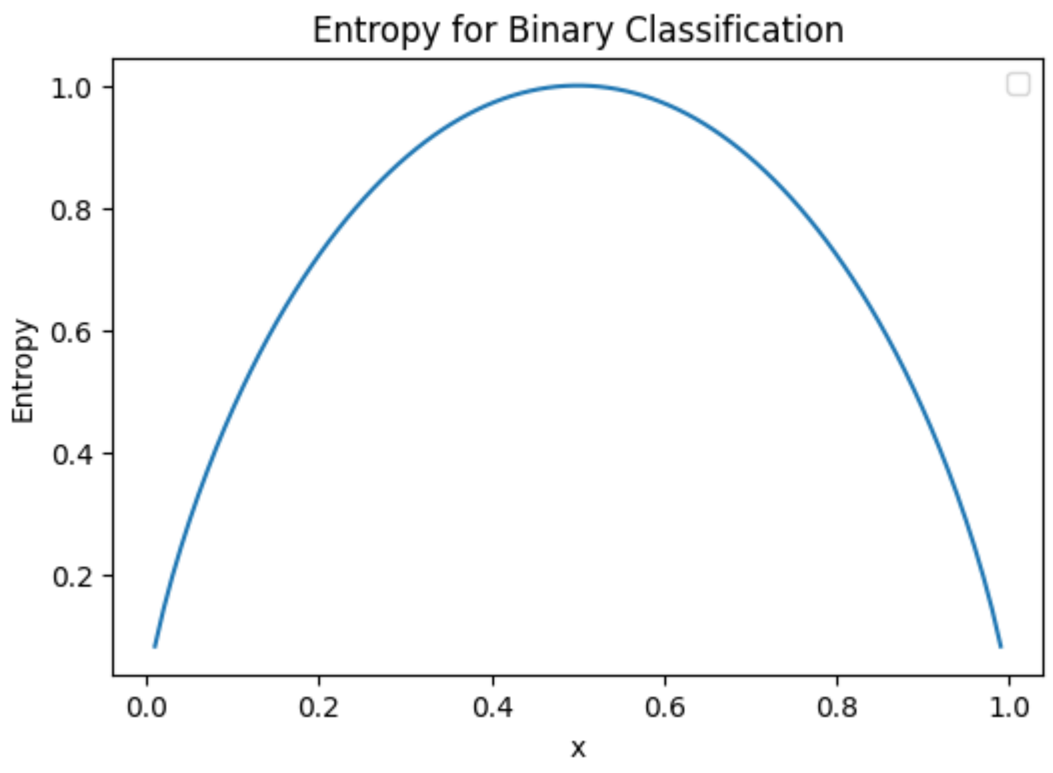
C)
L=0, 1
*2
L=1, 2
*2
L=2, 4
*2
L=3, 8
. . .

max-dist-pred = 2^L

```
# Part B (i)
```

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 1, 100)

entropy = -x * np.log2(x) - (1 - x) * np.log2(1 - x)

plt.figure(figsize=(6, 4))
plt.plot(x, entropy)
plt.title('Entropy for Binary Classification')
plt.xlabel('x')
plt.ylabel('Entropy')
plt.legend()
plt.show()
```

```
<ipython-input-1-8315b92912be>:8: RuntimeWarning: divide by zero encountere
  entropy = -x * np.log2(x) - (1 - x) * np.log2(1 - x)
<ipython-input-1-8315b92912be>:8: RuntimeWarning: invalid value encountered
  entropy = -x * np.log2(x) - (1 - x) * np.log2(1 - x)
<ipython-input-1-8315b92912be>:15: UserWarning: No artists with labels foun
  plt.legend()
```



Problem 2 (20 points)

In this exercise, you will explore the difference between the **ε-insensitive loss** used in Support Vector Regression (SVR) and the **mean squared error (MSE)** used in ordinary regression.

Given a true value y, the ε-insensitive loss is defined as:

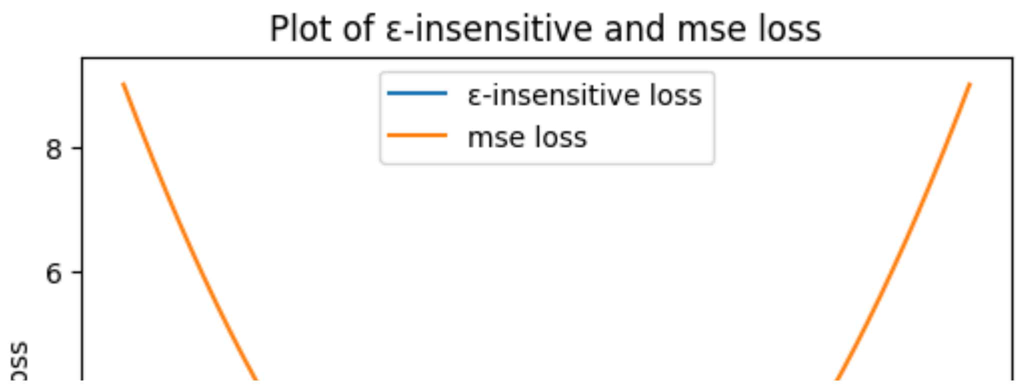
$$L_\varepsilon(y, \hat{y}) = \begin{cases} 0 & \text{if } |\hat{y} - y| \leq \varepsilon \\ |\hat{y} - y| - \varepsilon & \text{otherwise} \end{cases}$$

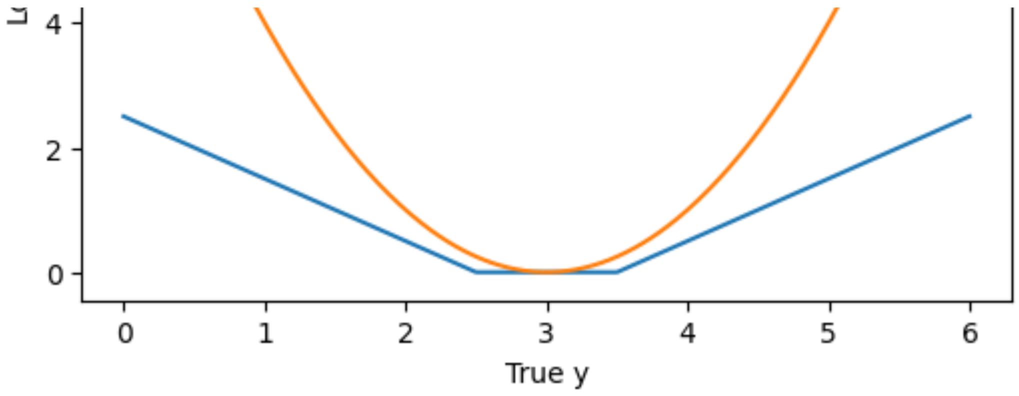
Let the true target value be $y = 3$, and let the prediction $\hat{y}$ vary over the range $[0, 6]$. Set $\varepsilon = 0.5$.

1. Write the ε-insensitive loss function as a single expression (i.e., without using a piecewise "case" definition). Use standard mathematical functions such as absolute value and maximum.
2. Plot the ε-insensitive loss $L_\varepsilon(3, \hat{y})$.
3. Plot the squared loss $L_{\text{MSE}}(3, \hat{y})$. Use the same axes and clearly label the functions. The x-axis should be $\hat{y} \in [0, 6]$, and the y-axis should be the loss value.
4. Based on the plots, explain why Support Vector Regression might behave differently from ordinary least squares regression, especially in the presence of small fluctuations.

Lϵ(y, ŷ) = max(|y- ŷ| - ϵ, 0)

```
import numpy as np
import matplotlib.pyplot as plt

x_eps = np.linspace(0, 6, 400)

y_eps = np.maximum(np.abs(3 - x_eps) - 0.5, 0)

x_mse = np.linspace(0, 6, 400)

target = np.full(400, 3)

y_mse = (x_mse - target) ** 2

plt.figure(figsize=(6, 4))
plt.plot(x_eps, y_eps, label='ε-insensitive loss')
plt.plot(x_mse, y_mse, label='mse loss')
plt.title('Plot of ε-insensitive and mse loss')
plt.xlabel('True y')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





Based on the plot above, the epsilon loss function would be less sensitive to values which are farther from the actual value - compared to the MSE where a small fluctuation could lead to a large fluctuation in the loss based on the growth of a quadratic. Moreover, values that are close to the actual value, within an epsilon ball, aren't penalized at all with the epsilon loss function, and wouldn't be penalized for small fluctuations, whereas the MSE would.

Problem 3 (40 points)

In this problem, you will compare the performance of three popular regression models on a real-world dataset using scikit-learn:

- **Support Vector Regression (SVR)**
- **Decision Tree Regression**
- **Random Forest Regression**

Use the built-in **Diabetes dataset** from scikit-learn:

```
from sklearn.datasets import load_diabetes
```

1. Load and preprocess the data:
   - Split the dataset into training and test sets (e.g., 80% training / 20% testing).
   - Standardize the features (important for SVR).
2. Train the following regression models:
   - Support Vector Regression (`SVR`)
   - Decision Tree Regressor (`DecisionTreeRegressor`)
   - Random Forest Regressor (`RandomForestRegressor`)
3. Evaluate each model on the test set:
   - Report performance using Mean Squared Error (MSE) and R^2 score.
4. Experiment with key hyperparameters:

- For Decision Trees and Random Forests, vary `max_depth` and observe the impact.
- For SVR, try different values of `C` (regularization strength) and compare at least two kernel types (`'linear'` and `'rbf'`).

```
# Load and preprocess

from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

data = load_diabetes()

X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random


# SVR with different regularization and kernels

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error

# Default
svr_clf = make_pipeline(StandardScaler(),
                                   SVR())

svr_clf.fit(X_train, y_train)

y_pred = svr_clf.predict(X_test)

print('Metrics for SVR:')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')

# Adjust kernel
svr_clf = make_pipeline(StandardScaler(),
                                   SVR(kernel='linear'))

svr_clf.fit(X_train, y_train)

y_pred = svr_clf.predict(X_test)

print('Metrics for SVR(linear):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')

# Adjust c
svr_clf = make_pipeline(StandardScaler(),
                                   SVR(kernel='linear', C=.5))

svr_clf.fit(X_train, y_train)
```

```
y_pred = svr_clf.predict(X_test)

print('Metrics for SVR(linear with adjusted C = 0.5):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')

svr_clf = make_pipeline(StandardScaler(),
                                   SVR(kernel='linear', C=10))

svr_clf.fit(X_train, y_train)

y_pred = svr_clf.predict(X_test)

print('Metrics for SVR(linear with adjusted C = 10):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')
```

```
    Metrics for SVR:
    R2 score: 0.18221699094239507
    Mean squared error: 4332.738479345717

    Metrics for SVR(linear):
    R2 score: 0.44512484353785875
    Mean squared error: 2939.8127804183164

    Metrics for SVR(linear with adjusted C = 0.5):
    R2 score: 0.448128920708151
    Mean squared error: 2923.8967237058537

    Metrics for SVR(linear with adjusted C = 10):
    R2 score: 0.4298747201726798
    Mean squared error: 3020.6102481906332
```

```
# Decision tree with differing max_depth

from sklearn.tree import DecisionTreeRegressor

tree = DecisionTreeRegressor(random_state=42)

tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)

print('Metrics for Decision Tree (no max depth):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')

tree = DecisionTreeRegressor(max_depth=5, random_state=42)

tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)
```

```
print('Metrics for Decision Tree (max depth=5):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')

tree = DecisionTreeRegressor(max_depth=10, random_state=42)

tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)

print('Metrics for Decision Tree (max depth=10):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')

tree = DecisionTreeRegressor(max_depth=15, random_state=42)

tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)

print('Metrics for Decision Tree (max depth=15):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')
```

```
Metrics for Decision Tree (no max depth):
R2 score: 0.060653981041140725
Mean squared error: 4976.797752808989

Metrics for Decision Tree (max depth=5):
R2 score: 0.3344819704370976
Mean squared error: 3526.0155119980145

Metrics for Decision Tree (max depth=10):
R2 score: 0.17179020054241645
Mean squared error: 4387.98119713478

Metrics for Decision Tree (max depth=15):
R2 score: 0.13544674643051302
Mean squared error: 4580.534332084894
```

```
# Random forest with differing max_depth

from sklearn.ensemble import RandomForestRegressor

tree = RandomForestRegressor(random_state=42)

tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)

print('Metrics for Random Forest (no max depth):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')
```

```
tree = RandomForestRegressor(max_depth=5, random_state=42)

tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)

print('Metrics for Random Forest (max depth=5):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')

tree = RandomForestRegressor(max_depth=10, random_state=42)

tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)

print('Metrics for Random Forest (max depth=10):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')

tree = RandomForestRegressor(max_depth=15, random_state=42)

tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)

print('Metrics for Random Forest (max depth=15):')
print(f'R2 score: {r2_score(y_test, y_pred)}')
print(f'Mean squared error: {mean_squared_error(y_test, y_pred)}\n')
```

```
Metrics for Random Forest (no max depth):
R2 score: 0.4428225673999313
Mean squared error: 2952.0105887640448

Metrics for Random Forest (max depth=5):
R2 score: 0.4561161622977984
Mean squared error: 2881.5791057118386

Metrics for Random Forest (max depth=10):
R2 score: 0.43863514832194295
Mean squared error: 2974.1961704738605

Metrics for Random Forest (max depth=15):
R2 score: 0.44971785164207745
Mean squared error: 2915.478326499999
```