**Assignment 1**

By Harsh Sharma

1. Machine learning is often referred to as an ill-posed problem. **(1)** What does **ill-posed problem** mean? **(2)** How do machine learning algorithms deal with the fact that machine learning is an ill-posed problem?

**Solution**

An ill-posed problem mean for which a unique solution cannot be determined using only the information that is available. These types of problems lack one or more of the fundamental properties required to have a unique and stable solution.

In terms of Machine Learning, there may be multiple solutions or no solution at all which perfectly fits the problem.

**Dealing with this problem in Machine Learning:**

We can use ensemble learning where we can combine the outputs of multiple model to get one final result. Different models could work effectively on different aspects of problem and dataset and hence improve the accuracy of result.

Also, we can try implementing domain based knowledge while training the input. Providing additional information while training can provide reasonable improvement, especially when the information is originated after years of experience.

1. In machine learning, **(1)** what is meant by the term **inductive bias**? **(2)** What can go wrong when an inappropriate inductive bias is used?

Solution

Inductive bias is a bias which occurs at the time of training a model. It is a set of assumptions, expectations, or prior beliefs that influence the learning algorithm's choice of a particular hypothesis or model over others when faced with ambiguous or incomplete data.

Problem with Inductive Bias:

**Overfitting**: If the bias is too strong towards one outcome and doesn’t capture the true data distribution, the model might be overfitted.

**Failure to Generalize:** An inappropriate inductive bias can lead to models that are overly specialized to the training data, making them unable to generalize well to unseen or new data. This is detrimental for the model's real-world applicability.

**Difficulty in Learning:** The learning algorithm may struggle to find an appropriate model or hypothesis due to conflicting or misleading inductive biases. This can lead to prolonged training times, convergence issues, or failure to converge.

**3. (1)** What is meant by the term **consistent model**? **(2)** Why might a consistent model not **generalize** well?

Solution:

A consistent model is a model that can perfectly fit the training data. It can achieve zero training error by learning the true underlying pattern or function that generated the data.

Problem with generalization:

A consistent model may end up fitting not only the true underlying pattern in the data but also the noise or random fluctuations present in the training data. Therefore this complex model won’t generalize well to new/unseen data as it has kind of memorized the noise.

**4. Consider the following set of training examples:**

This dataset shows a set of stroke risk factors and their probability of suffering a stroke in the next five years (*low*, *medium*, and *high*). All the descriptive features are Boolean, taking two levels: *true* or *false*.

How many possible models exist for the scenario described by the features in this dataset? Explain your answer.

Solution:

The dataset contains a total of 4 features and one target variable. The four features each can have two possible values, true or false. So one single feature can have a total of 2 outcomes. But since we have four features, the possible outcomes will be 2 to power of 4. i.e, 24 = 32.

And since each of these 32 possible outcomes can result in 3 possible outcomes to our target variable, the total number of possible model that could exist is = 3 to power of 32 = 332 = 1853020188851841

**5.** Briefly explain **mathematical optimization** for machine learning.

It involves finding the best solution for a mathematical function under given constraints. In the context of machine learning, optimization is crucial for training models and adjusting their parameters to minimize the error or loss function, ultimately leading to a better predictive performance.

The process of determining proper parameters for a model relies on a set of tools known as mathematical optimization.

Suppose a line model. The machine needs to find the right values for its two parameters – a slope and vertical intercept.

For example:

Objective Function (Cost/Loss Function):

The first step is defining an objective function, often referred to as a cost or loss function. This function measures the discrepancy between the predicted output of the model and the actual ground truth. The goal here is to minimize this function.

**Part II. Data Exploration**

In Part II, you will have hands-on data exploration. You can use any machine learning API or tool, or any program language. Answer to each of the following questions and also submit your program codes used for Part II.

1. **Check there are any missing values in the dataset. If any, report it.**

**Solution:** No, there are no missing values in the dataset.

1. Report the summary statistics for the TAX feature (full-value property-tax rate per $10,000) with **(1)** minimum, maximum, and range, **(2)** mean and media, **(3)** variance and standard deviation, **(4)** 1st quartile and 3rd quartile, **(5)** inter-quartile range, **(6)**12th percentile.

Solution:

Summary Statistics for TAX feature:

Minimum: 187.0

Maximum: 711.0

Range: 524.0

Mean: 408.2371541501976

Median: 330.0

Variance: 28404.75948812273

Standard Deviation: 168.53711605495903

1st Quartile: 279.0

3rd Quartile: 666.0

Inter-quartile Range: 387.0

12th Percentile: 243.6

1. Show (1) a **histogram** for each numerical feature including the target feature, MEDV. **(2)** Is there any features which show “bimodal” distributions?

Solution:

The histograms of each numerical feature are:

A graph of a graph

Description automatically generated

A graph of a number of individuals

Description automatically generated

A graph of blue bars

Description automatically generated

A graph of blue bars

Description automatically generated

A graph of a graph

Description automatically generated

A graph of age and age

Description automatically generated

A graph of dis

Description automatically generated

A graph of a graph

Description automatically generated

A graph of tax

Description automatically generated

A graph of a number of columns

Description automatically generated with medium confidence

A graph of a number of people

Description automatically generated with medium confidence

A graph of a number of bars

Description automatically generated with medium confidence

A graph of a patient's height

Description automatically generated with medium confidence

**3 (2)** Is there any features which show “bimodal” distributions?

Upon visually inspecting the histograms I could find these features to be bimodal features: TAX, RAD. Also, INDUS could be one.

1. Show (1) a **scatter plot matrix** of numeric features from the dataset to check for correlation between features. (2) Which feature pairs show positive correlation? (3) Which feature pairs show negative correlation?

A grid of blue lines

Description automatically generated

Feature pairs with positive correlation:

[('CRIM', 'RAD'), ('CRIM', 'TAX'), ('ZN', 'DIS'), ('INDUS', 'NOX'), ('INDUS', 'AGE'), ('INDUS', 'RAD'), ('INDUS', 'TAX'), ('INDUS', 'LSTAT'), ('NOX', 'INDUS'), ('NOX', 'AGE'), ('NOX', 'RAD'), ('NOX', 'TAX'), ('NOX', 'LSTAT'), ('RM', 'MEDV'), ('AGE', 'INDUS'), ('AGE', 'NOX'), ('AGE', 'TAX'), ('AGE', 'LSTAT'), ('DIS', 'ZN'), ('RAD', 'CRIM'), ('RAD', 'INDUS'), ('RAD', 'NOX'), ('RAD', 'TAX'), ('TAX', 'CRIM'), ('TAX', 'INDUS'), ('TAX', 'NOX'), ('TAX', 'AGE'), ('TAX', 'RAD'), ('TAX', 'LSTAT'), ('LSTAT', 'INDUS'), ('LSTAT', 'NOX'), ('LSTAT', 'AGE'), ('LSTAT', 'TAX'), ('MEDV', 'RM')]

Feature pairs with negative correlation:

[('ZN', 'INDUS'), ('ZN', 'NOX'), ('ZN', 'AGE'), ('INDUS', 'ZN'), ('INDUS', 'DIS'), ('NOX', 'ZN'), ('NOX', 'DIS'), ('RM', 'LSTAT'), ('AGE', 'ZN'), ('AGE', 'DIS'), ('DIS', 'INDUS'), ('DIS', 'NOX'), ('DIS', 'AGE'), ('DIS', 'TAX'), ('TAX', 'DIS'), ('PTRATIO', 'MEDV'), ('LSTAT', 'RM'), ('LSTAT', 'MEDV'), ('MEDV', 'PTRATIO'), ('MEDV', 'LSTAT')]

**5.** Conduction a **heatmap** which shows correlation values for every feature pairs.

A screenshot of a graph

Description automatically generated

**6.** Conduct **standardization** on all numeric features except the target feature, MEDV*.*

Standardized Dataset:

This contains only the first 7 values of the entire dataset.

CRIM ZN INDUS NOX RM AGE DIS

0 -0.419782 0.284830 -1.287909 -0.144217 0.413672 -0.120013 0.140214 \

1 -0.417339 -0.487722 -0.593381 -0.740262 0.194274 0.367166 0.557160

2 -0.417342 -0.487722 -0.593381 -0.740262 1.282714 -0.265812 0.557160

3 -0.416750 -0.487722 -1.306878 -0.835284 1.016303 -0.809889 1.077737

4 -0.412482 -0.487722 -1.306878 -0.835284 1.228577 -0.511180 1.077737

5 -0.417044 -0.487722 -1.306878 -0.835284 0.207096 -0.351157 1.077737

6 -0.410243 0.048772 -0.476654 -0.265154 -0.388411 -0.070229 0.839244

RAD TAX PTRATIO B LSTAT MEDV MEDV CHAS

0 -0.982843 -0.666608 -1.459000 0.441052 -1.075562 0.159686 24.0 0

1 -0.867883 -0.987329 -0.303094 0.441052 -0.492439 -0.101524 21.6 0

2 -0.867883 -0.987329 -0.303094 0.396427 -1.208727 1.324247 34.7 0

3 -0.752922 -1.106115 0.113032 0.416163 -1.361517 1.182758 33.4 0

4 -0.752922 -1.106115 0.113032 0.441052 -1.026501 1.487503 36.2 0

5 -0.752922 -1.106115 0.113032 0.410571 -1.043322 0.671222 28.7 0

6 -0.523001 -0.577519 -1.505237 0.426798 -0.031268 0.039964 22.9 0

**Part III. Dimensionality Reduction**

Use PCA to reduce the dataset’s dimensionality from 30 to 2, and then generate the two-dimensional scatter plot of the breast cancer dataset using the first two principal components, where each data point is shaped and colored according to its class label, i.e., blue dot for M and red triangle for B.

Solution:

I’m sharing the scatter plot below. Also, I’ll be attaching the code as jupyter notebook names as ‘part 3’. The dataset after reduced dimensionality is also saved as **wdbc\_2D.data** and will be attached along with the code.

A diagram of a graph

Description automatically generated with medium confidence

**Part IV. Decision Tree Learning**

1. Build a decision tree model from *party.data,* using the programs provided. Report **(1)** the **true class label** and **predicted class label** of each (training) instance, and **(2)** the **training error rate.**

Solution:

True Class Label: ['Party', 'Study', 'Party', 'Party', 'Pub', 'Party', 'Study', 'TV', 'Party', 'Study']

Predicted Class Label : ['Party', 'Study', 'Party', 'Party', 'Pub', 'Party', 'Study', 'TV', 'Party', 'Study']

The predicted class is exactly the same as the actual one and therefore the training error rate is **zero (0)**.

Also, I’ve modified parts of the code so that it runs correctly. I’ll be attaching working code aswell. They are named**: party.py** and **dtree.py**

Q 2. The program uses a given dataset for both training and validation purpose. (1) Revise the program code to prepare a training dataset and a validation dataset from a user input dataset using ***bootstrap****.*

(2) Revise the program code for computing the **test error rate**.

Submit the revised program code with your explanation of the changed/added program parts.

Solution:

Here is the bootstrap function that I created to divide the dataset for training and testing:

def bootstrap\_split(data, percentage=0.8):

    """Split the data into training and validation sets using bootstrap."""

    num\_samples = len(data)

    train\_size = int(percentage \* num\_samples)

    # Generate a bootstrap sample

    indices = np.random.choice(range(num\_samples), size=train\_size, replace=True)

    train\_data = [data[i] for i in indices]

    # Validation set is the remaining data not included in the training set

    validation\_data = [data[i] for i in range(num\_samples) if i not in indices]

    return train\_data, validation\_data

To calculate test error:

error\_count = sum(1 for true\_class, predicted\_class in zip(true\_classes, predicted\_classes) if true\_class != predicted\_class)

test\_error\_rate = error\_count / len(true\_classes)

I also had to update other parts of the code to accommodate these changes. They have been incorporated in these files which I’ll also sharing: **party2.py** and **dtree2.py**

**3.** For impurity measures, three metrics are popular used: *entropy*, *gini index* and *misclassification error*

The *gini index* of a node *t* is computed as 1−Σ𝑝𝑖(𝑡)2𝑐𝑖=1 , where *c* is the number of distinct class labels and 𝑝𝑖(𝑡). *Information gain* can be also computed with the gini index.

Revise the program so that the program can chooses a test attribute based on **the *information gain* with *gini index*** instead of entropy-based information gain.

Submit the revised program code with your explanation of the changed/added program parts.

Solution:

**I modified make\_tree() function to use gini index for calculating Information gain.** This involves updating the calculation of information gain and selecting the best feature based on the Gini index.

**This is the code:**

gain = np.zeros(nFeatures)

            featureSet = list(range(nFeatures))

            for feature in featureSet:

                g = self.calc\_info\_gain(data, classes, feature)

                gain[feature] = g

            bestFeature = np.argmax(gain)

            tree = {featureNames[bestFeature]: {}}

Also, I needed to modify the calc\_info\_gain() function to compute information gain using the Gini index instead of entropy. The Gini index is calculated as 1−∑pi21−∑pi2​, where pipi​ is the proportion of samples belonging to class ii in the dataset.

This is the code for the same:

def calc\_info\_gain(self, data, classes, feature):

        # Calculates the information gain based on Gini index

        gini\_gain = 0

        nData = len(data)

        # List the values that feature can take

        values = np.unique([datapoint[feature] for datapoint in data])

        for value in values:

            value\_indices = [i for i, datapoint in enumerate(data) if datapoint[feature] == value]

            value\_classes = [classes[i] for i in value\_indices]

            # Calculate Gini index for the current value

            gini\_value = 1.0

            unique\_classes = np.unique(value\_classes)

            for cls in unique\_classes:

                p\_i = np.sum([1 for c in value\_classes if c == cls]) / len(value\_classes)

                gini\_value -= p\_i\*\*2

            gini\_gain += (len(value\_classes) / nData) \* gini\_value

        return gini\_gain

These changes have been updated in a new file and I’ll be sharing it as well. It’s called: **dtree3.py**

**4.** Rebuild a decision tree model from *party.data,* using the revised program.

Report **(1)** the **training error rate** and **(2)** the **test error rate**.

Solution:

To incorporate the changes I made in dtree3.py, I created a new file **party3.py** which then calls the previous **dtree3** class and gives the output. Also, in this file I’ve added the part to **calculate training error** which wasn’t implemented before.

One thing to note is that the test error rate varies with each run of code. It is so because the numbers of occurunces of data in the dataset varies and is random. And so the error also varies. In my various runs, it has varied from 0.25 to 0.888.

Whereas the training error rate remains constant at zero.