

RESEARCH & PROJECT SUBMISSIONS



Program:

Course Code: CSE323(UG2003)

Course Name: Data Structure

Examination Committee

**Prof. Islam Ahmed Mahmoud
Elmadah**

**Ain Shams University
Faculty of Engineering
Spring Semester – 2020**



Student Personal Information for Group Work

Student Names:

Mariam Abdelrahman Mahmoud
Manal Ahmed Mohamed
Nada Emam Kamal
Nermeen Osama Ramadan
Nesma Mohamed Atef

Student Codes:

1601374
1601449
1601558
1601575
1601579

Plagiarism Statement

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student Name: Mariam Abdelrahman Mahmoud Ali

Date: 31/5/2020

Signature/Student Name: Manal Ahmed Mohamed

Signature/Student Name: Nada Emam Kamal

Signature/Student Name: Nermeen Osama Ramadan

Signature/Student Name: Nesma Mohamed Atef



Submission Contents

01: Background

02: Implementation Details

- Dataset preparation
- Node and Tree classes
- Helping functions
- Decision Tree visualisation
- Accuracy metric
- Graphical User Interface

03: Complexity Of Operations

04: References

Github Repository Link: <https://github.com/alien19/Decision-Tree-Classifer>

Video Link: <https://www.youtube.com/watch?v=9pPcerApEJk&t=10s>

What is machine learning?

Machine learning is a field that is growing really fast, it's a field of study whose main concern is the design and analysis of algorithms which allow the computers to learn. with much more awaiting to be discovered than is currently known, nowadays we are using machine learning to teach computers to perform a very wide array of useful operations. This includes operations like the automatic detection of objects in images (self-driving cars), speech recognition (which powers voice command technology), knowledge discovery in the medical sciences (it improves our understanding of complex diseases), and predictive analytics (economic forecasting). In this report we are going to give a high-level introduction to a specific field of machine learning which is "The Classification".

What is Classification?

Classification is a two-step process, learning step and prediction step, in machine learning. In the learning step, the model is developed based on given training data. In the prediction step, the model is used to predict the response for given data. Decision Tree is one of the easiest and popular classification algorithms to understand and interpret.

02

Second Topic

Implementation details

Project Main Purpose:

In this project it is required to implement a model that classifies some reviews of a hotel (positive or negative) based on a dataset containing some features and a text review.

This classifier model is based on Decision Tree algorithm to classify whether this review is positive or negative.

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the Decision Tree algorithm can be used for solving regression and classification problems too.

The goal of using a Decision Tree is to create a training model that can be used to predict the class or value of the target variable by learning simple decision rules inferred from prior data (training data).

In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the bases of comparison, we follow the branch corresponding to that value and jump to the next node.

We are provided with three files:

- 1) **Train data** file: the hotel reviews that we should use to build the decision tree.
- 2) **Validation data** file: the hotel reviews that we don't use to build the tree but we use it to evaluate the already-built tree. we will use it to check how the decision tree works for reviews that it didn't see before.
- 3) **Test data** file: the hotel reviews that we don't know their ratings.

For Building The Decision Tree:

- Dataset preparation:

- 1) Function `data_init(file)` is implemented to take a csv file or its path and convert it to a data frame to make the dataset easy manipulated later on.

```
def data_init(file):  
    trainData = pd.read_csv(file)      #read the file  
    dataframe = pd.DataFrame(trainData) #convert the file into a data frame shape  
    return dataframe
```

- 2) Instead of using categorical data (provided in the dataset files) we will depend on the count of every feature in a text review, now our tree is based on continuous data.

This is achieved using the function `words_count(dataFrame)` which takes the data frame and replaces the old-features' columns with the new-features' columns (each column represents the number of counts of each feature)

```
def words_count(dataFrame):  
    d = []  
    if 'rating' in dataFrame.columns:  
        d = dataFrame.rating  
        dataFrame = dataFrame.drop('rating',axis=1) #remove the rating column from the dataframe  
  
    # this is the input features  
    features = ['contains_No', 'contains_Please', 'contains_Thank', 'contains_apologize', 'contains_bad',  
                'contains_clean', 'contains_comfortable', 'contains_dirty', 'contains_enjoyed', 'contains_friendly',  
                'contains_glad', 'contains_good', 'contains_great', 'contains_happy', 'contains_hot', 'contains_issues',  
                'contains_nice', 'contains_noise', 'contains_old', 'contains_poor', 'contains_right', 'contains_small',  
                'contains_smell', 'contains_sorry', 'contains_wonderful', 'reviews.text', 'count_reviews.text']  
  
    # for every column in the dataframe we add a new column to count how many time this feature is called in the review  
    for i in dataFrame.columns:  
        c = i.replace('contains_', '')  
        col = []  
        for j in range(0,dataFrame.shape[0]):  
            if dataFrame[i][j]==1:  
                col.append(dataFrame['reviews.text'][j].count(c))  
            else:  
                col.append(0)  
        dataFrame['count_'+c]=col  
  
    # drop the old feature and join the rating column that we have dropped  
    dataFrame = dataFrame.drop(features,axis=1)  
    dataFrame = dataFrame.join(d)  
    return dataFrame
```



- Node and Tree classes:

For building the Decision Tree, Node and Tree classes were implemented to ease the traversing through the Decision Tree and storing the values.

```
class Node:

    def __init__(self, key):
        self.val = key
        self.left = None
        self.right = None
    def set_val(self, _val):
        self.val=_val
    def get_val(self):
        return self.val
    def set_left(self, _left):
        self.left=_left
    def get_left(self):
        return self.left
    def set_right(self, _right):
        self.right=_right
    def get_right(self):
        return self.right

    #@dispatch(Node, Node)
    def __eq__(self, other):
        if not isinstance(other, Node):
            # Don't recognise "other", so let *it* decide if we're equal
            return NotImplemented
        if(self.val== other.val):
            return True
        else:
            return False
```

```
class Tree:
    def __init__(self, _val = None):
        self.root = Node(_val)
        self.right, self.left = None, None
    def insert_right(self, right_node):
        if not isinstance(right_node, Tree):
            self.root.right = right_node
        else:
            self.root.right = right_node.root
    def insert_left(self, left_node):
        if not isinstance(left_node, Tree):
            self.root.left = left_node
        else:
            self.root.left = left_node.root
    #@dispatch(Tree, Tree)
    def __eq__(self, other):
        if not isinstance(other, Tree):
            # Don't recognise "other", so let *it* decide if we're equal
            return NotImplemented
        if (self.root.val == other.root.val):
            return True
        else:
            return False
```

- Helping functions:

- 1) "Is_pure(data) " this function takes the train data and check if the all data has the same rating (positive or negative) and return true or false

```
def Is_pure(data):
    rating_column = data[:, -1]
    # saves the different classes to check if all the ratings are the same or not
    unique_classes = np.unique(rating_column)

    # if there is only one unique class so the data is pure
    if len(unique_classes) == 1:
        return True
    else:
        return False
```


- 2) “Data_classification(data)” this function takes the data and return the rating of the majority class

```
def Data_classification(data):  
  
    rating_column = data[:, -1]  
  
    #save the unique classes and it's count  
    unique_classes, counts_unique_classes = np.unique(rating_column, return_counts=True)  
  
    #find the index that has the max count of unique classes  
    index = counts_unique_classes.argmax()  
    classification = unique_classes[index]  
  
    return classification
```

- 3) “get_splits(data)” this function takes the data and return all the points that we can split on

```
def get_splits(data):  
  
    splits = {}  
    n_columns = data.shape[1]  
    for column_index in range(n_columns - 1): # excluding the last column which is the rating  
        splits[column_index] = []  
        values = data[:, column_index] # values of the features  
        unique_vs = np.unique(values) # extracting the unique values only  
  
        splits[column_index]=unique_vs #setting the splits values  
  
    return splits
```

- 4) “splitting(data, split_column, split_value)” this function takes which column I chose to split on and the value of this point ,the function returns the data below the split line and the data above the split line.

```
def splitting(data, split_column, split_value):  
  
    split_values = data[:, split_column]  
  
    data_below = data[split_values <= split_value]  
    data_above = data[split_values > split_value]  
  
    return data_below, data_above
```

- 5) “entropy_calculation(data):” this function takes the data and calculate its entropy (disorder metric for helping the decision tree decide which feature better to split on to reduce its depth/height), which requires calculating the probability of this data to calculate it using this equation “entropy = sum(probabilities * -np.log2(probabilities))”

```
def entropy_calculation(data):  
    rating_column = data[:, -1]  
  
    #counts how often the class appear  
    _, counts = np.unique(rating_column, return_counts=True)  
  
    #calculate the probability  
    probabilities = counts / counts.sum()  
    entropy = sum(probabilities * -np.log2(probabilities)) #calculate entropy  
  
    return entropy
```

- 6) “total_entropy(data_below, data_above)” this function takes the data above and data below - the outputs of the function “splitting(data, split_column, split_value)” - and calculate the overall entropy by calculating the probability of the data below and data above and then calculate the overall entropy by the equation “total_entropy = (probability of data below * entropy of data below + probability of data above * entropy of data above)” ..

we calculate the entropy of the data above and data below by just calling the function “entropy_calculation(data):”

```
def total_entropy(data_below, data_above):  
    n = len(data_below) + len(data_above)  
  
    #calculate probability of the data above and data below  
    probability_data_below = len(data_below) / n  
    probability_data_above = len(data_above) / n  
  
    #calculate total entropy  
    total_entropy = (probability_data_below * entropy_calculation(data_below)  
                    + probability_data_above * entropy_calculation(data_above))  
  
    return total_entropy
```



- 7) “best_split(data, splits):” this function takes the data and the splits (the points that we can split on) and return the best point can split on (its column and its value)

```
def best_split(data, splits):  
    total_entropy = 9999  
  
    #Looping over all the splits and calculate the entropy at every split point  
    for column_index in splits:  
        for value in splits[column_index]:  
            data_below, data_above = splitting(data, split_column=column_index, split_value=value)  
            current_total_entropy = total_entropy(data_below, data_above)  
  
            #find which point has the lowest entropy  
            if current_total_entropy <= total_entropy:  
                total_entropy = current_total_entropy  
                best_split_column = column_index  
                best_split_value = value  
  
    return best_split_column, best_split_value
```

- 8) “decision_tree(df, counter=0, min_samples=2, max_depth=5)”: this function is divided into two main parts: the base case and the recursive part.

The input data will enter the recursive part until it reaches the leaf of the tree then it enters the base case part and return the trained tree.

```
def decision_tree(df, counter=0, min_samples=2, max_depth=5):  
  
    # data preparations  
    if counter == 0:  
        global COLUMN_HEADERS  
        COLUMN_HEADERS = df.columns  
        data = df.values  
    else:  
        data = df  
  
    # base cases  
    if (Is_pure(data)) or (len(data) < min_samples) or (counter == max_depth):  
        classification = Data_classification(data)  
        return Node(classification)  
  
    # recursive part  
    else:  
        counter += 1  
        # helper functions  
        potential_splits = get_splits(data)  
        split_column, split_value = best_split(data, potential_splits)  
        data_below, data_above = splitting(data, split_column, split_value)  
  
        if len(data_below)==0 or len(data_above)==0:  
            classification = Data_classification(data)  
            return Node(classification)  
  
        # instantiate sub-tree --to be converted into nodes  
        feature_name = COLUMN_HEADERS[split_column]  
        question = "{} = {}".format(feature_name, split_value)  
        sub_tree = Tree(question)  
  
        yes_answer= decision_tree(data_below, counter, min_samples, max_depth)  
        no_answer = decision_tree(data_above, counter, min_samples, max_depth)  
  
        if yes_answer == no_answer:  
            sub_tree = yes_answer  
        else:  
            sub_tree.insert_left(yes_answer)  
            sub_tree.insert_right(no_answer)  
        return sub_tree
```

- Decision Tree visualisation:

“draw_graph(root):” a function built for providing a real representation for the output tree, it takes the root of the tree and draws the visualisation of the trained Decision Tree as a hierarchical graph.

```
def draw_graph(root):
    global d
    # make the id of node in graph unique by getting the object id of the node and assign to it
    root_id = str(id(root))
    root_val = root.val
    # get the left node value and id
    yes_node = root.left
    yes_id = str(id(yes_node))
    yes_val = yes_node.val
    # get the right node value and id
    no_node = root.right
    no_id = str(id(no_node))
    no_val = no_node.val
    # add node with root of sub tree to graph
    d.node(root_id, root_val, style='filled', color="#FFC300", shape='rectangle')
    # if the yes_node was classification node add it to graph and don't recurse
    # create edge between the yes node and the root of subtree
    if yes_node.left == yes_node.right == None:
        d.node(yes_id, yes_val, style='filled', color= '#E50A0A' if yes_val == 'Negative' else '#10C160')
        d.edge(root_id, yes_id, label='True')
    # if node was a question (root to another tree) create node and edge , continue recursion
    else:
        d.node(yes_id, yes_val, style='filled', color= "#FFC300", shape='rectangle')
        d.edge(root_id, yes_id, label='True')
        draw_graph(yes_node)
    # do the same for no_node
    if no_node.left == no_node.right == None:
        d.node(no_id, no_val, style='filled', color= '#E50A0A' if no_val == 'Negative' else '#10C160')
        d.edge(root_id, no_id, label='False')
    else:
        d.node(no_id, no_val, style='filled', color= "#FFC300", shape='rectangle')
        d.edge(root_id, no_id, label='False')
        draw_graph(no_node)
    return
```

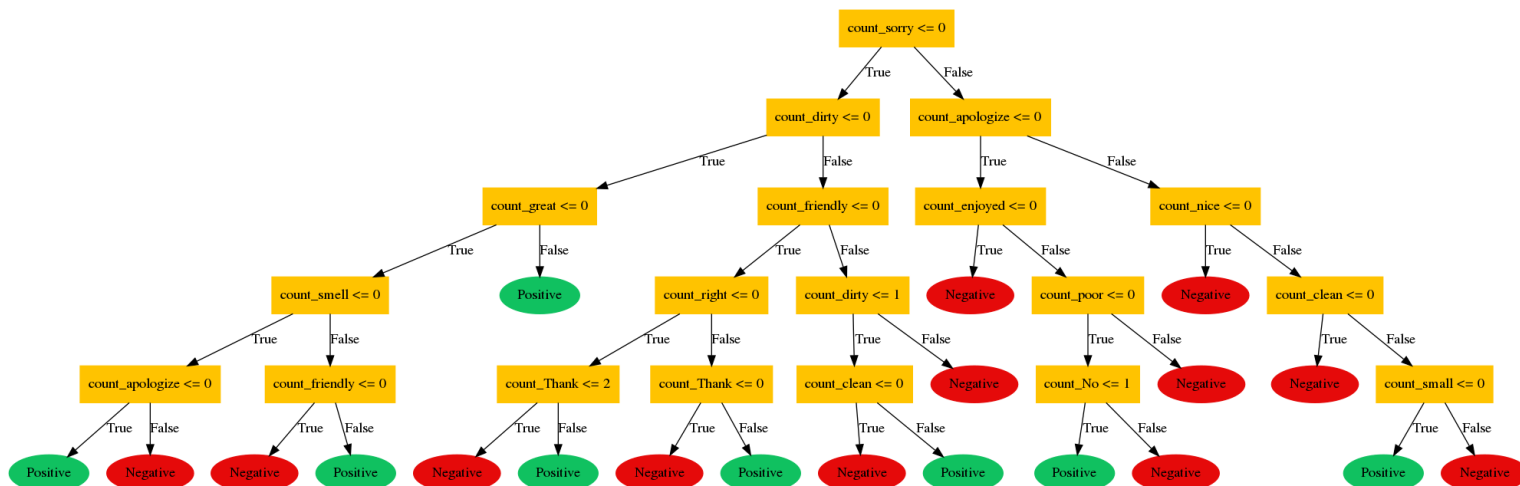


Fig. Graphical representation of a Decision Tree with depth = 5

- Predicting the rating of a review:

Function “classify_example(example, tree):” takes a row of the dataset containing a text review and return whether it is a positive or negative one.

```
def classify_example(example, root):
    question = root.val
    feature_name, comparison_operator, value = question.split()

    # ask question
    if str(example[feature_name]) <= value:
        answer = root.left
    else:
        answer = root.right

    # base case
    if answer.left==answer.right==None:
        return answer.val

    # recursive part
    else:
        residual_root = answer
        return classify_example(example, residual_root)
```

- Accuracy metric:

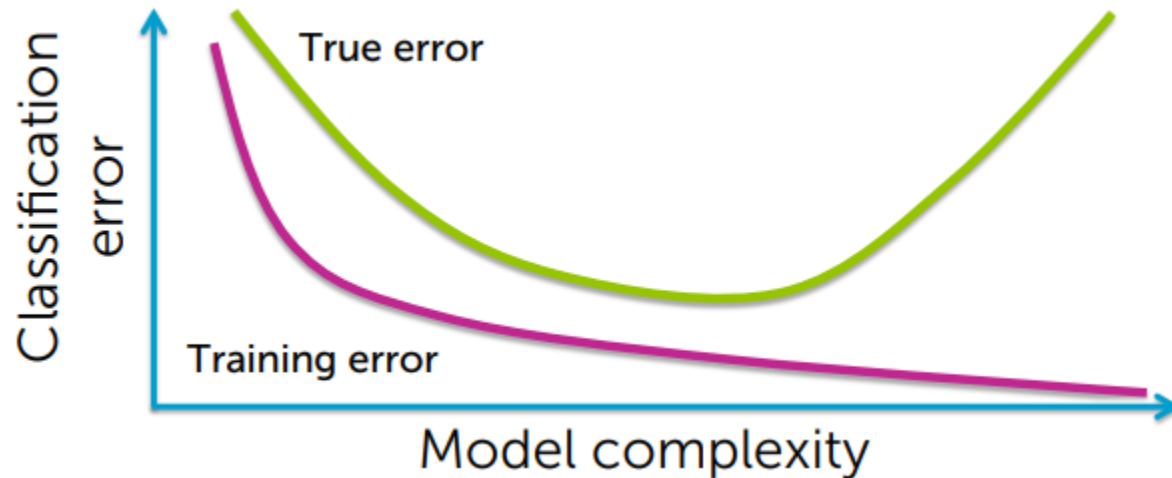
Function calculate_accuracy(df_, tree, show=False): is built to take the data as a dataframe and the trained tree, taking the column of the reviews and compare the predicted classification for each review with its real rating. This is achieved with using the train dataset for training the Decision Tree and the dev dataset for calculating the accuracy of the classifier model.

```
def calculate_accuracy(df_, tree, show=False):
    df=words_count(df_)

    #add two columns, one for the classification and the other say
    # whether the prediction is true or false
    df["classification"] = df.apply(classify_example, axis=1, args=(tree.root,))
    df["classification_correct"] = df["classification"] == df["rating"]

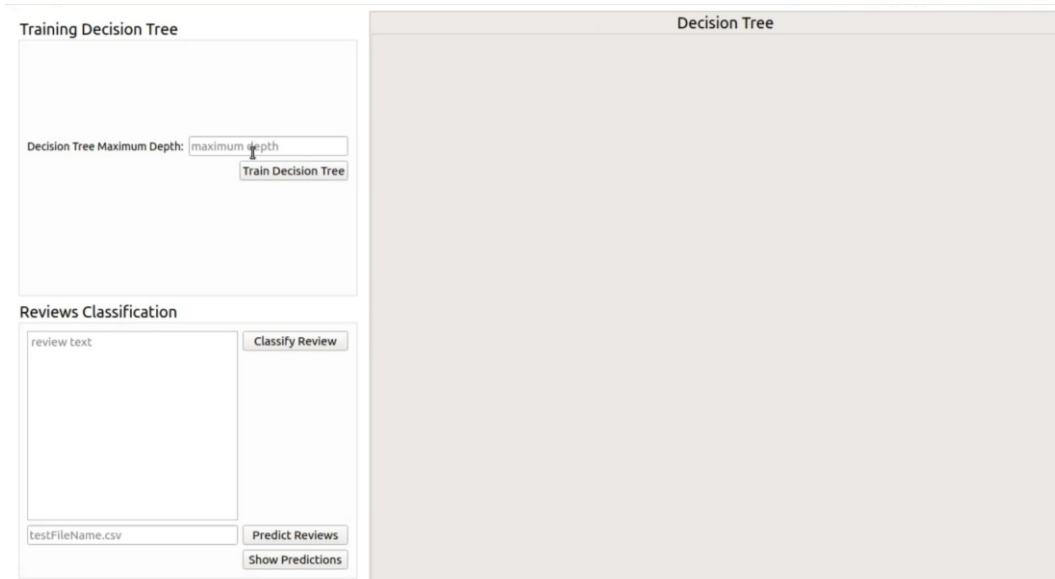
    accuracy = df["classification_correct"].mean()
    if show==True:
        print(df[['classification', 'classification_correct']])
    return accuracy
```

In our model the accuracy depends on something is called the Maximum depth of the tree, the maximum depth is described as the length of the longest path from the tree root to a leaf. When the maximum depth increases the accuracy increases but at a certain depth overfitting occurs. over-fitting is the phenomenon in which the learning system tightly fits the given training data so much that it would be inaccurate in predicting the outcomes of an untrained data.



For The Graphical User Interface:

A Graphical user interface - GUI - is designed to make it easier for the user to interact with the built model by prompting the user to specify the desired depth for the to-be-trained tree.



The screenshot shows a GUI with two main sections. The 'Training Decision Tree' section includes a text input field for 'Decision Tree Maximum Depth' with the value 'maximum depth' and a 'Train Decision Tree' button. The 'Reviews Classification' section includes a text area for 'review text', a 'Classify Review' button, a text input field for 'testFileName.csv', and 'Predict Reviews' and 'Show Predictions' buttons. A large, empty rectangular area on the right is labeled 'Decision Tree' at the top.

After the Decision Tree is trained, a colourful visualisation for the built tree appears for the user on the right side of the screen as shown:

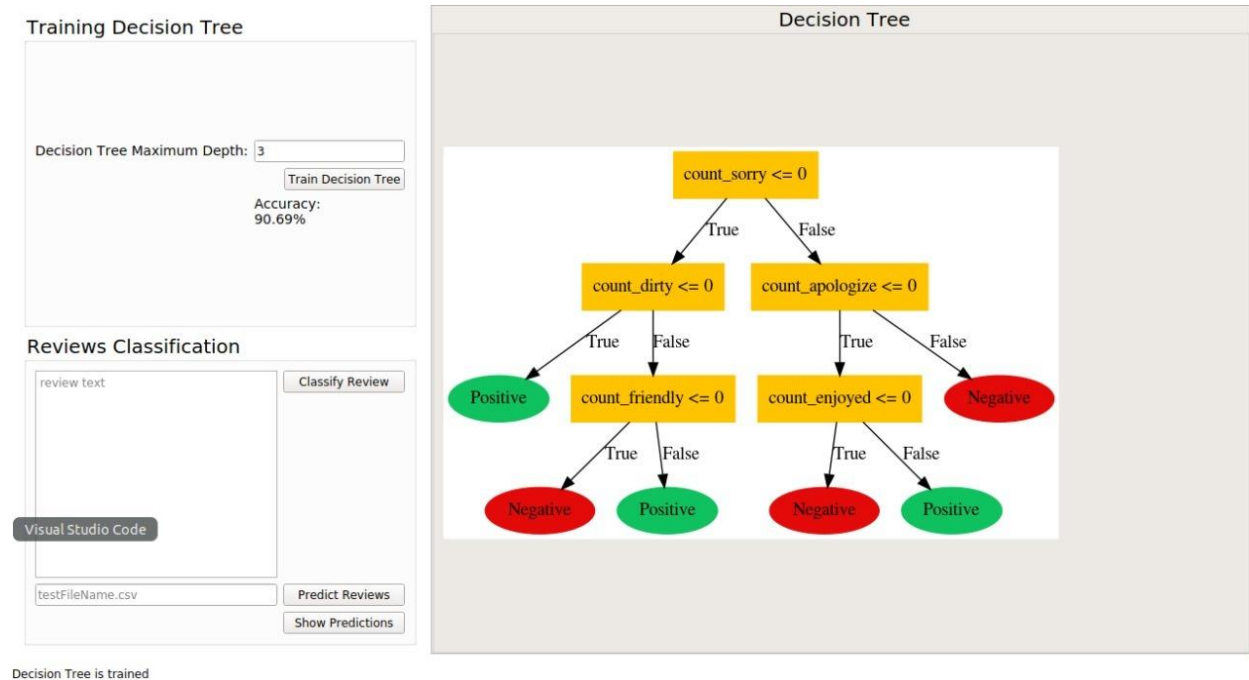


Fig. Choosing Decision Tree depth = 3 and tree visualization on the right side of the screen

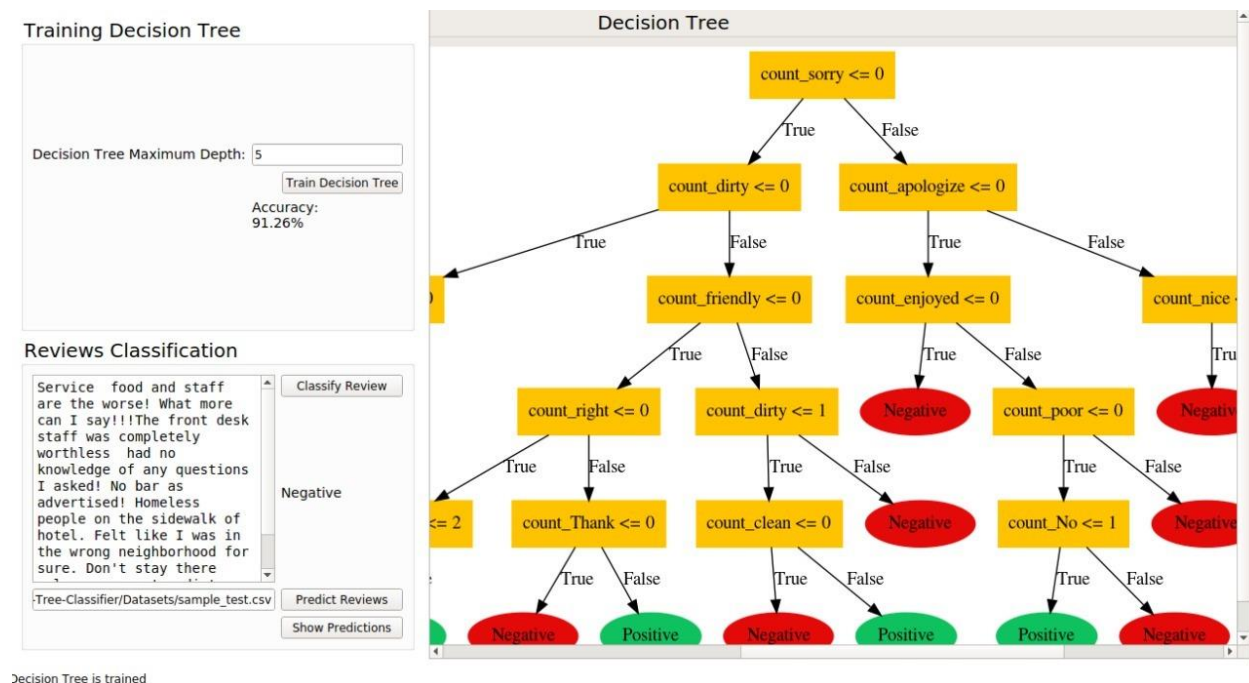


Fig. Choosing Decision Tree depth = 5 and tree visualization on the right side of the screen

Obviously, increasing the Decision Tree depth to 5 increased the model accuracy.

The user is also allowed to either:

- 1) Enter a review in text in a text edit box and classify it

Reviews Classification

Service food and staff are the worse! What more can I say!!!The front desk staff was completely worthless had no knowledge of any questions I asked! No bar as advertised! Homeless people on the sidewalk of hotel. Felt like I was in the wrong neighborhood for sure. Don't stay there

Classify Review

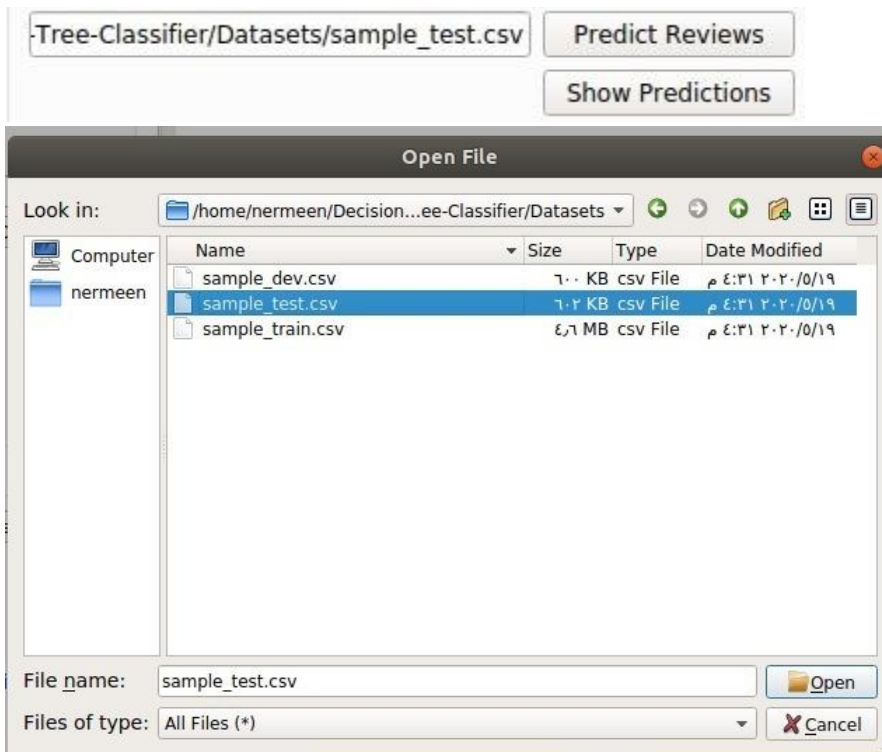
Reviews Classification

Service food and staff are the worse! What more can I say!!!The front desk staff was completely worthless had no knowledge of any questions I asked! No bar as advertised! Homeless people on the sidewalk of hotel. Felt like I was in the wrong neighborhood for sure. Don't stay there

Classify Review

Negative

- 2) Browse a csv file for the test dataset



And by clicking show predictions, the output of the test dataset is a table of two columns (text review – review prediction) appears as shown:

	Reviews Text	Prediction
1	The staff was wonderful. Diane at the front desk Ms. Yulonda in housekeeping and the gentleman cooking breakfast were so great in making my family and I feel welcomed. I will definitely return to this hotel. Dear Calvin K Thanks for all of your feedback. I have shared your review with the three employees that you recognized and with entire staff to show them how important everyone is in creating a great guest experience. We look forward to serving you again in the future.Tremeil BooneFront...	Positive
2	Have stayed at this hotel before and have never been disappointed it is a great value for the price and very close to Sea World which is where we went. No problems of any kind. The hotel was full so the breakfast area was crowded but the food was still good and for the most part was replenished when an item had run out. Will definitely...	Positive
3	Chain motels can be a hit or miss. This was a definite hit. I had specifically requested a room with a refrigerator as I had to keep medication refrigerated. Upon arrival the assigned room did not have a refrigerator. The	Positive

Back

After presenting the resulted predictions the user can retrain the Decision Tree or enter a new text review or select a new test dataset file by clicking on the **Back** button.

03

Third Topic

Complexity Of Operations

The complexity of the used functions in the project:

`data_init(file)` , `Is_pure(data)` , `Data_classification(data)` , $\rightarrow O(1)$

`words_count(dataFrame)` $\rightarrow O(c*r)$ where c is the number of columns, r is the number of rows.

`get_splits(data)` $\rightarrow O(n)$ where n is the number of columns.

`splitting(data, split_column, split_value)` $\rightarrow O(1)$

`entropy_calculation(data)` $\rightarrow O(1)$

`total_entropy(data_below, data_above)` $\rightarrow O(1)$

`best_split(data, splits)` $\rightarrow O(n^2)$ where n is the number of splits.

All Functions of class Node $\rightarrow O(1)$

All Functions Of class Tree $\rightarrow O(1)$

`decision_tree(df, counter=0, min_samples=2, max_depth=5)` $\rightarrow O(2^n)$ where n is the max depth + 1

`classify_example(example, root)` $\rightarrow O(d)$.. d is the max depth of the tree.

`calculate_accuracy(df_, tree, show=False)` $\rightarrow O(c*r)$ where c is the number of columns, r is the number of rows.

`draw_graph(root)` $\rightarrow O(2^n)$ where n is the max depth + 1



04

Fourth Topic

References

- 1) Machine Learning Refined : http://www.r-5.org/files/books/computers/algolist/statistics/data-mining/Jeremy_Watt-Machine_Learning_Refined-EN.pdf
- 2) Machine Learning: Classification Course by University of Washington: <https://www.coursera.org/learn/ml-classification>