**Final Year B. Tech. (CSE) – I: 2022-23**

**5CS462: PE5 - Data Mining Lab**

**Assignment No. 4**

**PRN: 2019BTECS00077 Date:01 Sep 2022**

**Full name: Biradar Avinash Vishnu**

**Title: Rule Base Classifier**

**Objective: Extracting Rules from decision Tree**

**Dataset Use: Car evaluation, Brest Cancer, Balance Scale**

**Introduction & Theory:**

**Rule Base:**

Rule-based classifier makes use of a set of IF-THEN rules for classification.

**Points to remember −**

* The IF part of the rule is called **rule antecedent** or **precondition**.
* The THEN part of the rule is called **rule consequent**.
* The antecedent part the condition consist of one or more attribute tests and these tests are logically ANDed.
* The consequent part consists of class prediction.

## Rule Extraction

Here we will learn how to build a rule-based classifier by extracting IF-THEN rules from a decision tree.

**Points to remember −**

To extract a rule from a decision tree −

* One rule is created for each path from the root to the leaf node.
* To form a rule antecedent, each splitting criterion is logically ANDed.
* The leaf node holds the class prediction, forming the rule consequen

**Measures of Coverage and Accuracy**

**Coverage** of a rule:

* Fraction of all records that satisfy the antecedent of a rule
* Count(instances with antecedent) / Count(training set)
* Example on left: ***(Status = 'Single') -> no***, Coverage = 4/10 = 40%

**Accuracy** of a rule:

* Fraction of records that satisfy the antecedent that also satisfy the consequent of a rule
* Count (instances with antecedent AND consequent) / Count(instances with antecedent)
* Example on left: ***(Status = 'Single') -> no***, accuracy = 2/4 = 50%

**Code:**

        st.header("Rule Base Classifier")

        # get the text representation

        text\_representation = tree.export\_text(clf,feature\_names=features)

        st.text(text\_representation)

        #Extract Code Rules

        st.subheader("Extract Code Rules")

        def tree\_to\_code(tree, feature\_names):

            tree\_ = tree.tree\_

            feature\_name = [

                feature\_names[i] if i != \_tree.TREE\_UNDEFINED else "undefined!"

                for i in tree\_.feature

            ]

            feature\_names = [f.replace(" ", "\_")[:-5] for f in feature\_names]

            st.text("def predict({}):".format(", ".join(feature\_names)))

            def recurse(node, depth):

                indent = "    " \* depth

                if tree\_.feature[node] != \_tree.TREE\_UNDEFINED:

                    name = feature\_name[node]

                    threshold = tree\_.threshold[node]

                    st.text("{}if {} <= {}:".format(indent, name, np.round(threshold,2)))

                    recurse(tree\_.children\_left[node], depth + 1)

                    st.text("{}else:  # if {} > {}".format(indent, name, np.round(threshold,2)))

                    recurse(tree\_.children\_right[node], depth + 1)

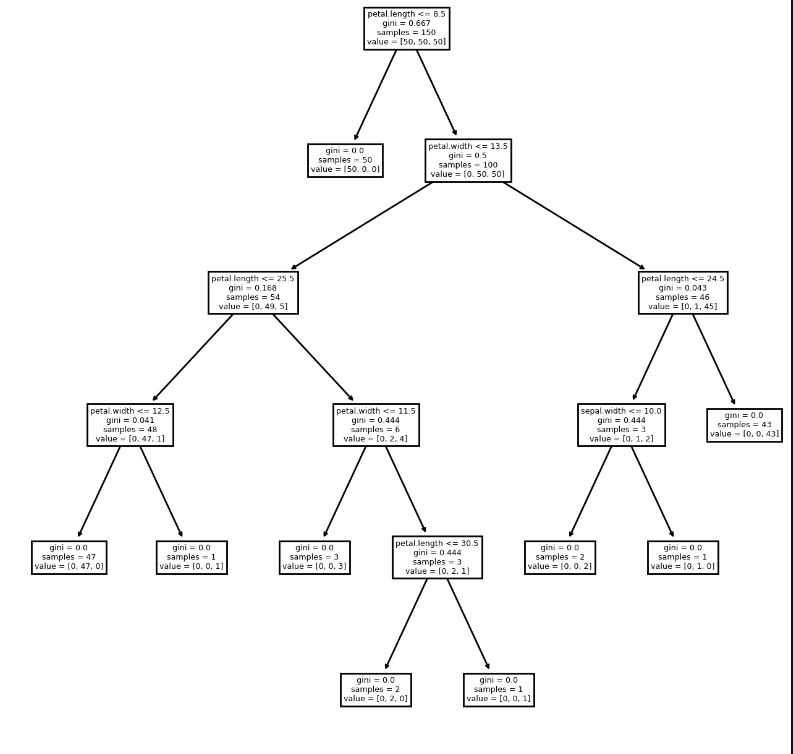
                else:

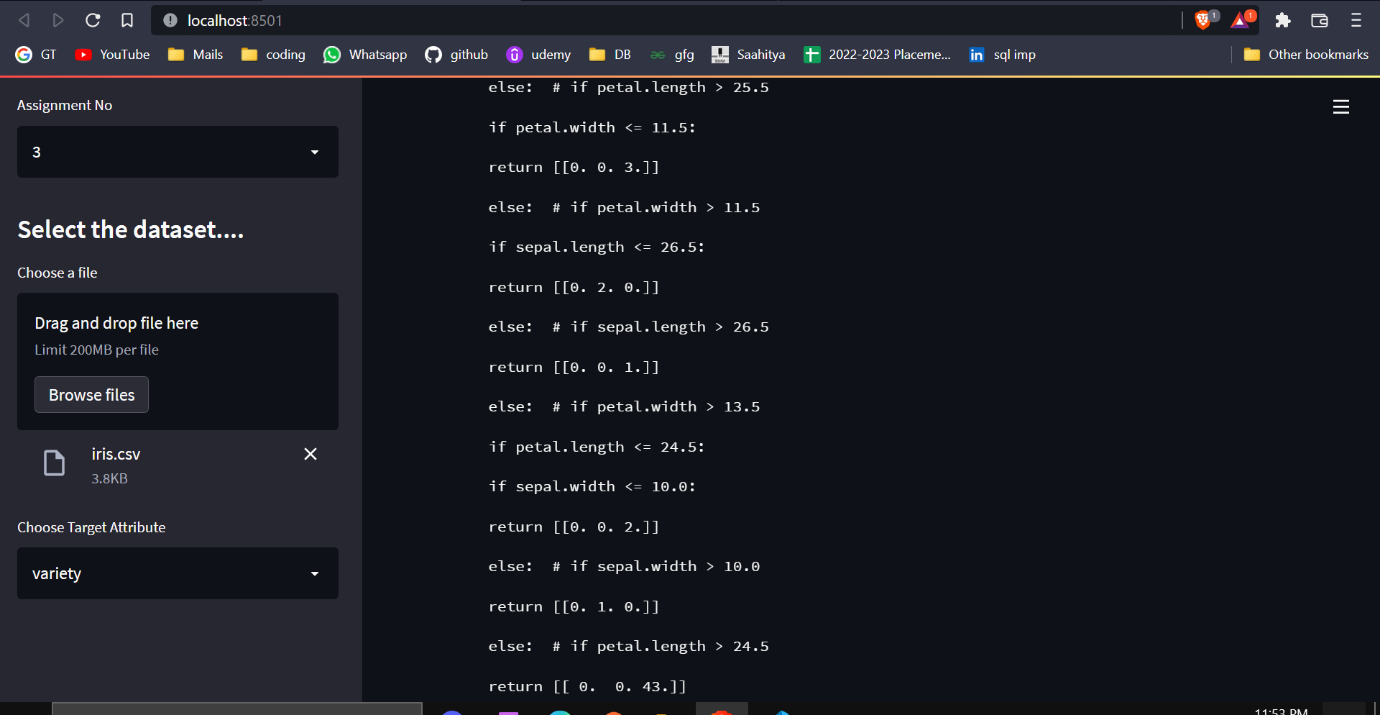
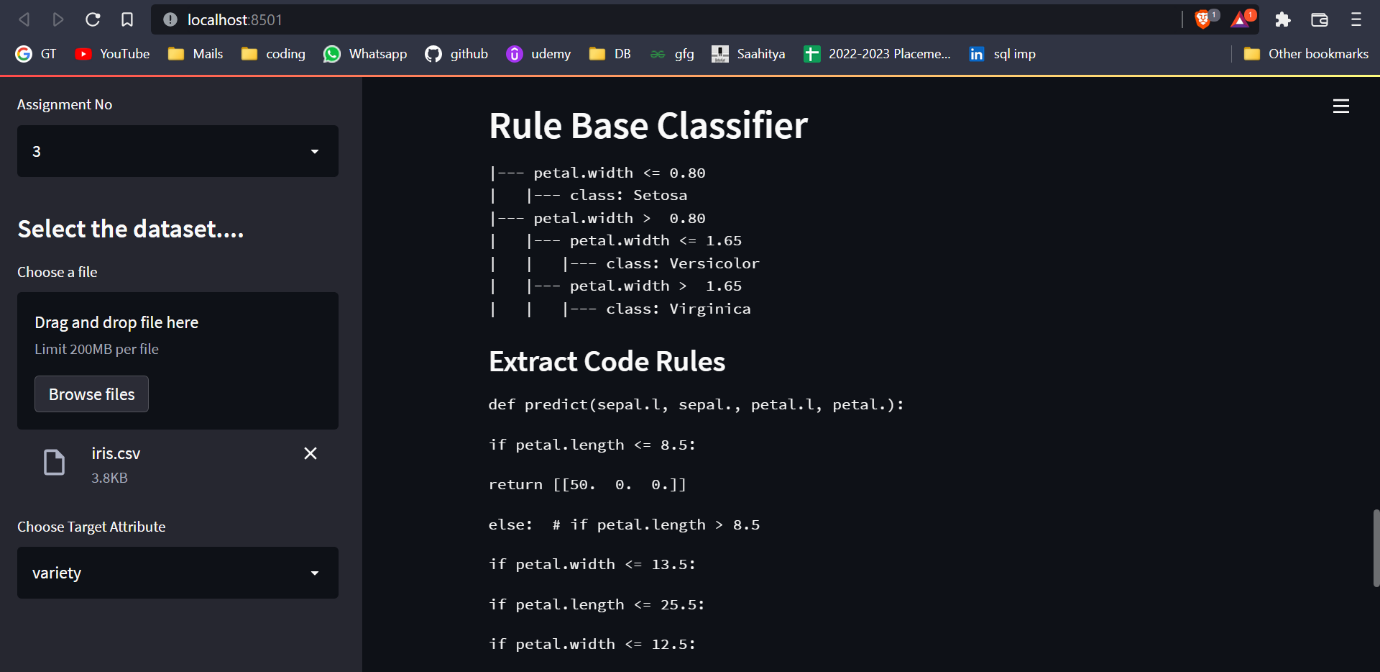
                    st.text("{}return {}".format(indent, tree\_.value[node]))

            recurse(0, 1)

        tree\_to\_code(decision\_tree,features)

**Screenshots/Output:**

****

****