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

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

**Assignment No. 3**

**PRN: 2019BTECS00077 Date:27 Aug 2022**

**Full name: Biradar Avinash Vishnu**

**Title: Decision Tree Classifier implementation and evaluation**

**Objective: Design a GUI such that shows decision tree classifier using the selection measures and graphically show/visualize Information using Gain, Gain Ratio, Gini Index and performance evaluation using confusion matrix**

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

**Introduction & Theory:**

**Decision Trees** are supervised machine learning algorithms that are best suited for classification and regression problems. These algorithms are constructed by implementing the particular splitting conditions at each node.

**What is Entropy?**

Entropy is the degree of uncertainty, impurity or disorder of a random variable, or a measure of purity. It characterizes the impurity of an arbitrary class of examples.

## ****What is Information Gain?****

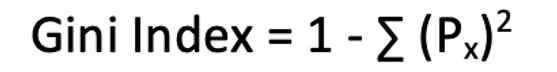
The concept of entropy plays an important role in measuring the information gain. However, “Information gain is based on the information theory”.

Information gain is used for determining the best features/attributes that render maximum information about a class. It follows the concept of entropy while aiming at decreasing the level of entropy, beginning from the root node to the leaf nodes.



**Gini Index** - Gini Index or Gini Impurity is the measurement of probability of a variable being classified wrongly when it is randomly chosen. The degree of Gini Index varies from zero to one.

Formula –



**Code:**

colums=df.columns

        targetAttr=st.sidebar.selectbox("Choose Target Attribute",colums)

        st.header("Decision Tree")

        data=df

        features = list(colums)

        features.remove(targetAttr)

        def entropy(labels):

            entropy=0

            label\_counts = Counter(labels)

            for label in label\_counts:

                prob\_of\_label = label\_counts[label] / len(labels)

                entropy -= prob\_of\_label \* math.log2(prob\_of\_label)

            return entropy

        def information\_gain(starting\_labels, split\_labels):

            info\_gain = entropy(starting\_labels)

            ans=0

            for branched\_subset in split\_labels:

                ans+=len(branched\_subset) \* entropy(branched\_subset) / len(starting\_labels)

            st.write("entropy:",ans)

            info\_gain-=ans

            return info\_gain

        def split(dataset, column):

            split\_data = []

            col\_vals = data[column].unique()

            for col\_val in col\_vals:

                split\_data.append(dataset[dataset[column] == col\_val])

            return(split\_data)

        def find\_best\_split(dataset):

            best\_gain = 0

            best\_feature = 0

            st.subheader("Overall Entropy:")

            st.write(entropy(dataset[targetAttr]))

            for feature in features:

                split\_data = split(dataset, feature)

                split\_labels = [dataframe[targetAttr] for dataframe in split\_data]

                st.subheader(feature)

                gain = information\_gain(dataset[targetAttr], split\_labels)

                st.write("Gain:",gain)

                if gain > best\_gain:

                    best\_gain, best\_feature = gain, feature

            st.subheader("Highest Gain:")

            st.write(best\_feature, best\_gain)

            return best\_feature, best\_gain

        new\_data = split(data, find\_best\_split(data)[0])

        # for i in new\_data:

        #    st.write(i)

        features = list(colums)

        features.remove(targetAttr)

        x = df[features]

        y = df[targetAttr] # Target variable

        dataEncoder = preprocessing.LabelEncoder()

        encoded\_x\_data = x.apply(dataEncoder.fit\_transform)

        st.header("1.Information Gain")

        # "leaves" (aka decision nodes) are where we get final output

        # root node is where the decision tree starts

        # Create Decision Tree classifer object

        decision\_tree = DecisionTreeClassifier(criterion="entropy")

        # Train Decision Tree Classifer

        decision\_tree = decision\_tree.fit(encoded\_x\_data, y)

        #plot decision tree

        fig, ax = plt.subplots(figsize=(6, 6))

        #figsize value changes the size of plot

        tree.plot\_tree(decision\_tree,ax=ax,feature\_names=features)

        plt.show()

        st.pyplot(plt)

        st.header("1.Gini Index")

        decision\_tree = DecisionTreeClassifier(criterion="gini")

        # Train Decision Tree Classifer

        decision\_tree = decision\_tree.fit(encoded\_x\_data, y)

        fig, ax = plt.subplots(figsize=(6, 6))

        tree.plot\_tree(decision\_tree,ax=ax,feature\_names=features)

        plt.show()

        st.pyplot(plt)

        X\_train, X\_test, y\_train, y\_test = train\_test\_split(

        x, y, test\_size=0.3, random\_state=1)

        # Create Decision Tree classifer object

        clf = DecisionTreeClassifier(max\_depth=2, random\_state=1)

        # Train Decision Tree Classifer

        clf = clf.fit(X\_train, y\_train)

        # Predict the response for test dataset

        y\_pred = clf.predict(X\_test)

        st.write("Model Accuracy: " + str(metrics.accuracy\_score(y\_test, y\_pred)))

        c\_matrix = confusion\_matrix(y\_test, y\_pred)

        print(c\_matrix)

        st.write(str(c\_matrix))

        print("True Positive:" + str(c\_matrix[0][0]))

        st.write("True Positive:" + str(c\_matrix[0][0]))

        st.text(x.columns)

        st.text(y.columns)

        plt.figure(figsize=(4, 3), dpi=150)

        # st.plotly\_chart(plot\_tree(clf, feature\_names=X.columns, filled=True))

        # st.write(plt.show(block=True))

        # st.pyplot(tree.plot\_tree(clf))

        tree = export\_graphviz(clf)

        st.graphviz\_chart(tree)

        # Tabulate the results in confusion matrix and evaluate the performance of above classifier using following metrics :

        st.write('Tabulate the results in confusion matrix and evaluate the performance of above classifier using following metrics :')

        tp = c\_matrix[1][1]

        tn = c\_matrix[2][2]

        fp = c\_matrix[1][2]

        fn = c\_matrix[2][1]

        # Recognition rate

        # precision score

        val = metrics.precision\_score(y\_test, y\_pred, average='macro')

        print('Precision score : ' + str(val))

        st.write('Precision score : ' + str(val))

        # Accuracy score

        val = metrics.accuracy\_score(y\_test, y\_pred)

        st.write('Accuracy score : ' + str(val))

**Output/Screenshots:**

