20200029
20201126
20201146
20200436

Assignment Machine learning - 2

problem 1

we started by doing some Data Preprocessing

- handling missing data by two methods:

1. imputation mode for categorical data
2. imputation means for linear data.

- encoding categorical variables.

```python
#1
data = pd.read_csv("drug.csv")
for column in data.columns:
    if data[column].dtype == 'object':
        # handle miss values in categorical value by Imputation mode / most frequent
        mode_value = data[column].mode()[0]
        data[column].fillna(mode_value, inplace=True)
        # encode categorical values
        label_encoder = LabelEncoder()
        data[column] = label_encoder.fit_transform(data[column])
        # let`s check if the null value disappeared or not
        print(column , "has number of missing values = ",data[column].isnull().sum())
    else:
        # numerical values
        # perform Imputation mean to handle miss values
        mean_value = data[column].mean()
        data[column].fillna(mean_value, inplace=True)
        # let`s check if the null value disappeared or not
        print(column , "has number of missing values = ",data[column].isnull().sum())
```

First experiment

Training and Testing with Fixed Train-Test Split Ratio 30% for Testing and 70% for Training

and repeated this experiment five times with different random splits of the data into training and test sets: 0 - 10 - 20 - 30 - 40.

```python
best_accuracy = 0
best_random_state = 0
best_model = None
features = data[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']]
target = data['Drug']
# 0 - 40 with step 10
random_state = np.arange(0,41,10)
print(random_state)
# 0 - 10 - 20 - 30 - 40
for i in random_state:
    train_data, test_data, train_labels, test_labels = train_test_split(
        features,
        target,
        test_size=0.3,
        random_state=i  # Use a different random seed for each iteration
    )
    model = DecisionTreeClassifier()
    model.fit(train_data, train_labels)
    predictions = model.predict(test_data)
    accuracy = accuracy_score(test_labels, predictions)
    if accuracy >= best_accuracy:
        best_accuracy = accuracy
        best_random_state = i
        best_model = model
    print(f"Random State: {i}  | Accuracy: {accuracy}")
print("\nBest Performing Model:")
print(f"Random State: {best_random_state} | Best Accuracy: {best_accuracy}")
```

here is the Comparison the results of different models and selecting the one that achieves the highest overall performance.

Random State	0	10	20	30	40
Accuracy	1.0	0.966	0.9833	0.9833	1.0

Second experiment

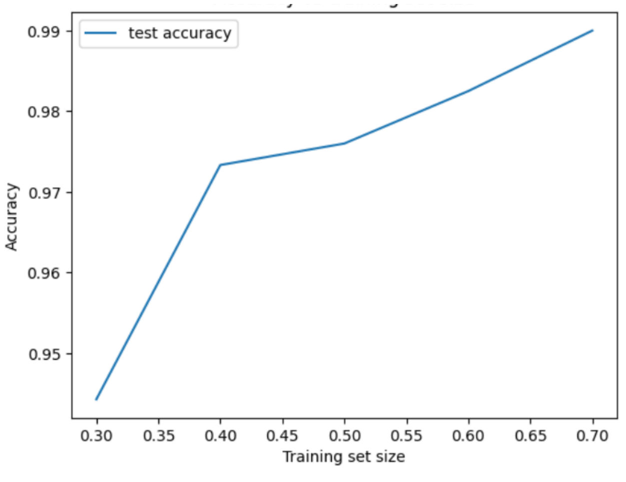
```python
accuracies = []
tree_sizes = []
test_acc = []
t_size = []
best_accuracy = 0
best_random_state = 0
best_Split_Ratios = 0
best_accuracy_for_each_Split_Ratios = 0
best_random_state_for_each_Split_Ratios = 0;
features = data[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']]
target = data['Drug']
# split ration 30 - 70 with step 10
Split_Ratios = np.arange(30,71,10)
Split_Ratios = Split_Ratios/100
random_state = np.arange(10,51,10)
# 30 - 40 - 50 - 60 - 70
for i in Split_Ratios:
    # 10 - 20 - 30 - 40 - 50
    for j in random_state:
        train_data, test_data, train_labels, test_labels = train_test_split(
            features, target, train_size=i)
        model = DecisionTreeClassifier(random_state=j)
        model.fit(train_data, train_labels)
        predictions = model.predict(test_data)
        accuracy = accuracy_score(test_labels, predictions)
        tree_size = model.tree_.node_count
        tree_sizes.append(tree_size)
        if(accuracy > best_accuracy_for_each_Split_Ratios):
            best_random_state_for_each_Split_Ratios = j
            best_accuracy_for_each_Split_Ratios = accuracy
            if(accuracy > best_accuracy):
                best_accuracy = accuracy
                best_random_state = j
                best_Split_Ratios = i
        accuracies.append(accuracy)
    test_acc.append(statistics.fmean(accuracies))
    t_size.append(int(statistics.fmean(tree_sizes)))
    print(f"for Split Ratios {i} : minimum accuarcy = {min(accuracies)}, avrage accuracy = {statistics.fmean(accuracies)}, maximum accuracy = {max(accuracies)} at random state = {best_random_state_for_each_Split_Ratios}")
    print(f"for Split Ratios {i} : minimum tree size = {min(tree_sizes)}, avrage tree size = {int(statistics.fmean(tree_sizes))}, maximum tree size = {max(tree_sizes)}")
    tree_sizes = []
    accuracies = []
    best_accuracy_for_each_Split_Ratios = 0;
    best_random_state_for_each_Split_Ratios = 0;
print(f"best accuracy = {best_accuracy}, best random state = {best_random_state}, best Split Ratios = {best_Split_Ratios}")
print(Split_Ratios)
print(test_acc)
print(t_size)
```

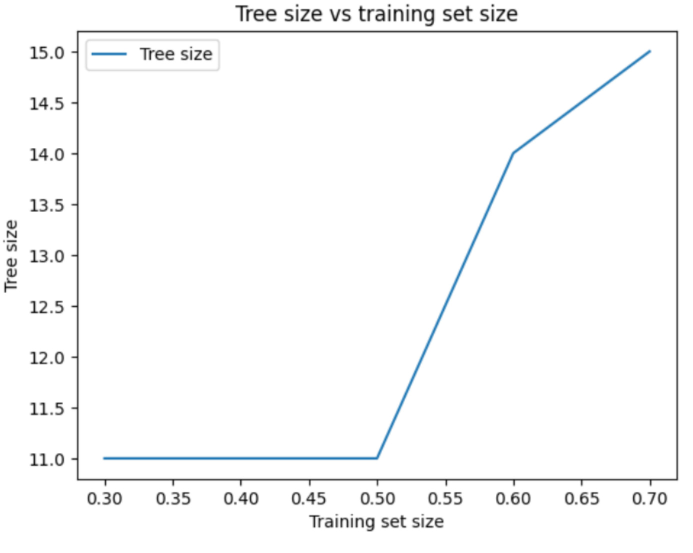
Training and Testing with a Range of Train-Test Split Ratios: training size 30 - 40 - 50 - 60 - 70, For each training set size: ran the experiment with five different random seeds 10 - 20 - 30 - 40 - 50.

Here`s the table of all statistics of each train set with its seeds:

split Ratio	minimum accuracy	average accuracy	maximum accuracy	minimum tree size	average tree size	maximum tree size
0.3	0.842	0.944	0.985	9	12	15
0.4	0.95	0.973	0.991	11	11	15
0.5	0.96	0.976	0.99	11	12	15
0.6	0.975	0.982	1.0	11	12	15
0.7	0.983	0.99	1.0	11	14	15

Here`s the plot of the relation between both of accuracy against training set size and the number of nodes in the final tree against training set size.





problem 2

First, we did some Data preprocessing.

- Normalize each feature column separately for training and test objects using Min-Max Scaling
- divide the data into 70% for training and 30% for testing.

```
data = pd.read_csv("diabetes.csv")
features = data[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI','DiabetesPedigreeFunction','Age']]
target = data['Outcome']
data=np.array(data)
np.random.shuffle(data)
data = data.T
```

```
newdata = []
for column in data:
    min_val = column.min()
    max_val = column.max()
    column = (column - min_val) / (max_val - min_val)
    newdata.append(column)
data = np.array(newdata).T
```

```
test_data, train_data = np.split(data, [int(0.3 * len(data))])
x_train = np.array(train_data.T[:-1].T)
y_train = np.array(train_data.T[-1].T)
x_test = np.array(test_data.T[:-1].T)
y_test = np.array(test_data.T[-1].T)
```

KNN

we did a model KNN for predicate the category the new data by Euclidean distance for computing distances between instances.

```
def KNN (x_test,x_train,y_train,K):
    m = y_train.shape[0]
    n = x_train.shape[1]
    distance = 0
    distances = np.zeros(m)
    result= np.zeros(K)
    for item in range(m):
        for i in range(n) :
            distance = distance + pow(x_test[i] - x_train[item][i],2)
        distance =math.sqrt(distance)
        distances[item] = distance
        distance = 0
    sorted_indices = np.argsort(distances)
    distances = distances[sorted_indices]
    y_train = y_train[sorted_indices]
    for j in range(K) :
        result[j] = y_train[j]
    result = result.astype(int)
    most_frequent_result = np.where(np.bincount(result) == np.max(np.bincount(result)))[0].tolist()
    if(np.size(most_frequent_result) == 1):
        return most_frequent_result[0]
```

for handle the tie case where the voting equals “won`t play penalty” calculate the distance between the instance and select the vote of high weight “low distances”

```
    else:
        zero_distance = 0
        one_distance = 0
        for i in range(K):
            if(y_train[i] == 0):
                zero_distance = zero_distance + distances[i]
            else:
                one_distance = one_distance + distances[i]
        if(zero_distance > one_distance):
            return 1
        else:
            return 0
```

we repeat the model 5 time with 5 different k from 1 to 5.

and store the info about the results:

```
K_values = np.arange(1,6)
n = x_test.shape[0]
avrage_Accuracy = 0
for K in K_values:
    number_correction = 0
    for i in range(n):
        if(KNN(x_test[i],x_train,y_train,K) == y_test[i]):
            number_correction = number_correction + 1
    print("K value:",K)
    print("Number of correctly classified instances:",number_correction)
    print("Total number of instances:",n)
    Accuracy = (number_correction / n) *100
    avrage_Accuracy = avrage_Accuracy + Accuracy
    print("Accuracy:",Accuracy,"%")
    print("------------------------------------------")
avrage_Accuracy = avrage_Accuracy / K_values.shape[0]
print(avrage_Accuracy)
```

K	Accuracy	Number of correctly classified instances	Total number of instances
1	71.7	165	230
2	71.7	165	230
3	75.2	173	230
4	74.7	172	230
5	73.9	170	230
Average	73.4		