VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning

Submitted by

Chaitanya Gadgil (1BM19CS223)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
May-2022 to July-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled "Machine Learning" carried out by **Chaitanya Gadgil (1BM19CS223),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Dr G R Asha   **Dr. Jyothi S Nayak**
Assistant Professor   Professor and Head
Department of CSE   Department of CSE
BMSCE, Bengaluru   BMSCE, Bengaluru

`

Index Sheet

Course Outcome

1) Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

```
In [14]: import numpy as np
         import pandas as pd
```

```
In [15]: data = pd.read_csv("finddata.csv")
         print(data,"\n")
```

```
      Time Weather Temperature Company Humidity Goes
0  Morning   Sunny        Warm     Yes     Mild  Yes
1  Evening   Rainy        Cold      No     Mild   No
2  Morning   Sunny    Moderate     Yes   Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Yes
```

```
In [19]: d = np.array(data)[:,:-1]
         print("\n The attributes are: ",d)
         target = np.array(data)[:,-1]
         print("\n The target is: ",target)
```

```
The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High']]

The target is:  ['Yes' 'No' 'Yes' 'Yes']
```

```
In [17]: def findS(c,t):
             for i, val in enumerate(t):
                 if val == "Yes":
                     specific_hypothesis = c[i].copy()
                     break

             for i, val in enumerate(c):
                 if t[i] == "Yes":
                     for x in range(len(specific_hypothesis)):
                         if val[x] != specific_hypothesis[x]:
                             specific_hypothesis[x] = '?'
                         else:
                             pass

             return specific_hypothesis
```

```
In [18]: print("\n The final hypothesis is:",findS(d,target))
```

```
The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?']
```

```
In [ ]:
```

2) For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples

```python
In [4]: import numpy as np
        import pandas as pd

        #to read the data in the csv file
        data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
        print(data,"\n")

        #making an array of all the attributes
        concepts = np.array(data.iloc[:,0:-1])
        print("The attributes are: ",concepts)

        #segregating the target that has positive and negative examples
        target = np.array(data.iloc[:,-1])
        print("\n The target is: ",target)

        #training function to implement candidate_elimination algorithm
        def learn(concepts, target):
         specific_h = concepts[0].copy()
         print("\n Initialization of specific_h and general_h")
         print(specific_h)
         general_h = [["?" for i in range(len(specific_h))] for i in
        range(len(specific_h))]
         print(general_h)
         for i, h in enumerate(concepts):
             if target[i] == "yes":
                 for x in range(len(specific_h)):
                     if h[x]!= specific_h[x]:
                         specific_h[x] ='?'
                         general_h[x][x] ='?'
                 #  print(specific_h)
             if target[i] == "no":
                 for x in range(len(specific_h)):
                     if h[x]!= specific_h[x]:
```

```python
    print(specific_h)
    print(general_h)
 indices = [i for i, val in enumerate(general_h) if val ==
['?', '?', '?', '?', '?', '?']]
 for i in indices:
     general_h.remove(['?', '?', '?', '?', '?', '?'])
 return specific_h, general_h
s_final, g_final = learn(concepts, target)

#obtaining the final hypothesis
print("\nFinal Specific_h:", s_final, sep="\n")
print("\nFinal General_h:", g_final, sep="\n")
```

```
     sky  temp humidity    wind water forcast enjoysport
0  sunny  warm  normal  strong  warm    same       yes
1  sunny  warm    high  strong  warm    same       yes
2  rainy  cold    high  strong  warm  change        no
3  sunny  warm    high  strong  cool  change       yes

The attributes are:  [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

 The target is:  ['yes' 'yes' 'no' 'yes']

 Initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

Steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

3)Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```python
In [24]: import pandas as pd
         import math
         import numpy as np
```

```python
In [34]: data = pd.read_csv("data.csv")
         features = [feat for feat in data]
         features.remove("answer")
```

```python
In [37]: class Node:
             def __init__(self):
                 self.children = []
                 self.value = ""
                 self.isLeaf = False
                 self.pred = ""
```

```python
In [38]: def entropy(examples):
             pos = 0.0
             neg = 0.0
             for _, row in examples.iterrows():
                 if row["answer"] == "yes":
                     pos += 1
                 else:
                     neg += 1
             if pos == 0.0 or neg == 0.0:
                 return 0.0
             else:
                 p = pos / (pos + neg)
                 n = neg / (pos + neg)
                 return -(p * math.log(p, 2) + n * math.log(n, 2))
```

```python
In [39]: def info_gain(examples, attr):
             uniq = np.unique(examples[attr])
             #print ("\n",uniq)
             gain = entropy(examples)
             #print ("\n",gain)
             for u in uniq:
                 subdata = examples[examples[attr] == u]
                 #print ("\n",subdata)
                 sub_e = entropy(subdata)
                 gain -= (float(len(subdata)) / float(len(examples))) * sub_e
                 #print ("\n",gain)
             return gain
```

```python
In [40]: def ID3(examples, attrs):
             root = Node()

             max_gain = 0
             max_feat = ""
             for feature in attrs:
                 #print ("\n",examples)
                 gain = info_gain(examples, feature)
                 if gain > max_gain:
                     max_gain = gain
                     max_feat = feature
             root.value = max_feat
             #print ("\nMax feature attr",max_feat)
             uniq = np.unique(examples[max_feat])
             #print ("\n",uniq)
             for u in uniq:
                 #print ("\n",u)
                 subdata = examples[examples[max_feat] == u]
                 #print ("\n",subdata)
                 if entropy(subdata) == 0.0:
                     newNode = Node()
                     newNode.isLeaf = True
                     newNode.value = u
                     newNode.pred = np.unique(subdata["answer"])
                     root.children.append(newNode)
                 else:
                     dummyNode = Node()
                     dummyNode.value = u
                     new_attrs = attrs.copy()
                     new_attrs.remove(max_feat)
                     child = ID3(subdata, new_attrs)
                     dummyNode.children.append(child)
                     root.children.append(dummyNode)
             return root
```

```python
In [41]: def printTree(root: Node, depth=0):
             for i in range(depth):
                 print("\t", end="")
             print(root.value, end="")
             if root.isLeaf:
                 print(" -> ", root.pred)
             print()
             for child in root.children:
                 printTree(child, depth + 1)
```

```python
In [42]: root = ID3(data, features)
         printTree(root)
```

```
outlook
        overcast ->  ['yes']

        rain
                wind
                        strong ->  ['no']

                        weak ->  ['yes']

        sunny
                humidity
                        high ->  ['no']

                        normal ->  ['yes']
```

4)Write a program to implement the naïve Bayesian classifier for a sample
training data set stored as a .CSV file. Compute the accuracy of the classifier,
considering few test data sets

```
In [1]: import numpy as np
        import pandas as pd
```

```
In [2]: data = pd.read_csv('/content/dataset.csv')
        data.head()
```

Out[2]:

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak

```
In [3]: y = list(data['PlayTennis'].values)
        X = data.iloc[:,1:].values
        print(f'Target Values: {y}')
        print(f'Features: \n{X}')
```

```
In [4]: y_train = y[:8]
        y_val = y[8:]
        X_train = X[:8]
        X_val = X[8:]
        print(f"Number of instances in training set: {len(X_train)}")
        print(f"Number of instances in testing set: {len(X_val)}")
```

```
Number of instances in training set: 8
Number of instances in testing set: 6
```

```python
In [5]: class NaiveBayesClassifier:
            def __init__(self, X, y):
                self.X, self.y = X, y
                self.N = len(self.X)
                self.dim = len(self.X[0])
                self.attrs = [[] for _ in range(self.dim)]
                self.output_dom = {}
                self.data = []
                for i in range(len(self.X)):
                    for j in range(self.dim):
                        if not self.X[i][j] in self.attrs[j]:
                            self.attrs[j].append(self.X[i][j])
                    if not self.y[i] in self.output_dom.keys():
                        self.output_dom[self.y[i]] = 1
                    else:
                        self.output_dom[self.y[i]] += 1
                    self.data.append([self.X[i], self.y[i]])
            def classify(self, entry):
                solve = None
                max_arg = -1
                for y in self.output_dom.keys():
                    prob = self.output_dom[y]/self.N
                    for i in range(self.dim):
                        cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                        n = len(cases)
                        prob *= n/self.N
                    if prob > max_arg:
                        max_arg = prob
                        solve = y
                return solve

In [6]: nbc = NaiveBayesClassifier(X_train, y_train)
        total_cases = len(y_val)
        good = 0
        bad = 0
        predictions = []
        for i in range(total_cases):
            predict = nbc.classify(X_val[i])
            predictions.append(predict)
            if y_val[i] == predict:
                good += 1
            else:
                bad += 1
        print('Predicted values:', predictions)
        print('Actual values:', y_val)
        print()
        print('Total number of testing instances in the dataset:', total_cases)
        print('Number of correct predictions:', good)
        print('Number of wrong predictions:', bad)
        print()
        print('Accuracy of Bayes Classifier:', good/total_cases)

        Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
        Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

        Total number of testing instances in the dataset: 6
        Number of correct predictions: 4
        Number of wrong predictions: 2

        Accuracy of Bayes Classifier: 0.6666666666666666
```

5) Implement the Linear Regression algorithm in order to fit data points.
Select appropriate data set for your experiment and draw graphs.

```
In [17]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
         from sklearn.metrics import r2_score
```

```
In [9]: dataset = pd.read_csv('salary_dataset.csv')
        X = dataset.iloc[:, :-1].values
        y = dataset.iloc[:, 1].values
```

```
In [10]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
```

```
In [11]: # Fitting Simple Linear Regression to the Training set
         from sklearn.linear_model import LinearRegression
         regressor = LinearRegression()
         regressor.fit(X_train, y_train)
```

```
Out[11]: LinearRegression()
```

```
In [15]: # Predicting the Test set results
         y_pred = regressor.predict(X_test)
         y_pred
```

```
Out[15]: array([ 40835.10590871, 123079.39940819,  65134.55626083,  63265.36777221,
                115602.64545369, 108125.8914992 , 116537.23969801,  64199.96201652,
                 76349.68719258, 100649.1375447 ])
```

```
In [18]: r2_score(y_test,y_pred)
```

```
Out[18]: 0.9749154407708353
```

```
In [19]: # Visualizing the Training set results
         viz_train = plt
         viz_train.scatter(X_train, y_train, color='red')
         viz_train.plot(X_train, regressor.predict(X_train), color='blue')
         viz_train.title('Salary VS Experience (Training set)')
         viz_train.xlabel('Year of Experience')
         viz_train.ylabel('Salary')
         viz_train.show()
```

In [14]:
```python
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```


Salary VS Experience (Test set)