Find-S

```
import pandas as pd
data = pd.read_csv('C://Users//rohit//Downloads//enjoysport.csv')
print("The total number of training instances are:", len(data), '\n',
data)
num_attribute = len(data.columns) - 1
hypothesis = ['0'] * num_attribute
print("The Initial Hypothesis is ",hypothesis)
for i in range(len(data)):
    if data.iloc[i, num_attribute] == 'yes':
        for j in range(num_attribute):
            if hypothesis[j] == '0' or hypothesis[j] == data.iloc[i,
j]:
                hypothesis[j] = data.iloc[i, j]
            else:
                hypothesis[j] = '?'
    print("\nThe hypothesis for the training instance {} is : \
n".format(i), hypothesis)

print("\nThe Maximally specific hypothesis for the training instances
is ", hypothesis)
```

```
The total number of training instances are: 4
      sky airtemp 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 Initial Hypothesis is  ['0', '0', '0', '0', '0', '0']

The hypothesis for the training instance 0 is :
 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

The hypothesis for the training instance 1 is :
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 2 is :
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 3 is :
 ['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instances is
['sunny', 'warm', '?', 'strong', '?', '?']
```

Candidate Algorithm

```
import pandas as pd
import numpy as np
data = pd.read_csv('C://Users//rohit//Downloads//enjoysport.csv')
concepts = data.iloc[:, :-1].values  # Features
target = data.iloc[:, -1].values       # Target variable
def candidate_elimination(concepts, target):
    specific_h = concepts[0].copy()  # Initialize specific hypothesis
    general_h = [["?" for _ in range(len(specific_h))] for _ in
range(len(specific_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] = '?'  # Generalize specific
hypothesis
                    general_h[x][x] = '?'  # Update general hypothesis
        else:
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
    general_h = [g for g in general_h if g != ['?' for _ in
range(len(specific_h))]]
    return specific_h, general_h
s_final, g_final = candidate_elimination(concepts, target)
print("\nFinal Specific Hypothesis:", s_final)
print("Final General Hypothesis:", g_final)


Final Specific Hypothesis: ['sunny' 'warm' '?' 'strong' '?' '?']
Final General Hypothesis: [['sunny', '?', '?', '?', '?', '?'], ['?',
'warm', '?', '?', '?', '?']]
```

Decision Tree

```
import pandas as pd
import numpy as np
class DecisionTreeID3:
    def fit(self, X, y):
        self.tree = self._build_tree(X, y)
    def _build_tree(self, X, y):
        if len(set(y)) == 1:
            return y.iloc[0]
        if X.empty:
            return y.mode()[0]  # Return the most common label if no
features left
        best_feature = self._best_feature(X, y)
        tree = {best_feature: {}}
        for value in X[best_feature].unique():
            sub_X = X[X[best_feature] ==
```

```
value].drop(columns=[best_feature])
            sub_y = y[X[best_feature] == value]
            tree[best_feature][value] = self._build_tree(sub_X, sub_y)
        return tree
    def _best_feature(self, X, y):
        base_entropy = self._entropy(y)
        return max(X.columns, key=lambda feature: base_entropy -
self._entropy_given_feature(X[feature], y))
    def _entropy(self, y):
        probabilities = y.value_counts(normalize=True)
        return -sum(probabilities * np.log2(probabilities + 1e-9))
    def _entropy_given_feature(self, feature, y):
        return sum((len(sub_y) / len(y)) * self._entropy(sub_y)
                   for value in feature.unique()
                   for sub_y in [y[feature == value]])
    def predict(self, X):
        return X.apply(self._predict_instance, axis=1)
    def _predict_instance(self, row):
        node = self.tree
        while isinstance(node, dict):
            feature = next(iter(node))
            node = node[feature].get(row[feature], None)
            if node is None:
                return None
        return node
if __name__ == "__main__":
    df = pd.read_csv("C://Users//rohit//Downloads//Tennis.csv")
    X = df.drop(columns=['Play'])
    y = df['Play']
    model = DecisionTreeID3()
    model.fit(X, y)
    print("Decision Tree Structure:")
    print(model.tree)
    test_data = pd.DataFrame({
        "Outlook": ["Sunny", "Overcast", "Rain"],
        "Temperature": ["Hot", "Mild", "Cool"],
        "Humidity": ["Normal", "Normal", "High"],
        "Wind": ["Weak", "Strong", "Strong"]
    })
    predictions = model.predict(test_data)
    print("\nPredictions:")
    print(predictions.tolist())
```

```
Decision Tree Structure:
{'Outlook': {'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}},
'Overcast': 'Yes', 'Rain': {'Wind': {'Weak': 'Yes', 'Strong': 'No'}}}}

Predictions:
['Yes', 'Yes', 'No']
```

Naive bayes accuracy

```
import pandas as pd
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
iris = datasets.load_iris()
X= iris.data
y=iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
model = GaussianNB().fit(X_train, y_train)
accuracy = accuracy_score(y_test, model.predict(X_test))
print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

```
Model Accuracy: 100.00%
```

Naive Bayes prediction

```
import pandas as pd
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score,
f1_score,classification_report
iris = datasets.load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
model = GaussianNB().fit(X_train, y_train)
y_pred = model.predict(X_test)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
report = classification_report(y_test, y_pred,
target_names=iris.target_names)
print("\nClassification Report:\n", report)
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

```
Classification Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11
```

```
    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

KNN

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
k = int(input("Enter the value of k (number of neighbors): "))
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
features_input = input("Enter sepal length, sepal width, petal length,
petal width (space-separated): ")
features = np.array([float(x) for x in
features_input.split()]).reshape(1, -1)
predicted_target = knn.predict(features)
species_names = iris.target_names[predicted_target]
print(f"The predicted species for the features {features}is:
{species_names[0]}")

Enter the value of k (number of neighbors):  5

Accuracy: 1.0

Enter sepal length, sepal width, petal length, petal width (space-
separated):  1.3 2.4 3.5 4.6

The predicted species for the features [[1.3 2.4 3.5 4.6]]is:
versicolor
```

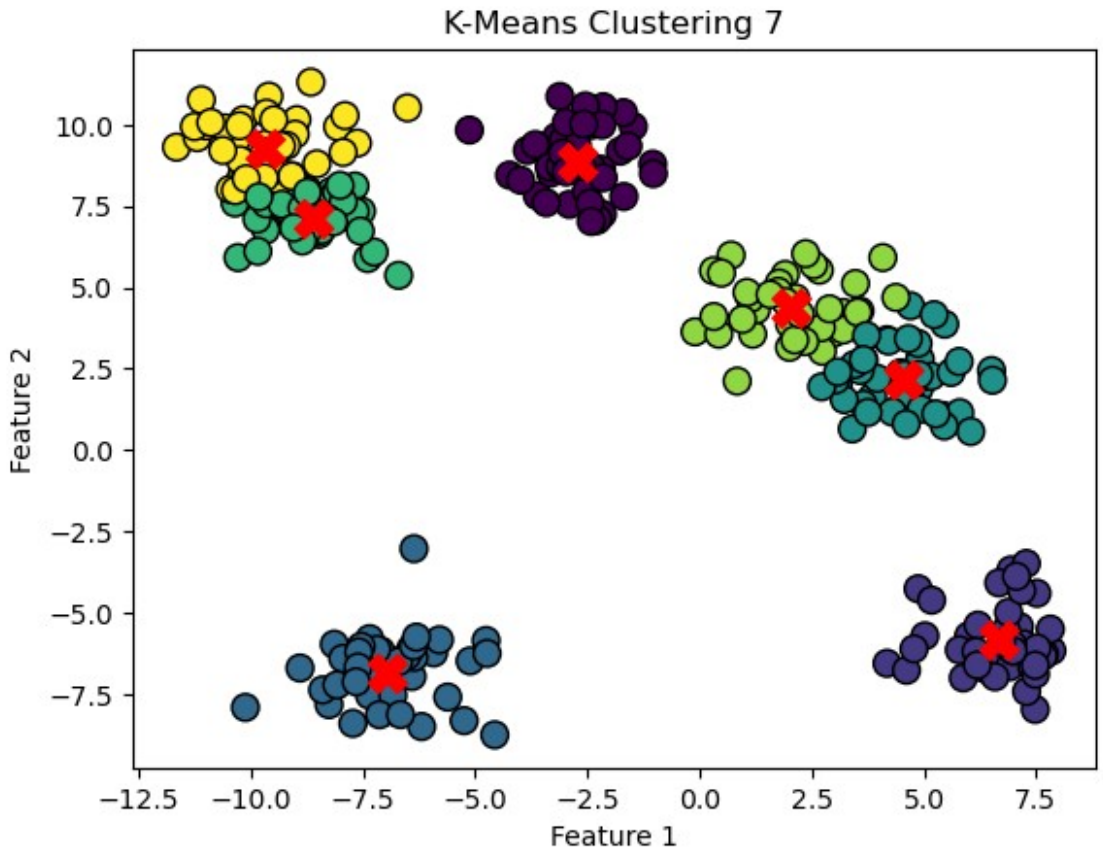
K-Means Algorithm

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
k= int(input("Enter the number of clusters (k): "))
X, _ = make_blobs(n_samples=300, centers=k, n_features=2,
random_state=42)
kmeans = KMeans(n_clusters=k, random_state=42).fit(X)
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis',
marker='o', edgecolor='k', s=100)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1], c='red', marker='X', s=200, label='Centroids')
plt.title(f'K-Means Clustering {k}')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
new_instance = np.array([[float(input("Enter feature 1: ")),
float(input("Enter feature 2: "))]])
predicted_cluster = kmeans.predict(new_instance)
print(f"The predicted cluster for the new instance is:
{predicted_cluster[0]}")
```

```
Enter the number of clusters (k):  7
```

```
C:\Users\rohit\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
C:\Users\rohit\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on
Windows with MKL, when there are less chunks than available threads.
You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
```



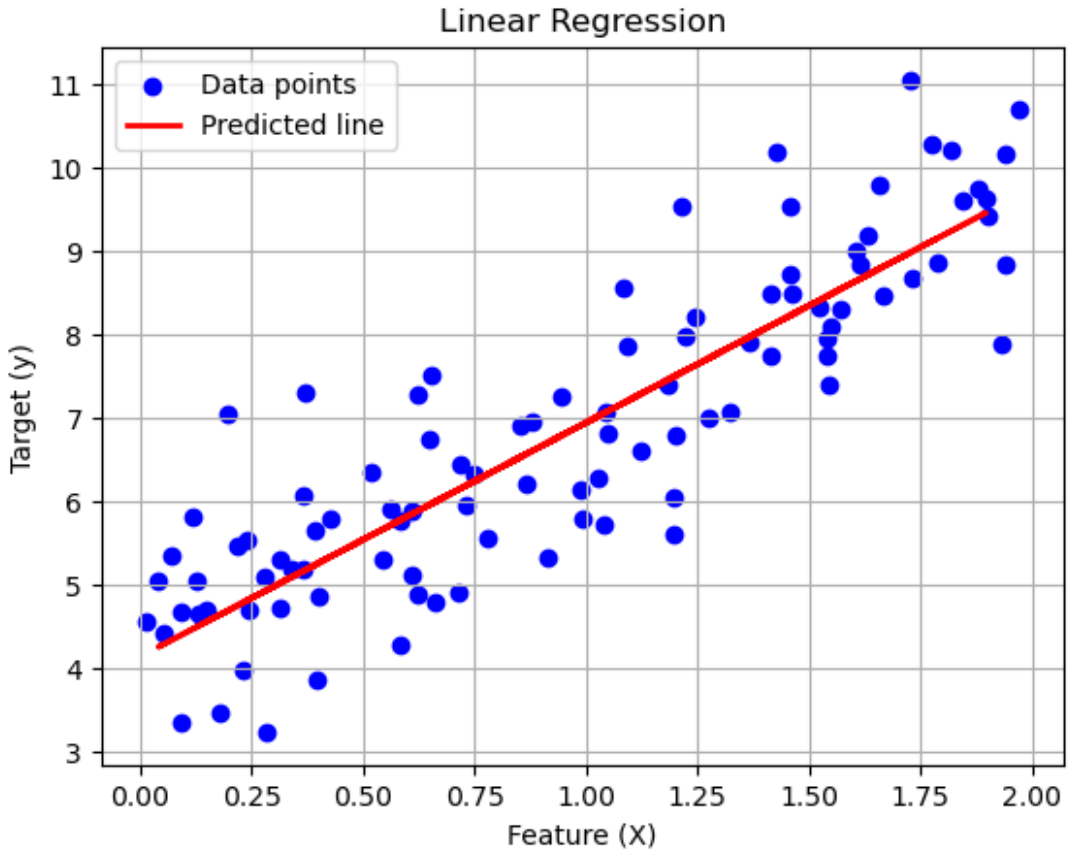

```
Enter feature 1:  -6
Enter feature 2:  4

The predicted cluster for the new instance is: 4
```

Linear Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted
line')
plt.title('Linear Regression')
```

```
plt.xlabel('Feature (X)')
plt.ylabel('Target (y)')
plt.legend()
plt.grid()
plt.show()
print(f"Intercept: {model.intercept_[0]}")
print(f"Slope: {model.coef_[0][0]}")
```




```
Intercept: 4.142913319458566
Slope: 2.7993236574802762
```