Computer Assignment 3

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

# open the txt file and read its contents
with open('ionosphere.txt', 'r', encoding='ISO-8859-1') as file:
    data = file.read()

# remove first 2 letters
data = data[2:]

# remove '\x00' from the data
data = data.replace('\x00', '')

# create a dataframe from the data
df = pd.DataFrame([row.split() for row in data.split('\n')])

# drop the row with odd index
df = df.drop(df.index[1::2])
# drop the last row
df = df.drop(df.index[-1])

# firt row is the column names
df.columns = df.iloc[0]
# drop the first row
df = df.drop(df.index[0])

column_name = [f"f{i}" for i in range(1, 35)] + ['class']

# rename the columns
df.columns = column_name

# convert the class column to binary
df['class'] = df['class'].replace(['g', 'b'], [1, 2]) # g = 1, b = 2

# convert the data type to float
df = df.astype(float)

# reset the index
df = df.reset_index(drop=True)

df.head()
```

Out[1]:

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	...	f26
0	1.0	0.0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	...	-0.51171
1	1.0	0.0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26569
2	1.0	0.0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40220
3	1.0	0.0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90695
4	1.0	0.0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65158

5 rows × 35 columns

Define variables, constants

In [2]:
```
N = len(df)
print(f"Number of samples: {N}")
```

```
Number of samples: 351
```

Feature selection

T-Test at 99% confident

In [3]:
```
# import stats
import scipy.stats as stats

def ttest(df, feature):
    """
    Perform t-test on the given feature and return the t-statistic and p-value
    """

    # group the dataframe by class
    grouped = df.groupby("class")
    # get the groups
    group1 = grouped.get_group(1)[feature]
    group2 = grouped.get_group(2)[feature]
    print(f"Group 1 Mean: {group1.mean()}")
    print(f"Group 1 Variance: {group1.var()}")
    print(f"Group 2 Mean: {group2.mean()}")
    print(f"Group 2 Variance: {group2.var()}")

    # if mean and variance are equal, then the t-statistic is 0
    # and the p-value is 1
    # so, we can return False
    if group1.mean() == group2.mean() and group1.var() == group2.var():
        print("The null hypothesis is accepted")
        print("The feature is rejected")
        return False

    # perform t-test
    # variance is unknown
    N1 = len(group1)
    N2 = len(group2)
    print(f"N1: {N1}")
    print(f"N2: {N2}")
    if N1 == N2:
        dof = 2*N1 - 2
        print("N1 == N2")
        print("Performing t-test...")
        s_squared = (group1.var() + group2.var())/(2*N1 - 2)
        s = s_squared**0.5
        q = (group1.mean() - group2.mean()) / (s * (2/N1))**0.5

        # p-value
        p_value = 1 - stats.t.cdf(abs(q), df=dof)
        print(f"p-value: {p_value}")

        # confidence interval
        ci = stats.t.ppf(0.99, dof)
```

```python
        print(f"Confidence Interval (ci): {ci}")
        print(f"q: {q}")
        print(f"q in (-ci, ci): {-ci <= q <= ci}")
        if -ci <= q <= ci:
            print("The null hypothesis is accepted")
            print("The feature is rejected")
            return False
        else:
            print("The null hypothesis is rejected")
            print("The feature is accepted")
            return True

    else:
        print("N1 != N2")
        print("Performing t-test...")
        dof = N1 + N2 - 2
        s_squared = (group1.var() + group2.var())/(N1 + N2 - 2)
        s = s_squared**0.5
        q = (group1.mean() - group2.mean()) / (s * ((1/N1) + (1/N2))**0.5)

        # p-value
        p_value = 1 - stats.t.cdf(abs(q), df=dof)
        print(f"p-value: {p_value}")

        # confidence interval
        ci = stats.t.ppf(0.99, dof)
        print(f"Confidence Interval (ci): {ci}")
        print(f"q: {q}")
        print(f"q in (-ci, ci): {-ci <= q <= ci}")
        if -ci <= q <= ci:
            print("The null hypothesis is accepted")
            print("The feature is rejected")
            return False
        else:
            print("The null hypothesis is rejected")
            print("The feature is accepted")
            return True
```

In [ ]:

```python
# calculate t-statistic and p-value for each feature
feature_accepted = []

print(f"Degrees of Freedom: {N - 2}")
print("T-Test Results\n")

for feature in df.columns[:-1]:
    print(f"Feature: {feature}")
    feature_accepted.append(ttest(df, feature))

    print()
```

In [5]:

```python
# the accepted features are
accepted_features = [feature for feature, accepted in zip(df.columns[:-1], feature_acc
print(f"Accepted Features: {accepted_features}")

print(f"Rejected Features: {[feature for feature in df.columns[:-1] if feature not in
```

```
Accepted Features: ['f1', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'f10', 'f11', 'f1
2', 'f13', 'f14', 'f15', 'f16', 'f17', 'f18', 'f19', 'f20', 'f21', 'f22', 'f23', 'f25',
'f27', 'f28', 'f29', 'f31', 'f32', 'f33', 'f34']
Rejected Features: ['f2', 'f24', 'f26', 'f30']
```

Linear classifier

In [6]:
```
class LinearClassifier:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.T = num_iterations
        self.w = None
        self.b = None
        self.labels = None

    def fit(self, X, y):
        # take the labels from y
        self.labels = np.unique(y)
        decision_boundary = (self.labels[0] + self.labels[1]) / 2

        # initialize the weights and bias to zeros
        self.w = np.zeros(X.shape[1])
        self.b = 0

        # gradient descent
        for i in range(self.T):
            # calculate the predicted values
            y_pred = np.dot(X, self.w) + self.b

            # calculate the gradients/cost function
            dw = (1/X.shape[0]) * np.dot(X.T, (y_pred - y))
            db = (1/X.shape[0]) * np.sum(y_pred - y)

            # update the weights and bias if misclassified
            self.w -= self.learning_rate * dw
            self.b -= self.learning_rate * db

            # check termination condition, if satisfied, break
            if np.linalg.norm(dw) < 1e-4:
                # print(f"Terminated at iteration {i}")
                break


    def predict(self, X):
        # calculate the predicted values
        y_pred = np.dot(X, self.w) + self.b

        # convert the predicted values to binary
        decision_boundary = (self.labels[0] + self.labels[1]) / 2
        y_pred_binary = np.where(y_pred < decision_boundary, self.labels[0], self.labe

        return y_pred_binary
```

In [7]:
```
def cross_validate(df, features, target):
    # shuffle the dataframe
    df = df.sample(frac=1).reset_index(drop=True)

    # calculate the number of samples in 10% of the dataframe
    n_samples = int(len(df) * 0.1)

    # initialize the accuracy list
    accuracy_list = []

    # loop through each split
```

```python
    for i in range(10):
        # calculate the start and end indices for the test set
        start_index = i * n_samples
        end_index = (i + 1) * n_samples

        # split the data into train and test sets
        X_test = df.iloc[start_index:end_index][features]
        X_train = pd.concat([df.iloc[:start_index][features], df.iloc[end_index:][feat
        y_train = pd.concat([df.iloc[:start_index][target], df.iloc[end_index:][target
        y_test = df.iloc[start_index:end_index][target]

        # initialize the linear classifier
        clf = LinearClassifier()

        # fit the classifier on the train set
        clf.fit(X_train, y_train)

        # predict the target values for the test set
        y_pred = clf.predict(X_test)

        # calculate the accuracy of the classifier
        accuracy = sum(y_pred == y_test) / len(y_test)

        # append the accuracy to the accuracy list
        accuracy_list.append(accuracy)

        # add other classifiers here

    # calculate the mean accuracy
    mean_accuracy = sum(accuracy_list) / len(accuracy_list)

    return mean_accuracy
```

The results

In [8]:

```python
target_column = 'class'
acc_list = []

# using all features
print("Using all features")

# collect time execution
import time
start = time.time()
accuracy = cross_validate(df, df.columns[:-1], target_column)
end = time.time()
print(f"Mean accuracy: {accuracy}\n")
acc_list.append(['all', accuracy, end-start])

# using accepted features
print("Using accepted features")
start = time.time()
accuracy = cross_validate(df, accepted_features, target_column)
end = time.time()
print(f"Mean accuracy: {accuracy}")
acc_list.append(['accepted', accuracy, end-start])
```

```
Using all features
Mean accuracy: 0.8142857142857143

Using accepted features
Mean accuracy: 0.8342857142857142
```

In [9]:

```
# using rejected features
print("Using rejected features")
rejected_features = [feature for feature in df.columns[:-1] if feature not in accepted
start = time.time()
accuracy = cross_validate(df, rejected_features, target_column)
end = time.time()
print(f"Mean accuracy: {accuracy}")
acc_list.append(['rejected', accuracy, end-start])
```

```
Using rejected features
Mean accuracy: 0.642857142857143
```

In [10]:

```
# using one feature at a time
# loop through the features and calculate the accuracy
for feature in df.columns[:-1]:
    start = time.time()
    accuracy = cross_validate(df, [feature], target_column)
    end = time.time()
    acc_list.append([feature, accuracy, end-start])
```

In [19]:

```
# create a dataframe from the accuracy list
acc_df = pd.DataFrame(acc_list, columns=['feature', 'accuracy', 'time'])
# convert accuracy to percent and round to 2 decimal places
acc_df['accuracy'] = round(acc_df['accuracy'] * 100, 2)
# convert time to 2 decimal places
acc_df['time'] = round(acc_df['time'], 2)
# show acc_df sorted by accuracy
acc_df.sort_values(by='accuracy', ascending=False)
```

Out[19]:

	feature	accuracy	time
1	accepted	83.43	29.28
7	f5	82.00	5.37
0	all	81.43	29.96
5	f3	76.86	4.84
16	f14	69.43	4.38
9	f7	68.00	3.56
10	f8	67.71	3.46
33	f31	67.14	5.51
31	f29	66.86	5.35
25	f23	66.00	3.89
11	f9	66.00	4.92
27	f25	65.14	3.97
18	f16	65.14	2.92
14	f12	64.86	4.12
6	f4	64.29	1.92
4	f2	64.29	0.04
2	rejected	64.29	5.36
32	f30	64.29	0.93
3	f1	64.29	4.85
21	f19	64.29	4.50
24	f22	64.29	5.17
35	f33	64.00	4.90
34	f32	64.00	3.41
30	f28	64.00	1.16
29	f27	64.00	4.93
28	f26	64.00	0.95
13	f11	64.00	4.09
26	f24	64.00	0.98
22	f20	64.00	3.24
20	f18	64.00	5.43
19	f17	64.00	5.39
8	f6	64.00	3.11
12	f10	64.00	5.01
36	f34	64.00	5.10
15	f13	62.00	4.43
23	f21	61.71	3.96
17	f15	61.43	5.49