

```
In [1]: from google.colab import drive  
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call  
drive.mount("/content/drive", force\_remount=True).

```
In [2]: import numpy as np  
from sklearn.model_selection import train_test_split  
from sklearn.datasets import fetch_openml  
from sklearn.linear_model import LogisticRegression  
from sklearn.metrics import accuracy_score  
import warnings  
warnings.filterwarnings('ignore')  
from sklearn.metrics.pairwise import euclidean_distances  
import pandas as pd  
from sklearn.feature_extraction.text import TfidfVectorizer  
from sklearn.datasets import fetch_openml  
import nltk  
from gensim.parsing import strip_tags, strip_numeric, strip_multiple_whitespaces  
from gensim.parsing import preprocess_string  
from gensim import parsing  
from sklearn.datasets import fetch_20newsgroups  
import re  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.preprocessing import StandardScaler  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.preprocessing import normalize  
from sklearn.decomposition import PCA
```

## Problem 1

### Logistic Regression - MNIST

```
In [3]: # Load MNIST dataset from OpenML  
mnist = fetch_openml('mnist_784', version = 1)  
  
# Split data and labels  
X = mnist.data  
y = mnist.target.astype(np.int)  
  
# Select 10000 samples  
X = X[:10000]  
y = y[:10000]  
  
# Split into train and test sets with 10,000 samples each  
X_mnist_train, X_mnist_test, y_mnist_train, y_mnist_test = train_test_split(X,
```

```
In [4]: # Train logistic regression model
clf = LogisticRegression(penalty = 'l2', solver = 'saga' ,max_iter = 100).fit(
    # Predict on test set
    y_mnist_pred = clf.predict(X_mnist_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_mnist_test, y_mnist_pred)
    print(f"Accuracy: {accuracy}")

    # Get the coefficients and sort them in descending order
    coef = clf.coef_[0]
    sorted_coef = sorted(range(len(coef)), key = lambda k: abs(coef[k]), reverse = True)

    # Print the top 30 features
    print("Top 30 features:")
    for i in range(30):
        print(f"{i+1}. Feature {sorted_coef[i]} with coefficient {coef[sorted_coef[i]]}")
```

```
Accuracy: 0.9165
Top 30 features:
1. Feature 379 with coefficient -0.0058
2. Feature 407 with coefficient -0.0058
3. Feature 434 with coefficient -0.0056
4. Feature 435 with coefficient -0.0051
5. Feature 408 with coefficient -0.0049
6. Feature 485 with coefficient 0.0048
7. Feature 380 with coefficient -0.0048
8. Feature 462 with coefficient -0.0047
9. Feature 242 with coefficient 0.0046
10. Feature 629 with coefficient 0.0045
11. Feature 324 with coefficient -0.0042
12. Feature 490 with coefficient -0.0039
13. Feature 378 with coefficient -0.0038
14. Feature 351 with coefficient -0.0037
15. Feature 517 with coefficient -0.0037
16. Feature 270 with coefficient 0.0036
17. Feature 571 with coefficient 0.0036
18. Feature 406 with coefficient -0.0035
19. Feature 512 with coefficient 0.0035
20. Feature 323 with coefficient -0.0035
21. Feature 489 with coefficient -0.0034
22. Feature 461 with coefficient -0.0033
23. Feature 383 with coefficient -0.0033
24. Feature 125 with coefficient 0.0033
25. Feature 437 with coefficient -0.0032
26. Feature 518 with coefficient -0.0032
27. Feature 454 with coefficient 0.0032
28. Feature 605 with coefficient -0.0032
29. Feature 655 with coefficient 0.0032
30. Feature 513 with coefficient 0.0031
```

## Logistic Regression - 20NG

In [5]:

```

transform_to_lower = lambda s: s.lower()
remove_emails = lambda s: re.sub(r'^[a-zA-Z0-9+_.-]+@[a-zA-Z0-9.-]+\$', '', s)
remove_single_char = lambda s: re.sub(r'\s+\w\{1\}\s+', '', s)

CLEAN_FILTERS = [remove_emails,
                  strip_tags,
                  strip_numeric,
                  remove_emails,
                  strip_punctuation,
                  strip_multiple_whitespaces,
                  transform_to_lower,
                  remove_stopwords]

def cleaningPipe(document):
    processed_words = preprocess_string(document, CLEAN_FILTERS)

    return processed_words

def joinList(processed_words):
    return ' '.join(processed_words)

def basicStemming(text):
    return parsing.stem_text(text)

newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_test = fetch_20newsgroups(subset='test')

ng_df_train = pd.DataFrame({'news' : newsgroups_train['data'], 'class' : newsgroups_train['target']})
ng_df_train['cleanedText'] = ng_df_train['news'].apply(cleaningPipe).apply(joinList)
ng_df_train.head()

```

Out[5]:

	news	class	cleanedText
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	7	lerxst wam umd edu s thing subject car nntp po...
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	4	guykuo carson u washington edu gui kuo subject...
2	From: twillis@ec.ecn.purdue.edu (Thomas E Willi...	4	twilli ec ecn purdu edu thoma e willi subject ...
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	1	jgreen amber joe green subject weitek p organ ...
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	14	jcm head cfa harvard edu jonathan mcdowell subj...

```
In [6]: ng_df_test = pd.DataFrame({"news" : newsgroups_test["data"], "class" : newsgroups_test["target"]})
ng_df_test["cleanedText"] = ng_df_test["news"].apply(cleaningPipe).apply(joinList)
ng_df_test.head()
```

Out[6]:

	news	class	cleanedText
0	From: v064mb9k@ubvmsd.cc.buffalo.edu (NEIL B. ...	7	vmbk ubvmsd cc buffalo edu neil b gandler subj...
1	From: Rick Miller <rick@ee.uwm.edu>\nSubject: ...	5	rick miller subject x face organ line distribu...
2	From: mathew <mathew@mantis.co.uk>\nSubject: R...	0	mathew subject strong weak atheism organ manti...
3	From: bakken@cs.arizona.edu (Dave Bakken)\nSub...	17	bakken cs arizona edu dave bakken subject saud...
4	From: livesey@solntze.wpd.sgi.com (Jon Livesey...	19	livesei solntze wpd sgi com jon livesei subject...

```
In [7]: num_points = 10000
```

```
ng_df_train = ng_df_train.iloc[:num_points,:]

vectorizer = TfidfVectorizer(stop_words="english")
X_train_ng = vectorizer.fit_transform(np.array(ng_df_train["cleanedText"]))

X_train = pd.DataFrame(X_train_ng.toarray())
y_train = np.array(ng_df_train["class"])
```

```
In [8]: num_points = 10000
```

```
ng_df_test = ng_df_test.iloc[:num_points,:]

X_test_ng = vectorizer.transform(np.array(ng_df_test["cleanedText"]))

X_test = pd.DataFrame(X_test_ng.toarray())
y_test = np.array(ng_df_test["class"])
```

```
In [9]: # Train logistic regression model
clf = LogisticRegression(penalty = 'l2', solver = 'lbfgs', max_iter = 100, random_state = 42)
clf.fit(X_train, y_train)

# Test model
y_pred = clf.predict(X_test)

# Test model and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)

# Get top 30 features
coef_abs = abs(clf.coef_)
top_30_idx = coef_abs.argsort()[0][-30:]
top_30_values = clf.coef_[0][top_30_idx]
print("\nTop 30 features:\n")
for idx, value in zip(top_30_idx, top_30_values):
    print(f"Feature {vectorizer.get_feature_names()[idx]}: {value}")
```

Accuracy: 0.8231545406266596

Top 30 features:

Feature halat: 1.4952560749614519  
Feature charlei: 1.498187620303281  
Feature liar: 1.503001522249867  
Feature gregg: 1.530709027405568  
Feature write: 1.554191310867334  
Feature osrh: 1.5726320652419132  
Feature argument: 1.5751121561935715  
Feature sandvik: 1.6131346654795269  
Feature cco: 1.6376258676366071  
Feature peopl: 1.6632263725661285  
Feature punish: 1.7290137321852834  
Feature schneider: 1.7354063682726226  
Feature cobb: 1.8374590217572115  
Feature wingat: 1.8388058922270343  
Feature mango: 1.903250439963916  
Feature rushdi: 1.9247682458952347  
Feature caltech: 1.980170864717595  
Feature jaeger: 2.030763767054162  
Feature benedikt: 2.038256005320304  
Feature livesei: 2.200817345898693  
Feature okcforum: 2.247556256534399  
Feature religion: 2.267245521092586  
Feature bibl: 2.3246848657883  
Feature mathew: 2.5497809991934623  
Feature moral: 2.9938806391924726  
Feature atheism: 3.2894051411807603  
Feature atheist: 3.473578464229222  
Feature islam: 3.6967976509903555  
Feature god: 3.912815829931186  
Feature keith: 4.691209911085273

## Logistic Regression - Spambase

```
In [10]: data_spam = pd.read_csv('/content/drive/MyDrive/spambase.data', header = None)
data_spam.rename(columns = {57 : 'spam'}, inplace = True)

X = data_spam.drop(['spam'], axis = 1)
y = data_spam['spam']

X = StandardScaler().fit_transform(X)

X_spam_train, X_spam_test, y_spam_train, y_spam_test=train_test_split(X, y, te
```

```
In [11]: # Train logistic regression model
clf = LogisticRegression(penalty = 'l2', solver = 'liblinear', max_iter = 100)
clf.fit(X_spam_train, y_spam_train)

# Test model
y_spam_pred = clf.predict(X_spam_test)

# Test model and print accuracy
accuracy = accuracy_score(y_spam_test, y_spam_pred)
print("\nAccuracy:", accuracy)

# Get top 30 features
coef_abs = abs(clf.coef_)
top_30_idx = coef_abs.argsort()[0][-30:]
top_30_values = clf.coef_[0][top_30_idx]
print("\nTop 30 features:\n")
for idx, value in zip(top_30_idx, top_30_values):
    print(f"Feature {vectorizer.get_feature_names()[idx]}: {value}")
```

Accuracy: 0.9370249728555917

Top 30 features:

Feature aanerud: -0.3020368558356185  
Feature aarskog: -0.3274998567327007  
Feature aaaaaggggghhh: 0.33841400348356093  
Feature aaw: 0.340839753772011  
Feature aardvark: -0.35077874329181  
Feature aaopwl: 0.37646916068842734  
Feature aantal: -0.39737363860303293  
Feature aario: -0.40394525972295603  
Feature aalborg: 0.41258999177212263  
Feature aafc: 0.5322641854280735  
Feature aaaaaaaaaaaa: 0.6269037455171719  
Feature aangegeven: -0.7110945227345816  
Feature aaron: -0.7227304025329542  
Feature aarnet: -0.772462502155514  
Feature aaltern: 0.7940853838684186  
Feature aauwpiugyvnn: 0.8042297944763976  
Feature aaah: 0.8165142496421441  
Feature aamir: -0.8334169174492623  
Feature aan: -0.9260149283382516  
Feature aaf: 0.9512190385445416  
Feature aarp: -1.069092061850569  
Feature aau: 1.1625883801605654  
Feature aaoepp: -1.2232526071749763  
Feature aarhu: -1.2613630259593502  
Feature aaronc: -1.3716497073791638  
Feature aavso: 1.4952182157129057  
Feature aav: 1.5518139906315318  
Feature aarghhhh: -1.5657406465477584  
Feature aamaz: -2.4301817188545822  
Feature aammaaaazzzzziinnnnnggggg: -4.1420385497411045

## Decision Tree Classifier - MNIST

```
In [12]: # Train decision tree model
clf = DecisionTreeClassifier()
clf.fit(X_mnist_train, y_mnist_train)

# Predict on test set
y_pred = clf.predict(X_mnist_test)

# Calculate accuracy
accuracy = accuracy_score(y_mnist_test, y_mnist_pred)
print(f"MNIST Accuracy: {accuracy}")
```

MNIST Accuracy: 0.9165

```
In [13]: # Define a list of feature names based on the word index
feature_names = [f"word_{i}" for i in range(784)]

# Extract the feature indices and thresholds for the first 30 nodes of the decision tree
node_features = clf.tree_.feature[:30]
node_thresholds = clf.tree_.threshold[:30]

# Print out the feature names and threshold values for each node
for i in range(30):
    feature_idx = node_features[i]
    feature_name = feature_names[feature_idx]
    threshold = node_thresholds[i]
    print(f"Node {i}: feature = {feature_name}, threshold = {threshold:.2f}")
```

```
Node 0: feature=word_350, threshold=148.50
Node 1: feature=word_155, threshold=0.50
Node 2: feature=word_239, threshold=0.50
Node 3: feature=word_543, threshold=78.50
Node 4: feature=word_295, threshold=119.00
Node 5: feature=word_183, threshold=162.50
Node 6: feature=word_597, threshold=15.50
Node 7: feature=word_69, threshold=14.50
Node 8: feature=word_323, threshold=141.00
Node 9: feature=word_122, threshold=232.00
Node 10: feature=word_470, threshold=252.50
Node 11: feature=word_409, threshold=10.00
Node 12: feature=word_462, threshold=162.00
Node 13: feature=word_319, threshold=141.00
Node 14: feature=word_376, threshold=88.00
Node 15: feature=word_782, threshold=-2.00
Node 16: feature=word_580, threshold=46.50
Node 17: feature=word_782, threshold=-2.00
Node 18: feature=word_782, threshold=-2.00
Node 19: feature=word_782, threshold=-2.00
Node 20: feature=word_549, threshold=243.50
Node 21: feature=word_782, threshold=-2.00
Node 22: feature=word_782, threshold=-2.00
Node 23: feature=word_210, threshold=254.00
Node 24: feature=word_154, threshold=137.50
Node 25: feature=word_782, threshold=-2.00
Node 26: feature=word_177, threshold=21.50
Node 27: feature=word_782, threshold=-2.00
Node 28: feature=word_782, threshold=-2.00
Node 29: feature=word_782, threshold=-2.00
```

## Decision Tree - 20NG

```
In [14]: # Instantiate the classifier with max_depth
dtc_5 = DecisionTreeClassifier(max_depth = 50)

# Fit the classifier on the training data
dtc_5.fit(X_train, y_train)

# Predict on the test data
y_pred_5 = dtc_5.predict(X_test)

# Compute accuracy on the test data
accuracy_5 = accuracy_score(y_test, y_pred_5)

# Instantiate the classifier with max_depth
dtc_10 = DecisionTreeClassifier(max_depth = 100)

# Fit the classifier on the training data
dtc_10.fit(X_train, y_train)

# Predict on the test data
y_pred_10 = dtc_10.predict(X_test)

# Compute accuracy on the test data
accuracy_10 = accuracy_score(y_test, y_pred_10)

print("Accuracy for Decision Tree with max_depth = 50: {:.2f}".format(accuracy_5))
print("Accuracy for Decision Tree with max_depth = 100: {:.2f}".format(accuracy_10))
```

Accuracy for Decision Tree with max\_depth = 50: 0.51  
Accuracy for Decision Tree with max\_depth = 100: 0.56

```
In [15]: # Define a list of feature names based on the pixel index in the flattened image
feature_names = [f"pixel_{i}" for i in range(X_train.shape[1])]

# Extract the feature indices and thresholds for the first 30 nodes of the decision tree
node_features = dtc_5.tree_.feature[:30]
node_thresholds = dtc_5.tree_.threshold[:30]

# Print out the feature names and threshold values for each node
for i in range(30):
    feature_idx = node_features[i]
    feature_name = feature_names[feature_idx]
    threshold = node_thresholds[i]
    print(f"Node {i}: feature = {feature_name}, threshold = {threshold:.2f}")
```

```
Node 0: feature=pixel_15425, threshold=0.00
Node 1: feature=pixel_5015, threshold=0.01
Node 2: feature=pixel_7241, threshold=0.04
Node 3: feature=pixel_58347, threshold=0.03
Node 4: feature=pixel_22714, threshold=0.00
Node 5: feature=pixel_21002, threshold=0.03
Node 6: feature=pixel_25471, threshold=0.01
Node 7: feature=pixel_8476, threshold=0.01
Node 8: feature=pixel_45958, threshold=0.05
Node 9: feature=pixel_49094, threshold=0.04
Node 10: feature=pixel_4084, threshold=0.02
Node 11: feature=pixel_31010, threshold=0.02
Node 12: feature=pixel_2594, threshold=0.01
Node 13: feature=pixel_13526, threshold=0.03
Node 14: feature=pixel_51849, threshold=0.03
Node 15: feature=pixel_8971, threshold=0.01
Node 16: feature=pixel_19321, threshold=0.02
Node 17: feature=pixel_20492, threshold=0.04
Node 18: feature=pixel_20118, threshold=0.03
Node 19: feature=pixel_8671, threshold=0.03
Node 20: feature=pixel_13517, threshold=0.01
Node 21: feature=pixel_46497, threshold=0.01
Node 22: feature=pixel_57360, threshold=0.02
Node 23: feature=pixel_38184, threshold=0.01
Node 24: feature=pixel_8956, threshold=0.01
Node 25: feature=pixel_8869, threshold=0.02
Node 26: feature=pixel_34435, threshold=0.02
Node 27: feature=pixel_40532, threshold=0.01
Node 28: feature=pixel_2245, threshold=0.05
Node 29: feature=pixel_18046, threshold=0.04
```

```
In [16]: # Define a list of feature names based on the pixel index in the flattened image
feature_names = [f"pixel_{i}" for i in range(X_train.shape[1])]

# Extract the feature indices and thresholds for the first 30 nodes of the decision tree
node_features = dtc_10.tree_.feature[:30]
node_thresholds = dtc_10.tree_.threshold[:30]

# Print out the feature names and threshold values for each node
for i in range(30):
    feature_idx = node_features[i]
    feature_name = feature_names[feature_idx]
    threshold = node_thresholds[i]
    print(f"Node {i}: feature = {feature_name}, threshold = {threshold:.2f}")
```

```
Node 0: feature=pixel_15425, threshold=0.00
Node 1: feature=pixel_5015, threshold=0.01
Node 2: feature=pixel_7241, threshold=0.04
Node 3: feature=pixel_58347, threshold=0.03
Node 4: feature=pixel_22714, threshold=0.00
Node 5: feature=pixel_21002, threshold=0.03
Node 6: feature=pixel_25471, threshold=0.01
Node 7: feature=pixel_8476, threshold=0.01
Node 8: feature=pixel_45958, threshold=0.05
Node 9: feature=pixel_49094, threshold=0.04
Node 10: feature=pixel_4084, threshold=0.02
Node 11: feature=pixel_31010, threshold=0.02
Node 12: feature=pixel_2594, threshold=0.01
Node 13: feature=pixel_13526, threshold=0.03
Node 14: feature=pixel_51849, threshold=0.03
Node 15: feature=pixel_8971, threshold=0.01
Node 16: feature=pixel_19321, threshold=0.02
Node 17: feature=pixel_20492, threshold=0.04
Node 18: feature=pixel_20118, threshold=0.03
Node 19: feature=pixel_8671, threshold=0.03
Node 20: feature=pixel_13517, threshold=0.01
Node 21: feature=pixel_46497, threshold=0.01
Node 22: feature=pixel_57360, threshold=0.02
Node 23: feature=pixel_38184, threshold=0.01
Node 24: feature=pixel_8956, threshold=0.01
Node 25: feature=pixel_8869, threshold=0.02
Node 26: feature=pixel_34435, threshold=0.02
Node 27: feature=pixel_40532, threshold=0.01
Node 28: feature=pixel_2245, threshold=0.05
Node 29: feature=pixel_18046, threshold=0.04
```

## Decision Tree - Spambase

```
In [17]: # Create a decision tree classifier
dt = DecisionTreeClassifier()

# Train the classifier on the training data
dt.fit(X_spam_train, y_spam_train)

# Make predictions on the test data
spam_test_predictions = dt.predict(X_spam_test)

# Calculate the accuracy of the classifier
spam_test_accuracy = accuracy_score(y_spam_test, spam_test_predictions)
print(f"Spambase Accuracy: {spam_test_accuracy:.4f}")
```

Spambase Accuracy: 0.9186

```
In [18]: # Define a list of feature names based on the pixel index in the flattened image
feature_names = [f"{i}" for i in range(X_spam_train.shape[1])]

# Extract the feature indices and thresholds for the first 30 nodes of the decision tree
node_features = dt.tree_.feature[:30]
node_thresholds = dt.tree_.threshold[:30]

# Print out the feature names and threshold values for each node
for i in range(30):
    feature_idx = node_features[i]
    feature_name = feature_names[feature_idx]
    threshold = node_thresholds[i]
    print(f"Node {i}: feature = {feature_name}, threshold = {threshold:.2f}")
```

```
Node 0: feature=52, threshold=-0.08
Node 1: feature=6, threshold=-0.16
Node 2: feature=51, threshold=0.10
Node 3: feature=15, threshold=-0.27
Node 4: feature=23, threshold=-0.17
Node 5: feature=54, threshold=0.09
Node 6: feature=4, threshold=1.92
Node 7: feature=27, threshold=12.96
Node 8: feature=24, threshold=-0.32
Node 9: feature=16, threshold=0.75
Node 10: feature=35, threshold=1.06
Node 11: feature=55, threshold=-0.20
Node 12: feature=7, threshold=20.92
Node 13: feature=22, threshold=5.93
Node 14: feature=19, threshold=18.90
Node 15: feature=39, threshold=2.44
Node 16: feature=18, threshold=-0.30
Node 17: feature=56, threshold=-0.25
Node 18: feature=54, threshold=-0.12
Node 19: feature=54, threshold=-0.12
Node 20: feature=56, threshold=-0.41
Node 21: feature=56, threshold=-0.46
Node 22: feature=55, threshold=-2.00
Node 23: feature=56, threshold=-0.45
Node 24: feature=26, threshold=0.28
Node 25: feature=55, threshold=-0.26
Node 26: feature=45, threshold=1.51
Node 27: feature=9, threshold=2.99
Node 28: feature=44, threshold=2.79
Node 29: feature=55, threshold=-2.00
```

## Problem 2

A)

## Logistic Regression with PCA - MNIST

```
In [19]: # PCA fit
pca = PCA(n_components = 5)
mnist_train_reduced = pca.fit_transform(X_mnist_train)
mnist_test_reduced = pca.transform(X_mnist_test)

# Train logistic regression model
clf = LogisticRegression(penalty = 'l2', solver = 'saga' ,max_iter = 100).fit()

# Predict on test set
y_mnist_pred = clf.predict(mnist_test_reduced)

# Calculate accuracy
accuracy = accuracy_score(y_mnist_test, y_mnist_pred)
print(f"MNIST Accuracy: {accuracy}")
```

MNIST Accuracy: 0.668

```
In [20]: # PCA fit
pca = PCA(n_components = 20)
mnist_train_reduced = pca.fit_transform(X_mnist_train)
mnist_test_reduced = pca.transform(X_mnist_test)

# Train logistic regression model
clf = LogisticRegression(penalty = 'l2', solver = 'saga' ,max_iter = 100).fit()

# Predict on test set
y_mnist_pred = clf.predict(mnist_test_reduced)

# Calculate accuracy
accuracy = accuracy_score(y_mnist_test, y_mnist_pred)
print(f"MNIST Accuracy: {accuracy}")
```

MNIST Accuracy: 0.872

## Decision Tree with PCA - MNIST

```
In [21]: # PCA fit
pca = PCA(n_components = 5)
mnist_train_reduced = pca.fit_transform(X_mnist_train)
mnist_test_reduced = pca.transform(X_mnist_test)

# Train decision tree model
clf = DecisionTreeClassifier()
clf.fit(mnist_train_reduced, y_mnist_train)

# Predict on test set
y_pred = clf.predict(mnist_test_reduced)

# Calculate accuracy
accuracy = accuracy_score(y_mnist_test, y_mnist_pred)
print(f"MNIST Accuracy: {accuracy}")
```

MNIST Accuracy: 0.872

```
In [22]: # PCA fit
pca = PCA(n_components = 20)
mnist_train_reduced = pca.fit_transform(X_mnist_train)
mnist_test_reduced = pca.transform(X_mnist_test)

# Train decision tree model
clf = DecisionTreeClassifier()
clf.fit(mnist_train_reduced, y_mnist_train)

# Predict on test set
y_pred = clf.predict(mnist_test_reduced)

# Calculate accuracy
accuracy = accuracy_score(y_mnist_test, y_mnist_pred)
print(f"MNIST Accuracy: {accuracy}")
```

MNIST Accuracy: 0.872

B)

## Logistic Regression with PCA - Spambase

```
In [23]: pca = PCA(n_components = 31).fit(X_spam_train, y_spam_train)
spam_train_reduced = pca.transform(X_spam_train)
spam_test_reduced = pca.transform(X_spam_test)

# Train logistic regression model
clf = LogisticRegression(solver = 'liblinear', max_iter = 100)
clf.fit(spam_train_reduced, y_spam_train)

# Test model and print confusion matrix
y_spam_pred = clf.predict(spam_test_reduced)

# Test model and print accuracy
accuracy = accuracy_score(y_spam_test, y_spam_pred)
print("\nAccuracy:", accuracy)
```

Accuracy: 0.9272529858849077

## Probelm 3

```
In [24]: # Load the MNIST dataset and extract the first 10,000 samples
mnist = fetch_openml("mnist_784")
X = mnist.data[:10000]
y = mnist.target[:10000]

# Normalize the data by subtracting the mean and dividing by the standard deviation
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)

# Calculate the covariance matrix
covariance_matrix = np.cov(X_norm.T)

# Calculate the eigenvalues and eigenvectors of the covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

# Sort the eigenvalues in descending order
idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]

# Choose the number of principal components (K)
K = 5

# Project the dataset onto the K principal components
projection_matrix = eigenvectors[:, :K]
X_pca = np.dot(X_norm, projection_matrix)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size = 0.2, random_state = 42)

# Train a Logistic regression model on the training data
clf = LogisticRegression(max_iter = 100)
clf.fit(X_train, y_train)

# Evaluate the accuracy of the model on the testing data
accuracy = clf.score(X_test, y_test)
print("MNIST PCA Accuracy:", accuracy)
```

MNIST PCA Accuracy: 0.6995

```
In [25]: import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Load the MNIST dataset and extract the first 10,000 samples
mnist = fetch_openml("mnist_784")
X = mnist.data[:10000]
y = mnist.target[:10000]

# Normalize the data by subtracting the mean and dividing by the standard deviation
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)

# Calculate the covariance matrix
covariance_matrix = np.cov(X_norm.T)

# Calculate the eigenvalues and eigenvectors of the covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

# Sort the eigenvalues in descending order
idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]

# Choose the number of principal components (K)
K = 20

# Project the dataset onto the K principal components
projection_matrix = eigenvectors[:, :K]
X_pca = np.dot(X_norm, projection_matrix)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size = 0.2, random_state=42)

# Train a Logistic regression model on the training data
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train, y_train)

# Evaluate the accuracy of the model on the testing data
accuracy = clf.score(X_test, y_test)
print("MNIST PCA Accuracy:", accuracy)
```

MNIST PCA Accuracy: 0.8835