Project Idea:

This is a simple machine learning classification project where we recognize digits in photos. We used the MINISIT handwritten digits dataset which has a training set of 60,000 examples and a test set with 10,000 photos.

This project was developed using Python and run on Google Co-lab.

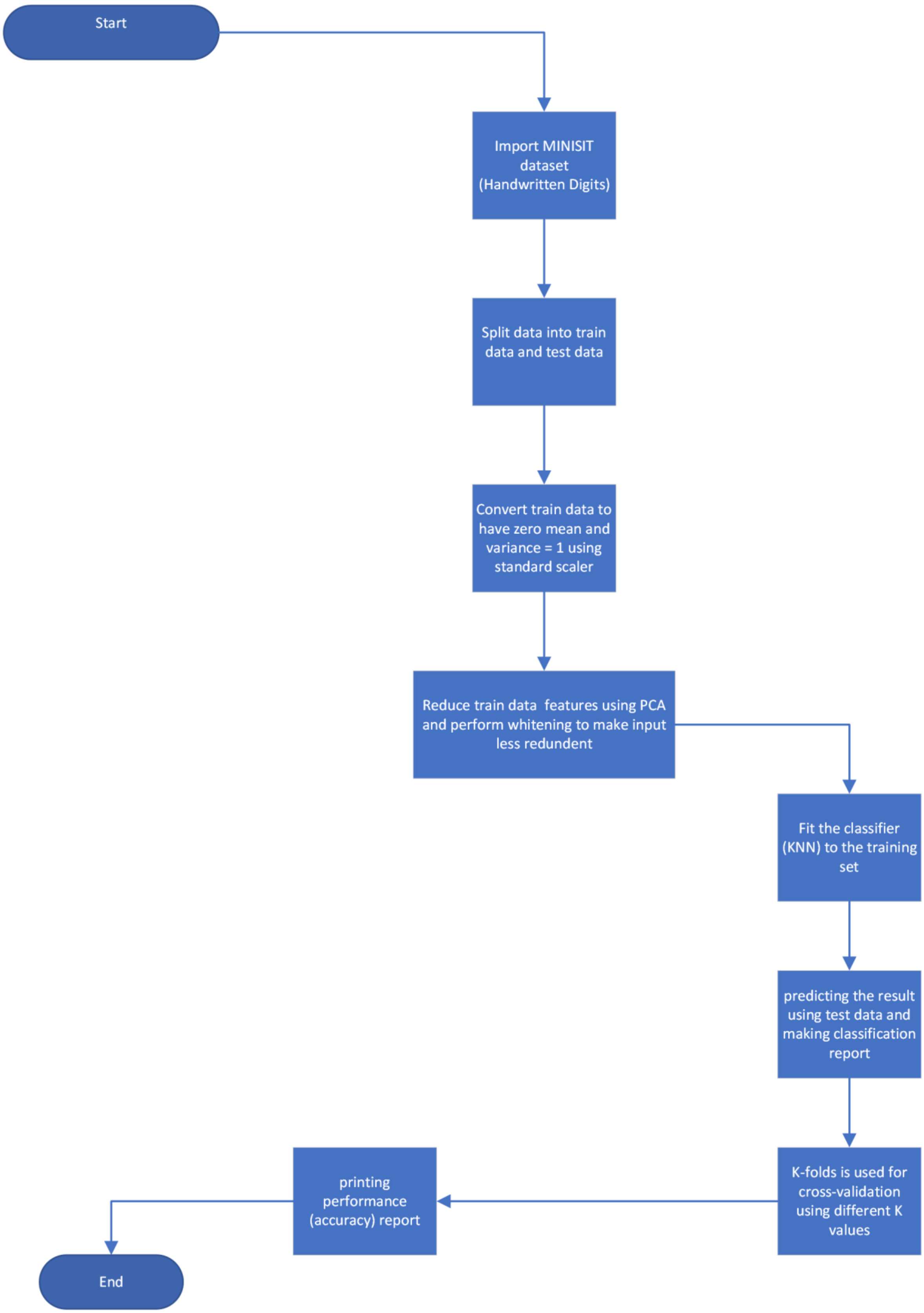
Google Colab Link:

https://colab.research.google.com/drive/1otj-OWf9gcVA44r5Y9CqdDwZAl4KkMEP?usp=sharing

Explanation for the classifier:

We used the KNN (K-Nearest Neighbours). It is based on the supervised learning technique. KNN assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. It stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. During the training phase it just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Flow Chart:



Start
Import MINISIT dataset (Handwritten Digits)
Split data into train data and test data
Convert train data to have zero mean and variance = 1 using standard scaler
Reduce train data features using PCA and perform whitening to make input less redundent
Fit the classifier (KNN) to the training set
predicting the result using test data and making classification report
K-folds is used for cross-validation using different K values
printing performance (accuracy) report
End

Code and Sample Run:

We used Google Colab.

Imports and Fetching data from digits Dataset

```
[ ] from numpy import mean
    from numpy import std
    from sklearn.decomposition import PCA
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn import metrics
    from sklearn.model_selection import KFold
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import train_test_split
    from keras.datasets import mnist

    (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

Convert X_train to zero mean and variance of 1 to be easier to use by using the PCA(Principal Component Analysis) to fit the X_train which reduce the variable numbers to smaller number to be easier and faster to deal with, whiten is used to make input less redundent

```
[ ] # performing preprocessing part
    from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = X_train.reshape(len(X_train), -1) #Converting 2d array to Vector
    X_train = sc.fit_transform(X_train)

    X_test = X_test.reshape(len(X_test), -1) #Converting 2d array to Vector
    X_test = sc.transform(X_test)

    # Compute a PCA
    n_components = 75
    pca = PCA(n_components=n_components, whiten=True).fit(X_train)
    #n_components -> principal components used in dimensionality reduction
    #whiten -> it is needed for some algorithms. If we are training on images, the raw input is redundant,
    #since adjacent pixel values are highly correlated. The goal of whitening is to make the input less redundant
    # apply PCA transformation
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)
```

```
[ ]  from sklearn.neighbors import KNeighborsClassifier
     print("Fitting the classifier to the training set")
     clf = KNeighborsClassifier(n_neighbors=7).fit(X_train_pca, y_train)
     # clf= GaussianNB().fit(X_train, y_train)
     # clf= GaussianNB(var_smoothing=2e-5).fit(X_train_pca, y_train)
     #1e-9 -> default variance so when decreasing it , accuracy decreases
```

```
Fitting the classifier to the training set
```

Using test in prediction and printing a classification report and an accuracy score

```
[ ]  y_pred = clf.predict(X_test_pca)
     print(metrics.classification_report(y_test, y_pred))
     print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.96      0.99      0.98      1135
           2       0.96      0.94      0.95      1032
           3       0.92      0.95      0.94      1010
           4       0.96      0.94      0.95       982
           5       0.93      0.94      0.94       892
           6       0.97      0.97      0.97       958
           7       0.95      0.93      0.94      1028
           8       0.95      0.92      0.93       974
           9       0.93      0.93      0.93      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000

Accuracy: 0.9495
```

KFold is used to cross so that the test data and trained data are merged and the naive bayes model is used

```
[ ] for i in range(5,20):
      # cv = KFold(n_splits=i, random_state=1, shuffle=True)
      cv = KFold(n_splits=i)
      scores = cross_val_score(clf, X_train_pca, y_train, scoring='accuracy', cv=cv, n_jobs=-1)
      # report performance
      print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.948 (0.002)
Accuracy: 0.948 (0.004)
Accuracy: 0.948 (0.005)
Accuracy: 0.949 (0.005)
Accuracy: 0.949 (0.004)
Accuracy: 0.950 (0.005)
Accuracy: 0.949 (0.005)
Accuracy: 0.949 (0.006)
Accuracy: 0.950 (0.006)
Accuracy: 0.949 (0.005)
Accuracy: 0.950 (0.006)
Accuracy: 0.950 (0.006)
Accuracy: 0.950 (0.006)
Accuracy: 0.950 (0.007)
Accuracy: 0.950 (0.007)
```

Running the classifier for different K values:

For K = 7:

Using test in prediction and printing a classification report and an accuracy score

```
[ ] y_pred = clf.predict(X_test_pca)
    print(metrics.classification_report(y_test, y_pred))
    print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.96      0.99      0.98      1135
           2       0.96      0.94      0.95      1032
           3       0.92      0.95      0.94      1010
           4       0.96      0.94      0.95       982
           5       0.93      0.94      0.94       892
           6       0.97      0.97      0.97       958
           7       0.95      0.93      0.94      1028
           8       0.95      0.92      0.93       974
           9       0.93      0.93      0.93      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000

Accuracy: 0.9495
```

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    print("Fitting the classifier to the training set")
    clf = KNeighborsClassifier(n_neighbors=7).fit(X_train_pca, y_train)
    # clf= GaussianNB().fit(X_train, y_train)
    # clf= GaussianNB(var_smoothing=2e-5).fit(X_train_pca, y_train)
    #1e-9 -> default variance so when decreasing it , accuracy decreases
```

```
Fitting the classifier to the training set
```

KFold is used to cross so that the test data and trained data are merged and the naive bayes model is used

```
[ ] for i in range(5,20):
      # cv = KFold(n_splits=i, random_state=1, shuffle=True)
      cv = KFold(n_splits=i)
      scores = cross_val_score(clf, X_train_pca, y_train, scoring='accuracy', cv=cv, n_jobs=-1)
      # report performance
      print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.948 (0.002)
Accuracy: 0.948 (0.004)
Accuracy: 0.948 (0.005)
Accuracy: 0.949 (0.005)
Accuracy: 0.949 (0.004)
Accuracy: 0.950 (0.005)
Accuracy: 0.949 (0.005)
Accuracy: 0.949 (0.006)
Accuracy: 0.950 (0.006)
Accuracy: 0.949 (0.005)
Accuracy: 0.950 (0.006)
Accuracy: 0.950 (0.006)
Accuracy: 0.950 (0.006)
Accuracy: 0.950 (0.007)
Accuracy: 0.950 (0.007)
```

For k = 6:

```
[6] from sklearn.neighbors import KNeighborsClassifier
    print("Fitting the classifier to the training set")
    clf = KNeighborsClassifier(n_neighbors=6).fit(X_train_pca, y_train)
    # clf= GaussianNB().fit(X_train, y_train)
    # clf= GaussianNB(var_smoothing=2e-5).fit(X_train_pca, y_train)
    #1e-9 -> default variance so when decreasing it , accuracy decreases
```

```
Fitting the classifier to the training set
```

Using test in prediction and printing a classification report and an accuracy score

```
y_pred = clf.predict(X_test_pca)
print(metrics.classification_report(y_test, y_pred))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.96      0.99      0.98      1135
           2       0.95      0.94      0.94      1032
           3       0.93      0.96      0.94      1010
           4       0.96      0.95      0.95       982
           5       0.93      0.94      0.94       892
           6       0.97      0.97      0.97       958
           7       0.95      0.94      0.94      1028
           8       0.95      0.91      0.93       974
           9       0.95      0.91      0.93      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000

Accuracy: 0.9504
```

KFold is used to cross so that the test data and trained data are merged and the naive bayes model is used

```
for i in range(5,20):
  # cv = KFold(n_splits=i, random_state=1, shuffle=True)
  cv = KFold(n_splits=i)
  scores = cross_val_score(clf, X_train_pca, y_train, scoring='accuracy', cv=cv)
  # report performance
  print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.948 (0.002)
Accuracy: 0.948 (0.004)
Accuracy: 0.949 (0.005)
Accuracy: 0.949 (0.005)
Accuracy: 0.949 (0.004)
Accuracy: 0.949 (0.005)
Accuracy: 0.949 (0.006)
Accuracy: 0.949 (0.007)
Accuracy: 0.949 (0.005)
Accuracy: 0.949 (0.005)
Accuracy: 0.950 (0.005)
Accuracy: 0.949 (0.006)
Accuracy: 0.950 (0.006)
Accuracy: 0.949 (0.006)
Accuracy: 0.950 (0.006)
```

For k = 5:

```
[10] from sklearn.neighbors import KNeighborsClassifier
     print("Fitting the classifier to the training set")
     clf = KNeighborsClassifier(n_neighbors=5).fit(X_train_pca, y_train)
     # clf= GaussianNB().fit(X_train, y_train)
     # clf= GaussianNB(var_smoothing=2e-5).fit(X_train_pca, y_train)
     #1e-9 -> default variance so when decreasing it , accuracy decreases
```

```
Fitting the classifier to the training set
```

Using test in prediction and printing a classification report and an accuracy score

```
y_pred = clf.predict(X_test_pca)
print(metrics.classification_report(y_test, y_pred))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       980
           1       0.96      0.99      0.97      1135
           2       0.95      0.94      0.95      1032
           3       0.93      0.96      0.94      1010
           4       0.97      0.95      0.96       982
           5       0.94      0.94      0.94       892
           6       0.97      0.97      0.97       958
           7       0.95      0.94      0.95      1028
           8       0.96      0.93      0.94       974
           9       0.95      0.93      0.94      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000

Accuracy: 0.9539
```

KFold is used to cross so that the test data and trained data are merged and the naive bayes model is used

```
for i in range(5,20):
  # cv = KFold(n_splits=i, random_state=1, shuffle=True)
  cv = KFold(n_splits=i)
  scores = cross_val_score(clf, X_train_pca, y_train, scoring='accuracy', cv=cv)
  # report performance
  print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.949 (0.002)
Accuracy: 0.950 (0.004)
Accuracy: 0.950 (0.004)
Accuracy: 0.950 (0.005)
Accuracy: 0.950 (0.003)
Accuracy: 0.950 (0.004)
Accuracy: 0.950 (0.005)
Accuracy: 0.950 (0.007)
Accuracy: 0.951 (0.005)
Accuracy: 0.951 (0.005)
Accuracy: 0.951 (0.005)
Accuracy: 0.950 (0.006)
Accuracy: 0.951 (0.005)
Accuracy: 0.951 (0.006)
Accuracy: 0.951 (0.006)
```

For K = 4:

```
from sklearn.neighbors import KNeighborsClassifier
print("Fitting the classifier to the training set")
clf = KNeighborsClassifier(n_neighbors=4).fit(X_train_pca, y_train)
# clf= GaussianNB().fit(X_train, y_train)
# clf= GaussianNB(var_smoothing=2e-5).fit(X_train_pca, y_train)
#1e-9 -> default variance so when decreasing it , accuracy decreases
```

```
Fitting the classifier to the training set
```

Using test in prediction and printing a classification report and an accuracy score

```
y_pred = clf.predict(X_test_pca)
print(metrics.classification_report(y_test, y_pred))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       980
           1       0.96      0.99      0.98      1135
           2       0.95      0.94      0.94      1032
           3       0.93      0.95      0.94      1010
           4       0.96      0.96      0.96       982
           5       0.93      0.94      0.94       892
           6       0.97      0.97      0.97       958
           7       0.94      0.94      0.94      1028
           8       0.95      0.91      0.93       974
           9       0.95      0.91      0.93      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000

Accuracy: 0.9509
```

KFold is used to cross so that the test data and trained data are merged and the naive bayes model is used

```
for i in range(5,20):
  # cv = KFold(n_splits=i, random_state=1, shuffle=True)
  cv = KFold(n_splits=i)
  scores = cross_val_score(clf, X_train_pca, y_train, scoring='accuracy', cv=cv)
  # report performance
  print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.947 (0.002)
Accuracy: 0.947 (0.004)
Accuracy: 0.948 (0.005)
Accuracy: 0.948 (0.005)
Accuracy: 0.948 (0.004)
Accuracy: 0.949 (0.005)
Accuracy: 0.949 (0.006)
Accuracy: 0.949 (0.006)
Accuracy: 0.949 (0.004)
Accuracy: 0.949 (0.005)
Accuracy: 0.949 (0.006)
Accuracy: 0.949 (0.006)
Accuracy: 0.950 (0.005)
Accuracy: 0.949 (0.006)
Accuracy: 0.949 (0.007)
```

As the K decreases, the accuracy decreases but the change is not large.

We found that the best accuracy was for K = 7.