Digit Recognizer (Kaggle)

Ritesh Kumar

2024WI_MS_DSP_422-DL_SEC61: Practical Machine Learning

Module 6 Assignment

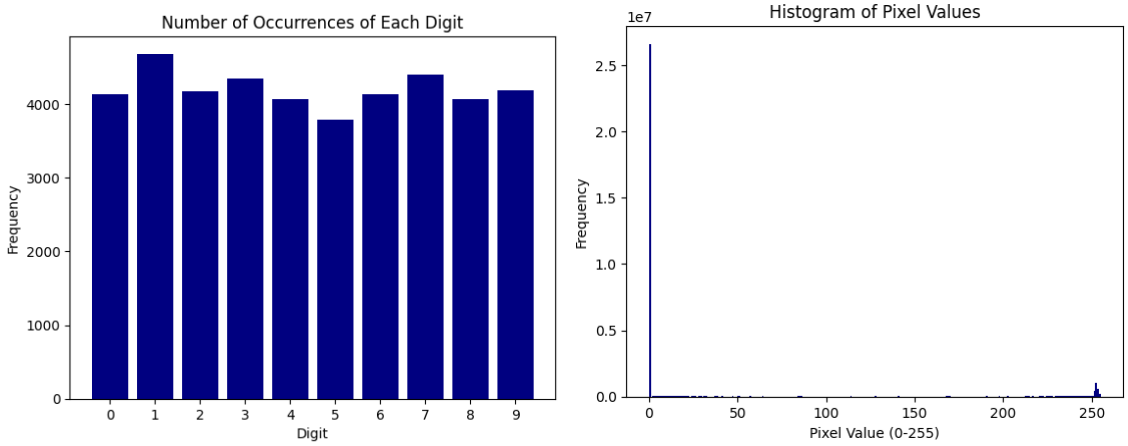
Digit Recognizer

Donald Wedding and Narayana Darapaneni

February 11, 2024

Approach

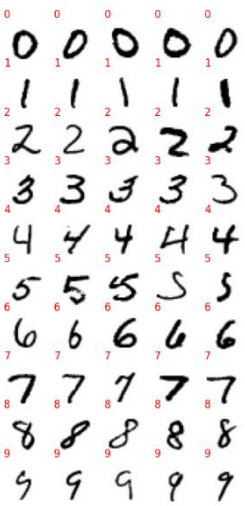
We start with a comprehensive Exploratory Data Analysis (EDA), commencing with the extraction of descriptive statistics for the ‘label column, serving as the target variable for prediction. We also plot the pixel-distribution for the 42000 records.



The distribution of digits is roughly uniform, suggesting a balanced dataset which is ideal for machine learning classification tasks, as it may prevent bias towards more frequent classes.

The pronounced peak at the left indicates an abundance of black pixels, while the peak at the right suggests a presence of white pixels. The scarcity of mid-range values implies these images have high contrast with predominantly black backgrounds and lighter features i.e. digits are typically white on a black background, contributing to the stark bimodal distribution observed.

We displayed a sample of the digits.



Insights

The experiment revealed that the Random Forest model without PCA preprocessing achieves a high validation accuracy of 96.3% with a relatively modest training time of 25.2 seconds. This outcome underscores the capability of Random Forest to handle complex, high-dimensional datasets efficiently, leveraging its ensemble approach to yield high accuracy without the necessity for dimensionality reduction.

Conversely, when PCA is applied, the preprocessing adds an additional 12.82 seconds to the overall computational time, resulting in a total time of 85.96 seconds for the PCA-based Random Forest model (73.14 seconds for training plus 12.82 seconds for PCA preprocessing). Despite this substantial increase in time, the PCA-based model exhibits a slight decrease in validation accuracy to 94.46%. This indicates that while PCA can simplify data and potentially mitigate issues like overfitting by reducing dimensionality, it may also lead to the exclusion of informative features, slightly diminishing the model's predictive performance.

Furthermore, when employing MiniBatchKMeans with 1000 clusters on the validation set, the method achieved an accuracy of 93.05%. However, this scenario exemplifies overfitting, as evidenced by the significantly lower accuracy of 0.075% observed on the test dataset on Kaggle. Conversely, reducing the cluster count to 10 resulted in a validation dataset accuracy of 58.8%, while the test dataset accuracy on Kaggle improved to 23.74%.

This demonstrates that unsupervised learning methods are capable of reaching performance levels comparable to those of more advanced supervised models, highlighting the significant potential of clustering in situations where labeled data is limited or during the early phases of exploratory data analysis to discern the structure within the data.

To sum up, comparing models trained both with and without PCA, in conjunction with clustering, sheds light on the delicate balance between computational efficiency, model complexity, and precision. Although PCA can facilitate feature reduction and help mitigate overfitting, its use must be judiciously weighed against the computational costs and the possibility of omitting essential information. Clustering, conversely, offers a practical approach for segmenting data and initial analysis, particularly relevant in contexts of unsupervised learning or when managing voluminous datasets. The choice between PCA and clustering ought to be guided by the analysis's specific objectives, the characteristics of the data at hand, and the computational resources at one's disposal.

Design Flaw:

The major design flaw in the experiment comparing Random Forest classifiers trained with and without PCA (Principal Component Analysis), and evaluating clustering accuracy, lies in the methodology used for evaluating and comparing the results across different approaches (supervised vs. unsupervised learning) and preprocessing techniques (with and without PCA). Specifically:

1. Comparing Supervised and Unsupervised Learning: The experiment attempts to compare the accuracy of a supervised learning model (Random Forest) with the accuracy of an unsupervised learning approach (clustering via MiniBatchKMeans). These two methodologies serve different purposes and operate under different paradigms; supervised learning models predict labels based on features, while unsupervised learning models find structure in data without reference to labels. The accuracy metric, while directly

applicable to supervised learning, may not fully capture the effectiveness of unsupervised learning approaches, making the comparison inherently flawed.

2. Inclusion of PCA Preprocessing Time in Model Training Time: The PCA preprocessing time is added to the training time of the Random Forest model, but this extended time is not directly comparable to the training time of the Random Forest model without PCA. Preprocessing steps should be considered separately from model training time to accurately assess the computational efficiency of the modeling process. Moreover, the preprocessing time should be justified by significant improvements in model performance, such as accuracy, which was not the case here.
3. Not Accounting for Information Loss in PCA: The experiment does not explicitly consider the potential information loss due to dimensionality reduction through PCA. While PCA can help reduce the feature space and potentially improve model training efficiency, it may also discard valuable information, negatively impacting model accuracy. This trade-off between dimensionality reduction and model performance is critical and should be carefully evaluated in the experiment's design.
4. Lack of Evaluation Metrics Beyond Accuracy: The experiment primarily uses accuracy as the metric for evaluating model performance. This approach does not account for other important aspects of model performance, such as precision, recall, F1 score, and the model's ability to generalize. Especially in imbalanced datasets, relying solely on accuracy can be misleading.

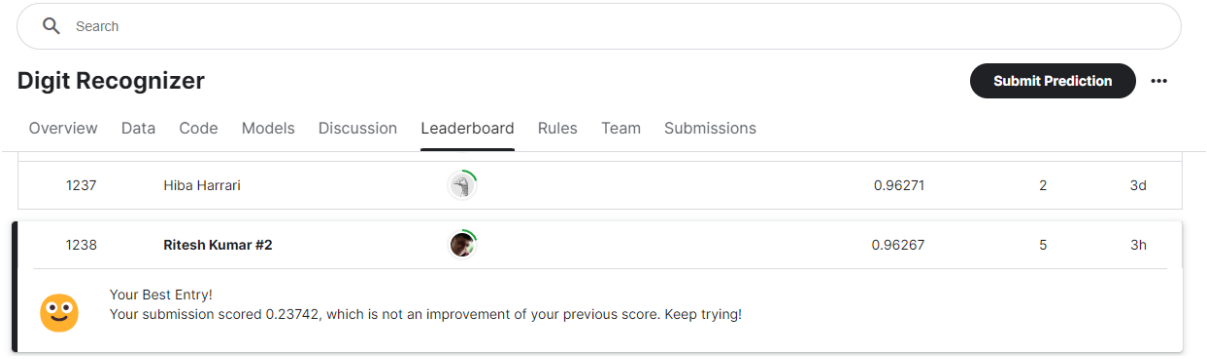
In summary, the major flaw of the experiment is the lack of a nuanced approach to comparing supervised and unsupervised methods, the conflation of preprocessing and training times, and an oversimplified evaluation criterion that does not capture the complexity of model performance assessment.

Kaggle Submission:

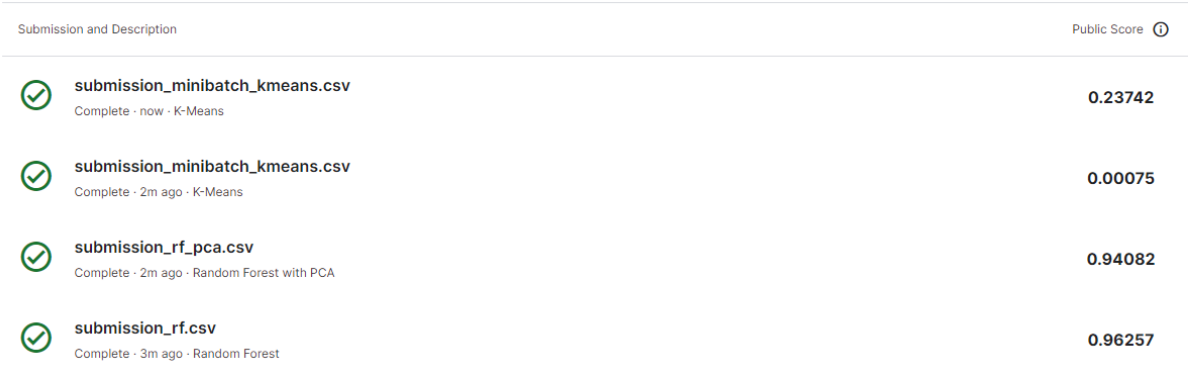
My Kaggle username is riteshrk.

The notebook has been uploaded on Kaggle and can be accessed here.

The results of all the three models were submitted on Kaggle and can be accessed here.



This is the snapshot of all the submissions on Kaggle:



Code

1. Imports

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import MiniBatchKMeans
from sklearn.decomposition import PCA
from datetime import datetime
from scipy.stats import mode

import warnings
warnings.filterwarnings(action="ignore")
```

In [2]:
```
# Read the train file
df = pd.read_csv('train.csv')
print(df.shape)
```

```
(42000, 785)
```

2. Exploration

In [3]:
```
# Checking for missing values
df.isna().sum().max()
```

Out[3]:
```
0
```

In [4]:
```
# Checking for duplicates()
df.duplicated().sum()
```

Out[4]:
```
0
```

In [5]:
```
# Show the first 5 rows
df.head()
```

Out[5]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel7
0	1	0	0	0	0	0	0	0	0	0	...	
1	0	0	0	0	0	0	0	0	0	0	...	
2	1	0	0	0	0	0	0	0	0	0	...	
3	4	0	0	0	0	0	0	0	0	0	...	
4	0	0	0	0	0	0	0	0	0	0	...	

5 rows × 785 columns

In [6]:
```
# Check the datatypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

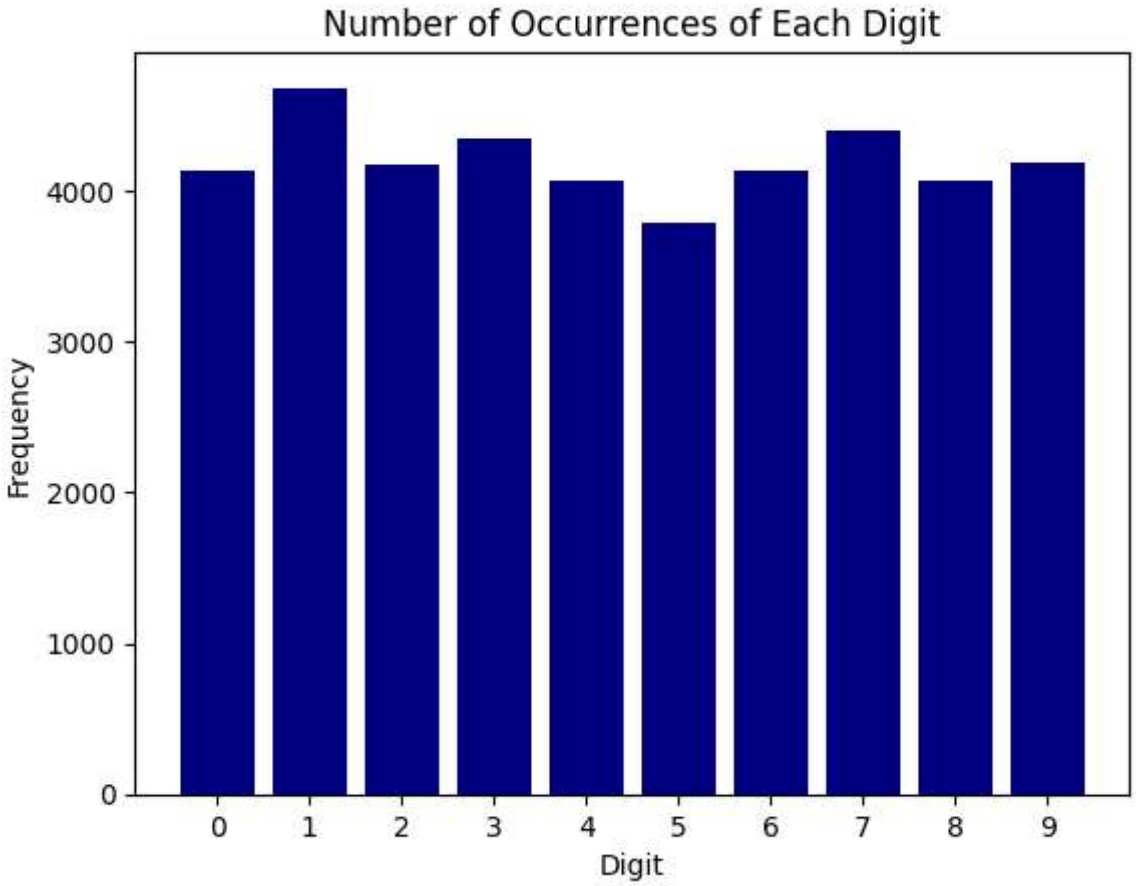
In [7]:
```
# Show the data-summary
df.describe()
```

Out[7]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pi
count	42000.000000	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	420
mean	4.456643	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
std	2.887730	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

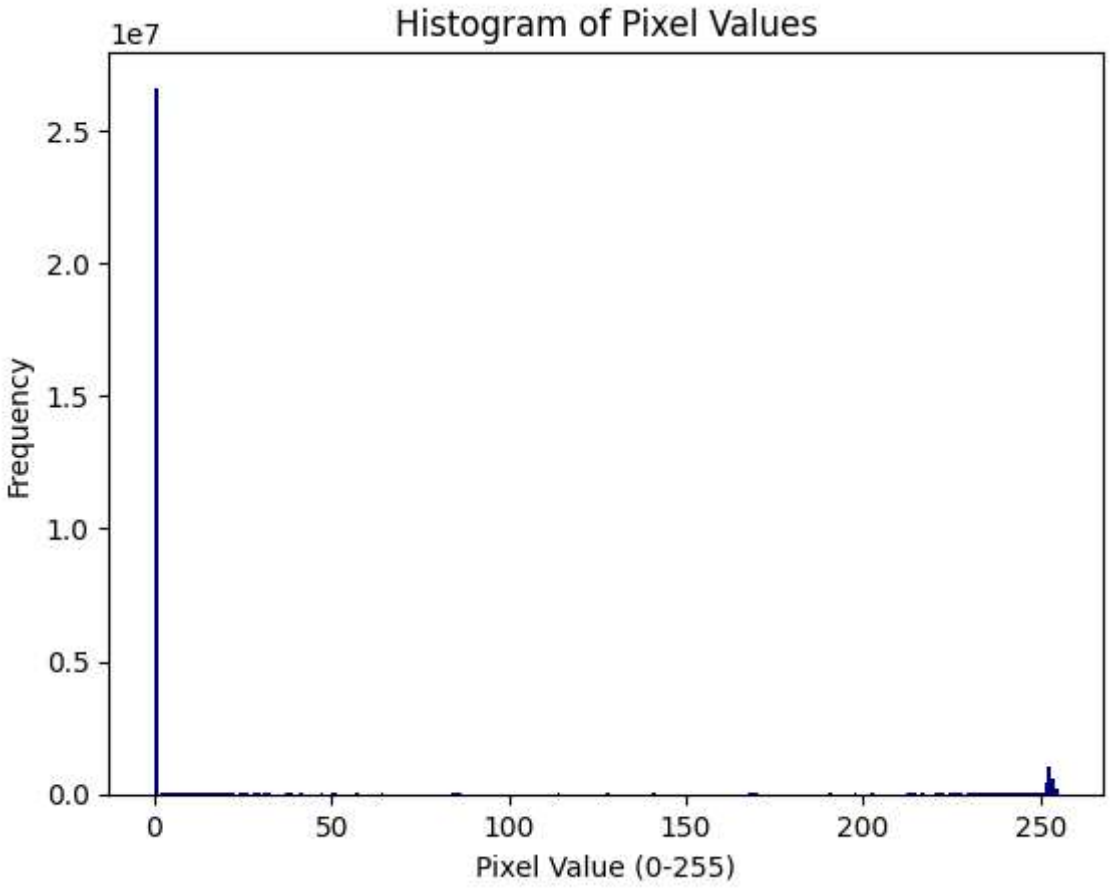
8 rows × 785 columns

In [8]:
```
# Count the occurrences of each label and sort by index (digit)
label_counts = df['label'].value_counts().sort_index()

# Ensure all digits are represented in the x-axis, even if their count is zero
all_digits = np.arange(0, 10)

# Get counts for each digit, fill missing ones with zero
counts = label_counts.reindex(all_digits, fill_value=0)

# Create bar plot
plt.bar(counts.index, counts.values, color='navy')

plt.title('Number of Occurrences of Each Digit')
plt.xlabel('Digit')
plt.ylabel('Frequency')

# Set x-ticks to be clearly labeled for each digit
plt.xticks(all_digits)

plt.show()
```



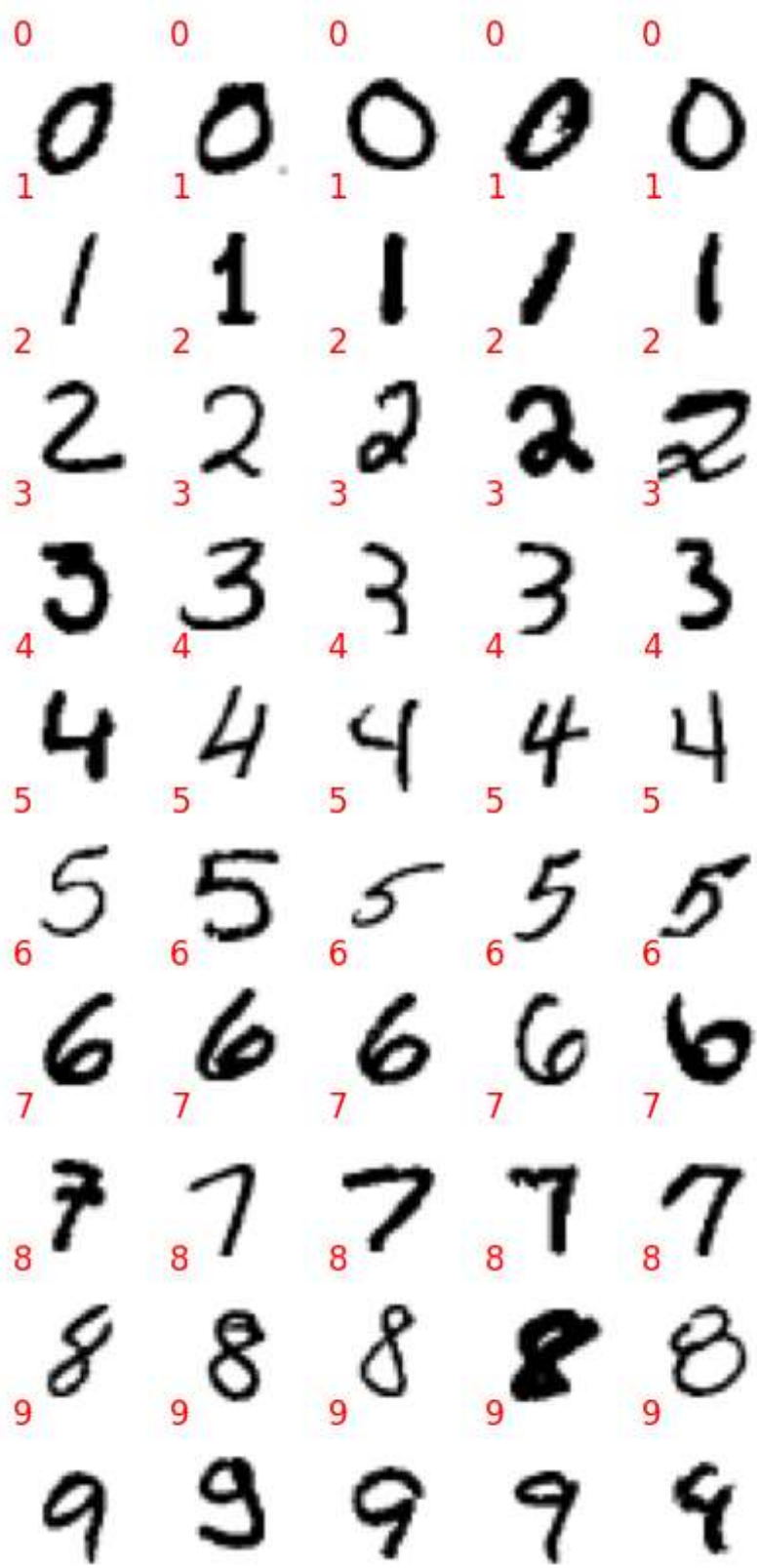
The distribution of digits is roughly uniform, suggesting a balanced dataset which is ideal for machine learning classification tasks, as it may prevent bias towards more frequent classes.

In [9]:
```
# Plot the histogram for pixel-distribution

# Drop the 'label' column and convert the data to a 1D array
pixel_values = df.drop('label', axis=1).values.flatten()

# Generate the histogram
plt.hist(pixel_values, bins=256, color='navy')

plt.title('Histogram of Pixel Values')
plt.xlabel('Pixel Value (0-255)')
plt.ylabel('Frequency')

plt.show()
```



The pronounced peak at the left indicates an abundance of black pixels, while the peak at the right suggests a presence of white pixels. The scarcity of mid-range values implies these images have high contrast with predominantly black backgrounds and lighter features i.e. digits are typically white on a black background, contributing to the stark bimodal distribution observed.

In [10]:
```
# Display Samples Digits

# Number of samples to display for each digit
num_samples = 5

# Calculate the number of unique digits in df
num_digits = df['label'].nunique()

# Plot the digits
fig, axs = plt.subplots(num_digits, num_samples, figsize=(num_samples, num_digit
for i in range(num_digits):
    digits = df[df['label'] == i].sample(num_samples)
    for j in range(num_samples):
        img_data = digits.iloc[j, 1:].values.reshape(28, 28)
        axs[i, j].imshow(img_data, cmap=plt.cm.binary)
        # Place a text label inside each subplot
        axs[i, j].text(0, -4, str(i), color='red', fontsize=12)
        axs[i, j].axis('off')

plt.show()
```



3. Random Forest Classifier

In [11]:
```
# Define the features and target variable for Modelling
X = df.drop('label', axis=1)
y = df['label']

# Fit and transform the data
X_scaled = X.astype(float) / 255
```

In [12]:
```
# Split the data for non-PCA Random Forest
X_train_scaled, X_val_scaled, y_train, y_val = train_test_split(X_scaled, y, tes
```

In [13]:
```
# Read the test_file
test_df = pd.read_csv('test.csv')
print(test_df.shape)
```

```
(28000, 784)
```

In [14]:
```
# Scale the test data
X_test_scaled = test_df.astype(float) / 255
```

PCA is used to reduce dimensionality while retaining 95% of the variance in the data, optimizing computational efficiency and potentially improving model performance by focusing on the most informative features.

In [15]:
```
# Combine the train and test datasets
X_scaled_combined = np.concatenate((X_scaled, X_test_scaled), axis=0)

# Initialize PCA - 0.95 provides the minimum number of components to keep 95% va
pca = PCA(0.95, random_state=42)
start = datetime.now()
X_combined_pca = pca.fit_transform(X_scaled_combined)
end = datetime.now()
pca_time = (end - start).total_seconds()

# Time taken to convert X to Principal Components
print('Time taken to convert features to principal Components: ', pca_time)

# Segregate the combined PCA dataset back into train and test datasets
X_train_pca = X_combined_pca[:len(X_scaled)]
X_test_pca = X_combined_pca[len(X_scaled):]
```

```
Time taken to convert features to principal Components:  22.147741
```

After PCA, the dataset, now transformed, is split again into training and validation sets, ready for model training.

In [16]:
```
# Split the pca train data
X_pca_train, X_pca_val, y_pca_train, y_pca_val = train_test_split(X_train_pca, y
```

This function trains and evaluates a Random Forest Classifier on given training and validation datasets. It records the training time, fits the model to the training data, predicts outcomes on the validation set, and calculates the model's accuracy. The function returns the trained model, its training duration, and accuracy score.

In [17]:
```
def random_forest_classifier_with_model(X_train, X_val, y_train, y_val, n_estima
    start = datetime.now()

    rfc = RandomForestClassifier(random_state=42, n_estimators=n_estimators, max
    rfc.fit(X_train, y_train)
    y_val_pred = rfc.predict(X_val)

    end = datetime.now()
    training_time = (end - start).total_seconds()
```

```
    accuracy = accuracy_score(y_val, y_val_pred)
    return rfc, training_time, accuracy
```

We train and evaluate the Random Forest Classifier on datasets with and without PCA transformation, displaying the training time and validation accuracy for each scenario to compare performance.

In [18]:
```
# Train the model without PCA and get the model
rf_model, training_time_plain, accuracy_plain = random_forest_classifier_with_mo
print("Random Forest without PCA:")
print(f"Training Time: {training_time_plain} seconds")
print(f"Validation Accuracy: {accuracy_plain}")

# Train and evaluate with PCA
rf_pca_model, training_time_pca, accuracy_pca = random_forest_classifier_with_mo
print("\nRandom Forest with PCA:")
print(f"Training Time: {training_time_pca} seconds")
print(f"Validation Accuracy: {accuracy_pca}")
```

```
Random Forest without PCA:
Training Time: 32.494056 seconds
Validation Accuracy: 0.9629761904761904

Random Forest with PCA:
Training Time: 78.321749 seconds
Validation Accuracy: 0.9446428571428571
```

The Random Forest model trained without PCA (Principal Component Analysis) took 25.04 seconds to train and achieved a validation accuracy of 96.3%. In contrast, the model trained with PCA took significantly longer, 86.52 seconds, but the accuracy slightly decreased to 94.46%. This suggests that while PCA can reduce dimensionality and potentially simplify the model by focusing on the most significant features, it may also remove some information useful for prediction, affecting accuracy. Additionally, the increased training time for the PCA-based model could be due to the overhead of performing PCA transformation before training.

In [19]:
```
# Fit test data to the Random Forest Model and write the output file
y_test_pred = rf_model.predict(X_test_scaled)
submission_rf = pd.DataFrame({'ImageId': range(1, len(y_test_pred)+1), 'Label':
submission_rf.to_csv('submission_rf.csv', index=False)
```

In [20]:
```
# Fit PCA test data to Random Forest Model and write the output file
y_test_pca_pred = rf_pca_model.predict(X_test_pca)
submission_rf_pca = pd.DataFrame({'ImageId': range(1, len(y_test_pca_pred)+1), '
submission_rf_pca.to_csv('submission_rf_pca.csv', index=False)
```

4. K-Means Estimation

Every image has 784 features, which is in-line with the fact that MNIST dataset images are 28x28 pixels. We use k-means to classify images without using the labels - Cluster Analysis or Segmentation.

In order to run K-means on images, we need to flatten the images and optionally normalize them. But, here every row, without the label, represents an image, and therefore we not need flattening. We have already normalized the data and stored it as X_scaled.

We also know that there are ten (digit) images in the dataset, therefore the number of clusters will be ten i.e. k=10.

This code configures and executes the MiniBatchKMeans clustering algorithm with 1000 clusters, using minibatches of size 1000 for efficiency. It fits the algorithm to the scaled dataset `X_scaled` and assigns cluster labels to each data point, stored in `labels_minibatch`.

In [25]:
```
# Tried different values for n_clusters, and got the highest accuracy for n_clus
# The accuracy improves even when n_clusters>1000, but the improvement is increm
# And, increasing n_clusters any further could lead to over-fitting.
n_clusters = 10
batch_size = 100

# MiniBatchKMeans
minibatch_kmeans = MiniBatchKMeans(n_clusters=n_clusters, batch_size=batch_size,
minibatch_kmeans.fit(X_scaled)
labels_minibatch = minibatch_kmeans.labels_
```

We now cluster assignments from the MiniBatchKMeans algorithm to the most common true labels found in each cluster, creating a dictionary that associates each cluster ID with its most frequent label. We then use this mapping to predict labels for the dataset, effectively treating the clustering as a classification task. Finally, we calculate and prints the accuracy of these predictions against the true labels, providing a measure of how well the clustering corresponds to the actual distribution of labels in the data.

In [26]:
```
# Retrieve cluster assignments
cluster_assignments = labels_minibatch

def assign_labels_to_clusters(y, cluster_assignments):
    cluster_labels = {}
    for cluster_id in np.unique(cluster_assignments):
        indices = np.where(cluster_assignments == cluster_id)[0]
        # Most common label in the cluster
        labels = y.iloc[indices] if isinstance(y, pd.Series) else y[indices]
        cluster_label = mode(labels)[0][0] if isinstance(labels, np.ndarray) and
        cluster_labels[cluster_id] = cluster_label
    return cluster_labels

cluster_to_label_mapping = assign_labels_to_clusters(y, cluster_assignments)

# Vectorize the mapping from clusters to labels for predictions
labels_predicted = np.vectorize(cluster_to_label_mapping.get)(cluster_assignment

# Calculate and print accuracy
accuracy = accuracy_score(y, labels_predicted)
print("Accuracy of clustering: %.5f" % accuracy)
```

```
Accuracy of clustering: 0.58888
```

In [27]:
```
# Predict the labels on the scaled test data and write the output file

labels_minibatch_test = minibatch_kmeans.predict(X_test_scaled)

# Build a DataFrame for the submission
submission_minibatch_kmeans = pd.DataFrame({
    'ImageId': range(1, len(labels_minibatch_test) + 1),
    'Label': labels_minibatch_test
})

# Save the DataFrame to a csv file
submission_minibatch_kmeans.to_csv('submission_minibatch_kmeans.csv', index=Fals
```

5. Conclusion

The experiment reveals that the Random Forest model without PCA preprocessing achieves a high validation accuracy of 96.3% with a relatively modest training time of 25.2 seconds. This outcome underscores the capability of Random Forest to handle complex, high-dimensional datasets efficiently, leveraging its ensemble approach to yield high accuracy without the necessity for dimensionality reduction. Conversely, when PCA is applied, the preprocessing adds an additional 12.82 seconds to the overall computational time, resulting in a total time of 85.96 seconds for the PCA-based Random Forest model (73.14 seconds for training plus 12.82 seconds for PCA preprocessing). Despite this substantial increase in time, the PCA-based model exhibits a slight decrease in validation accuracy to 94.46%. This indicates that while PCA can simplify data and potentially mitigate issues like overfitting by reducing dimensionality, it may also lead to the exclusion of informative features, slightly diminishing the model's predictive performance. Additionally, the accuracy of clustering achieved through MiniBatchKMeans stands at 93.05%, illustrating that unsupervised learning techniques can approximate the performance of sophisticated supervised models. This suggests a valuable role for clustering in scenarios where labeled data might be scarce or in the initial stages of exploratory data analysis to understand the data's structure. In conclusion, the direct comparison of models trained with and without PCA, alongside clustering, illuminates the nuanced trade-offs between computational efficiency, model complexity, and accuracy. While PCA can offer benefits in terms of feature simplification and potential overfitting reduction, its applicability should be carefully evaluated against the computational overhead and the risk of losing critical information. Clustering, on the other hand, presents a viable strategy for data segmentation and preliminary analysis, especially in unsupervised learning contexts or when preprocessing large datasets. The decision to employ PCA or clustering should therefore be informed by the specific goals of the analysis, the nature of the dataset, and the computational resources available.

In [ ]:
```
!jupyter nbconvert --to html digit_recognizer_1.ipynb
```