Problem 2: Classification on 20newsgroup Data

Problem 1

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import confusion_matrix
import numpy as np
import pandas as pd
```

In [ ]:
```
data_frame = pd.read_csv('20newsgroup\data_set.csv')
```

In [ ]:
```
data_frame.head()
```

Out[ ]:

	group	aids	baseball	bible	bmw	cancer	car	card	case	children	...	university	ver
0	1	0	0	0	0	0	0	0	0	0	...	0	
1	1	0	0	0	0	0	0	0	0	0	...	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	
3	1	0	0	0	0	0	0	0	0	0	...	1	
4	1	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 101 columns

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(data_frame.iloc[:,1:], d
```

First we assume number of estimators is 100,and max features is sqrt

In [ ]:
```
rf = RandomForestClassifier(n_estimators=100, max_features="sqrt", random_st
rf.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier(max_features='sqrt', random_state=42)
```

In [ ]:
```
cv_error = 1 - np.mean(cross_val_score(rf, X_train, y_train, cv=5))
print(cv_error)
```

```
0.19995364475174182
```

In [ ]:
```
y_pred = rf.predict(X_test)
confusion_matrix = confusion_matrix(y_test, y_pred)
```

In [ ]:
```
print(confusion_matrix)
```

```
[[812  32  35  43]
 [ 49 563  26  78]
 [110  34 329  75]
 [ 38  30  44 951]]
```

Next, we do some gird search

In [ ]:
```
grid = {
    'n_estimators': [50,100,150,200],
    'max_features': ['sqrt', 'log2'],
    # 可添加更多参数
}
```

In [ ]:
```
rf = RandomForestClassifier(random_state=20987228)
grid_search = GridSearchCV(rf, grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
```

Out[ ]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=20987228),
             n_jobs=-1,
             param_grid={'max_features': ['sqrt', 'log2'],
                         'n_estimators': [50, 100, 150, 200]},
             scoring='accuracy')
```

In [ ]:
```
print(grid_search.best_params_)
```

```
{'max_features': 'log2', 'n_estimators': 200}
```

We find the performance would be better when take max_features as log2, and n_estimator as 200

In [ ]:
```
rf = RandomForestClassifier(n_estimators=200, max_features="log2", random_st
rf.fit(X_train, y_train)
cv_error = 1 - np.mean(cross_val_score(rf, X_train, y_train, cv=5))
print(cv_error)
```

```
0.19664426508567134
```

In [ ]:
```
y_pred = rf.predict(X_test)
confusion_matrix = confusion_matrix(y_test, y_pred)
```

In [ ]:
```
print(confusion_matrix)
```

```
[[819  29  36  38]
 [ 51 568  22  75]
 [111  31 334  72]
 [ 34  25  42 962]]
```

In fact, it didn't improve so much

Problem 2

In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [ ]:
```
grid = {
    'n_estimators': [ 100, 200,300],
    'learning_rate': [0.01, 0.1] ,
    'max_depth':[5,10]

}

gbc = GradientBoostingClassifier(random_state=20987228)
grid_search = GridSearchCV(gbc, grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
```

```
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
```

In [ ]:
```
gbc = GradientBoostingClassifier(random_state=20987228,learning_rate = 0.1,
gbc.fit(X_train, y_train)
```

```
cv_error = 1 - np.mean(cross_val_score(gbc, X_train, y_train, cv=5))
print(cv_error)
```

```
0.18302180533106094
```

In [ ]:
```
y_pred = gbc.predict(X_test)
confusion_matrix = confusion_matrix(y_test, y_pred)
```

In [ ]:
```
print(confusion_matrix)
```

```
[[822  11  47  42]
 [ 53 547  33  83]
 [108  15 355  70]
 [ 39  30  46 948]]
```

Comparing the results from random forest and boosting trees, we observe that the latter model exhibits a relatively smaller classification cv_error.

Problem 4

In [ ]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
cv_error = 1 - np.mean(cross_val_score(lda, X_train, y_train, cv=5))
print(cv_error)
```

```
0.20488024499267055
```

Problem 5

In [ ]:
```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)
cv_error = 1 - np.mean(cross_val_score(qda, X_train, y_train, cv=5))
print(cv_error)
```

```
C:\Users\Peterson\anaconda3\lib\site-packages\sklearn\discriminant_analysi
s.py:878: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\Peterson\anaconda3\lib\site-packages\sklearn\discriminant_analysi
s.py:878: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\Peterson\anaconda3\lib\site-packages\sklearn\discriminant_analysi
s.py:878: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\Peterson\anaconda3\lib\site-packages\sklearn\discriminant_analysi
s.py:878: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\Peterson\anaconda3\lib\site-packages\sklearn\discriminant_analysi
s.py:878: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\Peterson\anaconda3\lib\site-packages\sklearn\discriminant_analysi
s.py:878: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
0.2928521984383762
```

Problem 6

In [ ]:
```
from sklearn.svm import SVC
svm = SVC(kernel='linear', random_state=20987228)
svm.fit(X_train, y_train)
```

```
cv_error = 1 - np.mean(cross_val_score(svm, X_train, y_train, cv=5))
print(cv_error)
```

0.19402793340602076

Under comparesion these 3 methods, we can find that, SVM model have a better performance with 0.194 cv_error, and QDA have the worst performance,it's cv error is nearly 30. these methods is not good for do this classification task.

Problem 3 Spectral Clustering (PCA + K-means)on 20newsgroup Data

Problem 1, implement KNN

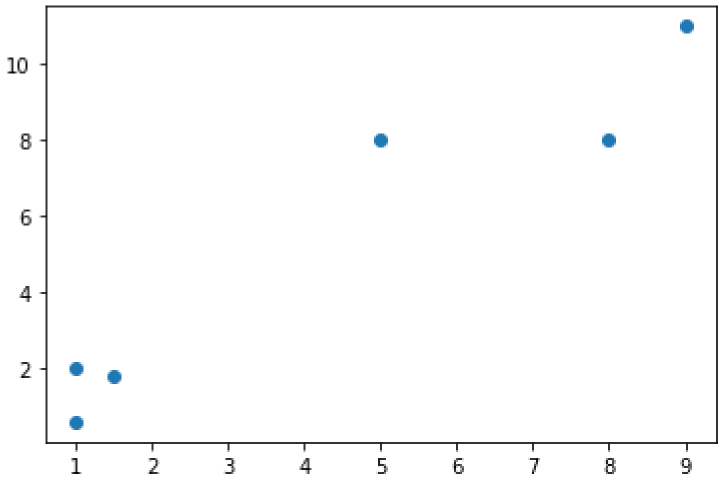
In [ ]:
```
import numpy as np
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
import random
```

In [ ]:
```
class K_Means:
    def __init__(self, k=2):
        self.k_ = k
    def calculate_difference(self,dict1, dict2):
        diff_dict = {}

        # 遍历第一个字典的键值对
        for key, value in dict1.items():
            if key in dict2:
                # 检查第二个字典是否包含相同的键
                value_diff = value - dict2[key]
                diff_dict[key] = np.linalg.norm(value_diff, ord=2)  # 计算二

        return diff_dict
    def fit(self, train_data):
        self.centers_ = {}
        # initialy random select 2 points from train data as centers
        random_center = random.sample(range(len(train_data)), self.k_)
        for i in range(self.k_):
            self.centers_[i] = train_data[random_center[i]]
        while True:#change the center for 300 times
            self.clf_ = {}
            self.label_ = []
            for j in range(self.k_):
                self.clf_[j] = []
            for data_point in train_data:
                distance = []
                for center in self.centers_:
                    distance.append(np.linalg.norm(data_point - self.center
                classification = distance.index(min(distance))#Find the dis
                self.label_.append(classification)
                self.clf_[classification].append(data_point)

            prev_centers = dict(self.centers_)
#           print(prev_centers)
            for c in self.clf_:
                self.centers_[c] = np.average(self.clf_[c], axis=0)
#           print(self.centers_)
            difference_values = self.calculate_difference(prev_centers,self
#           print(difference_values)
```

```
#             print(list(difference_values))
            difference_sum = np.sum(list(difference_values.values()))
            #print(difference_sum)
            if (difference_sum<= 0.0001):
                break
    def predict(self,predict_data):
        label_list = []
        for data_point in predict_data:


            distance = []
            for center in self.centers_:
                distance.append(np.linalg.norm(data_point - self.centers_[ce

                classification = distance.index(min(distance))
            label_list.append(classification)
        return(label_list)

        #return classification
```

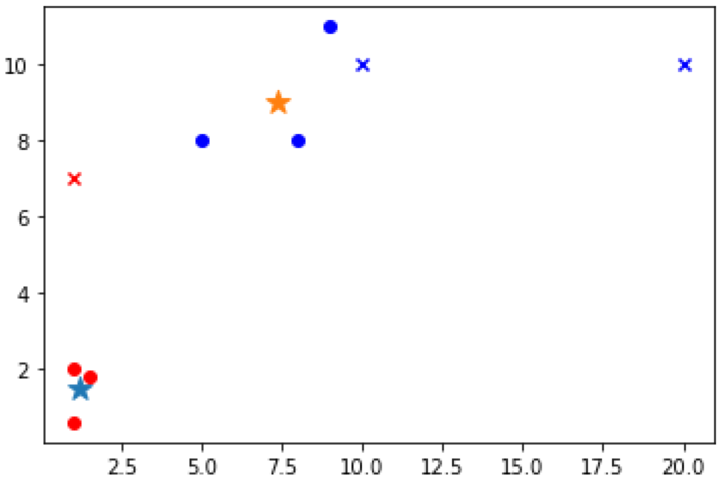
test

In [ ]:
```
x = np.array([ [1,2],[1.5,1.8],[5,8],[8,8],[1,0.6],[9,11] ])
plt.scatter(x[:,0], x[:,1])
plt.show()
```



In [ ]:
```
x = np.array([[1, 2], [1.5, 1.8], [5, 8], [8, 8], [1, 0.6], [9, 11]])
k_means = K_Means(k=2)
k_means.fit(x)
for center in k_means.centers_:
        pyplot.scatter(k_means.centers_[center][0], k_means.centers_[center]
print(k_means.centers_)
for cat in k_means.clf_:
        for point in k_means.clf_[cat]:
            pyplot.scatter(point[0], point[1], c=('r' if cat == 0 else 'b'))
predict = [[1, 7], [20, 10],[10,10]]
cat = k_means.predict(predict)
print(cat)
for i in range(len(cat)):
    pyplot.scatter(predict[i][0], predict[i][1], c=('r' if cat[i] == 0 else
```

```
{0: array([1.16666667, 1.46666667]), 1: array([7.33333333, 9.        ])}
[0, 1, 1]
```



Problem 2

In [ ]:
```python
from sklearn.decomposition import PCA
```

In [ ]:
```python
data_frame = pd.read_csv('20newsgroup\data_set.csv')
```

In [ ]:
```python
occurrence_matrix = data_frame.iloc[:,1:]
```

In [ ]:
```python
occurrence_matrix
```

Out[ ]:

	aids	baseball	bible	bmw	cancer	car	card	case	children	christian	...	univers
0	0	0	0	0	0	0	0	0	0	0	...	
1	0	0	0	0	0	0	0	0	0	0	...	
2	0	0	0	0	0	0	0	0	0	0	...	
3	0	0	0	0	0	0	0	0	0	0	...	
4	0	0	0	0	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
16237	0	0	0	0	0	0	0	0	0	0	...	
16238	0	0	0	0	0	0	0	0	0	0	...	
16239	0	0	0	0	0	0	0	0	0	1	...	
16240	0	0	0	0	0	0	0	0	0	0	...	
16241	0	0	0	0	0	0	0	0	0	1	...	

16242 rows × 100 columns

In [ ]:
```python
pca = PCA(n_components=4)
singular_vectors = pca.fit_transform(occurrence_matrix)
```

In [ ]:
```python
singular_vectors[:10]
```

Out[ ]:
```
array([[ 0.09725084,  0.72355366,  0.37168084, -0.06025652],
       [-0.29544357, -0.16735527,  0.15453398,  0.15594552],
       [-0.24786788,  0.28215541,  0.04644827,  0.07595447],
       [-0.19298501,  0.41508968,  0.82906179, -0.48541374],
       [-0.12360964,  0.38961209,  0.12312901,  0.15977925],
       [-0.20521948,  0.01826588, -0.03171292,  0.09593449],
       [ 0.02399533,  0.68906691,  0.43578386, -0.29719677],
       [-0.06342986,  0.29999905,  0.33899378, -0.31892955],
       [ 0.24340537,  0.63195361,  1.04644662, -0.34879521],
       [-0.05853238,  0.57331061,  0.17062308, -0.85434398]])
```

In [ ]:
```
k_means = K_Means(k=4)
start_time = time.time()
k_means.fit(singular_vectors)
end_time = time.time()
print("Cost time is {}".format(end_time - start_time))
```

```
Cost time is 6.671731233596802
```

In [ ]:
```
cluster_labels = k_means.label_
true_labels = data_frame.iloc[:,1]
```

In [ ]:
```
import numpy as np
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import accuracy_score


# Gets unique values for real and cluster categories
unique_true_labels = np.unique(true_labels)
unique_cluster_labels = np.unique(cluster_labels)

# Create a cost matrix for category matching
cost_matrix = np.zeros((len(unique_true_labels), len(unique_cluster_labels))

# Filling cost matrix
for i, true_label in enumerate(unique_true_labels):
    for j, cluster_label in enumerate(unique_cluster_labels):
        # Calculate the number of mismatches as a cost
        mismatch_count = np.sum((true_labels == true_label) & (cluster_labe
        cost_matrix[i, j] = -mismatch_count

# Find the best match through the least cost match
row_ind, col_ind = linear_sum_assignment(cost_matrix)

# Category re-labeling based on matching results
mapped_labels = np.zeros_like(cluster_labels)
for true_label, cluster_label in zip(unique_true_labels[row_ind], unique_clu
    mapped_labels[cluster_labels == cluster_label] = true_label

# Correct rate of calculation
accuracy = accuracy_score(true_labels, mapped_labels)

# Output result
print("Mis-clustering error rate:", 1-accuracy)
```

```
Mis-clustering error rate: 0.1281861839674917
```

In [ ]:
```
len(mapped_labels)
```

Out[ ]:
```
16242
```

Problem 3

In [ ]:
```
pca = PCA(n_components=5)
singular_vectors = pca.fit_transform(occurrence_matrix)
```

In [ ]:
```
k_means = K_Means(k=4)
start_time = time.time()
k_means.fit(singular_vectors)
end_time = time.time()
print("Cost time is {}".format(end_time - start_time))
```

```
Cost time is 10.195596694946289
```

We do not know the label relationship between the current clustering and the correct clustering, so we use 432*1 method to conduct exhaustive analysis and select the one with the highest matching degree as the result

In [ ]:
```
cluster_labels = k_means.label_
true_labels = data_frame.iloc[:,1]
# Gets unique values for real and cluster categories
unique_true_labels = np.unique(true_labels)
unique_cluster_labels = np.unique(cluster_labels)

# Create a cost matrix for category matching
cost_matrix = np.zeros((len(unique_true_labels), len(unique_cluster_labels))

# Filling cost matrix
for i, true_label in enumerate(unique_true_labels):
    for j, cluster_label in enumerate(unique_cluster_labels):
        # Calculate the number of mismatches as a cost
        mismatch_count = np.sum((true_labels == true_label) & (cluster_labe
        cost_matrix[i, j] = -mismatch_count

# Find the best match through the least cost match
row_ind, col_ind = linear_sum_assignment(cost_matrix)

# Category re-labeling based on matching results
mapped_labels = np.zeros_like(cluster_labels)
for true_label, cluster_label in zip(unique_true_labels[row_ind], unique_clu
    mapped_labels[cluster_labels == cluster_label] = true_label

# Correct rate of calculation
accuracy = accuracy_score(true_labels, mapped_labels)

# Output result
print("Mis-clustering error rate:", 1-accuracy)
```

```
Mis-clustering error rate: 0.1267085334318434
```

Problem 4

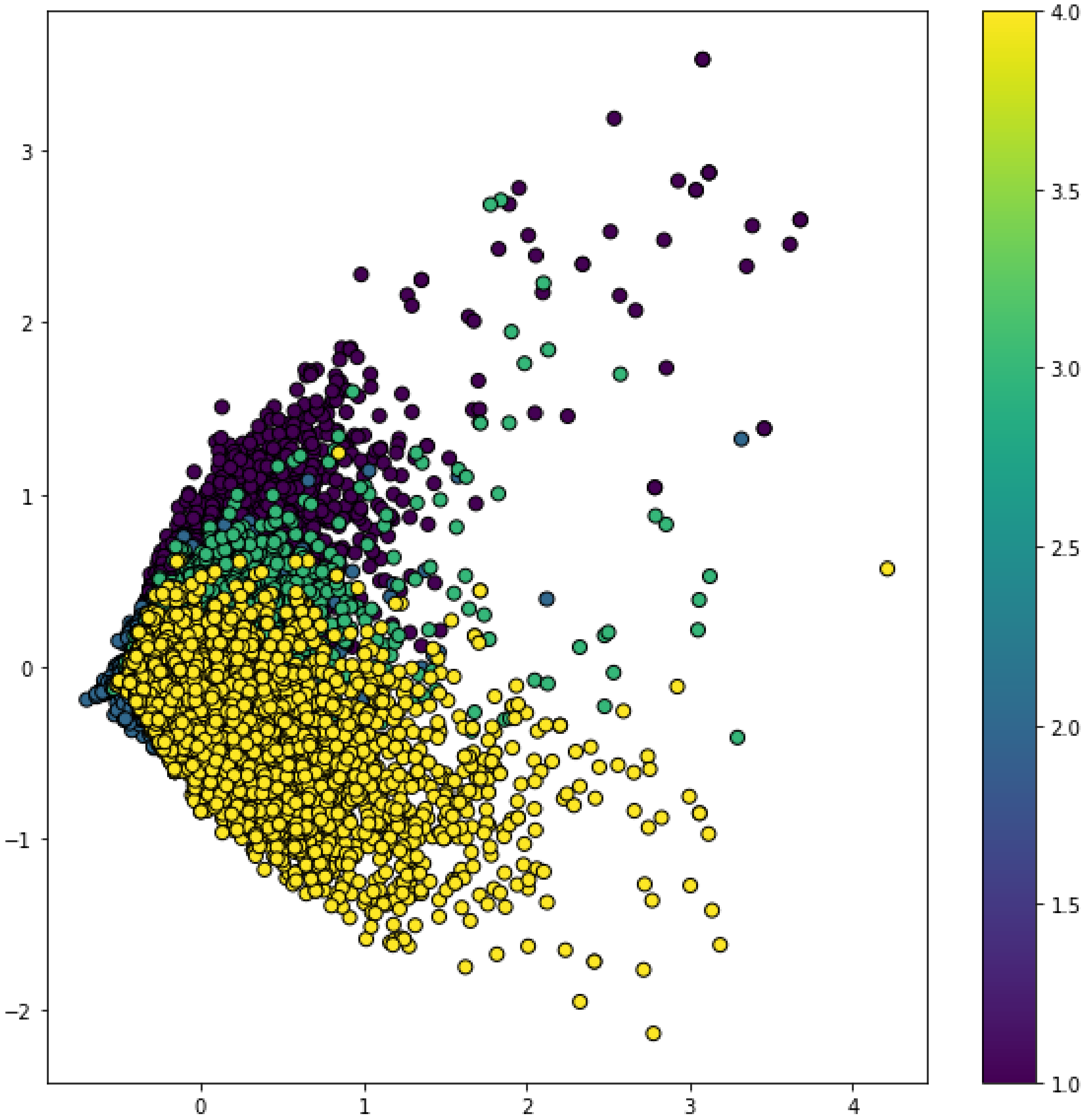
Under comparesion, we find that the PCA-K-means method's performance is better than that in problem2, the mis-error reaches 0.1267

Problem 5

Here shows the 2-dimensional.

In [ ]:
```
pca_2D = PCA(n_components = 2)
singular_vectors = pca_2D.fit_transform(occurrence_matrix)
```

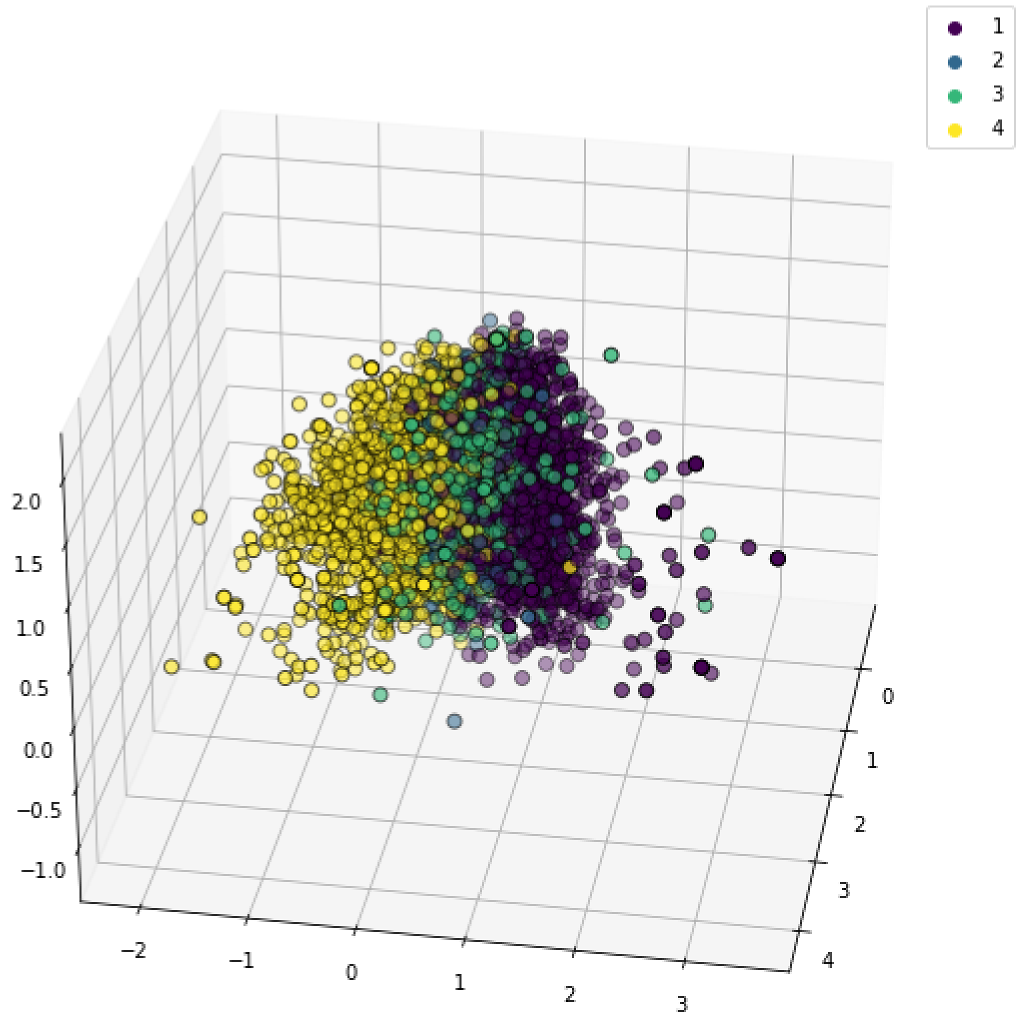
In [ ]:
```
plt.figure(figsize=(10,10))
plt.scatter(singular_vectors[:,0],singular_vectors[:,1],c = data_frame.iloc
plt.colorbar()
plt.show()
```



Here shows the 3-dimensional.

In [ ]:
```
pca_3D = PCA(n_components =3)
singular_vectors = pca_3D.fit_transform(occurrence_matrix)
from mpl_toolkits.mplot3d import Axes3D
```

In [ ]:
```
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111,projection = '3d')
sc = ax.scatter(singular_vectors[:,0],singular_vectors[:,1],singular_vectors
legend1 = ax.legend(*sc.legend_elements())
ax.view_init(elev=30, azim=10)
```



Based on the 2D and 3D principal component projection plots, it is observed that when the number of principal components is set to 3, the differences between different categories become more pronounced. This is due to the fact that a higher number of principal components can capture more information. However, despite this, there still remains a challenge in effectively separating label 2 and label 3.

Problem 4 Classification on MNIST Data

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score
import time
```

Problem 1

In [ ]:
```
train_data_frame = pd.read_csv('MNIST/train_resized.csv')
(train_data_frame) = train_data_frame.loc[train_data_frame['label'].isin([3,
train_data_frame['label'] = np.where(train_data_frame['label'] == 3, 1, 0)
num_of_train = len(train_data_frame['label'])
x_train = train_data_frame.iloc[:,1:].values
```

```
y_train = train_data_frame.iloc[:,0].values
```

In [ ]:
```
y_train
```

Out[ ]:
```
array([1, 0, 0, ..., 0, 0, 0])
```

In [ ]:
```
test_data_frame = pd.read_csv('MNIST/test_resized.csv')
(test_data_frame) = test_data_frame.loc[test_data_frame['label'].isin([3,6]
test_data_frame['label'] = np.where(test_data_frame['label'] == 3, 1, 0)
num_of_test = len(test_data_frame['label'])
x_test = test_data_frame.iloc[:,1:].values

y_test = test_data_frame.iloc[:,0].values
```

In [ ]:
```
y_test
```

Out[ ]:
```
array([1, 0, 1, ..., 1, 1, 0])
```

In [ ]:
```
np.shape(x_train )
```

Out[ ]:
```
(6026, 144)
```

In [ ]:
```
# 定义参数网格
param_grid = {'C': [1, 10, 100, 1000]}#When using grid search, GridSearchCV
svm_model = SVC(kernel='linear')#use linear kernel
grid_search = GridSearchCV(svm_model, param_grid, cv=5)#by 5 fold cross vali

grid_search.fit(x_train, y_train)
end_time = time.time()
```

In [ ]:
```
best_params = grid_search.best_params_
best_cost = best_params['C']
print(best_params,best_cost)
```

```
{'C': 1} 1
```

In [ ]:
```
good_model = SVC(kernel='linear', C=1)
start_time = time.time()
good_model.fit(x_train, y_train)
end_time = time.time()
```

In [ ]:
```
y_pred = good_model.predict(x_test)
```

In [ ]:
```
accurate = accuracy_score(y_test, y_pred)#test accurate
print("Misclassification error is {}".format(1-accurate))
```

```
Misclassification error is 0.008123476848091005
```

In [ ]:
```
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(confusion_mat)
```

```
Confusion matrix:
[[1190   10]
 [  10 1252]]
```

In [ ]:
```
print("Time cost for modeling is {}".format(end_time - start_time))
```

```
Time cost for modeling is 0.17507481575012207
```

Problem 2

In [ ]:
```
param_grid = {'C': [0.01,0.1, 1, 10], 'gamma': [0.00001,0.0001,0.001,0.01,0.
svm_model = SVC(kernel='rbf')
grid_search = GridSearchCV(svm_model, param_grid, cv=5)
```

In [ ]:
```
grid_search.fit(x_train, y_train)
```

Out[ ]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.01, 0.1, 1, 10],
                         'gamma': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 1
0]})
```

In [ ]:
```
best_params = grid_search.best_params_
best_cost = best_params['C']
best_gamma = best_params['gamma']
print(best_params,best_cost)
```

```
{'C': 1, 'gamma': 1e-05} 1
```

In [ ]:
```
good_svm_model = SVC(kernel='rbf', C=1, gamma=.00001)
start_time = time.time()
good_svm_model.fit(x_train, y_train)
end_time = time.time()
```

In [ ]:
```
y_pred = good_svm_model.predict(x_test)
```

In [ ]:
```
accurate = accuracy_score(y_test, y_pred)#test accurate
print("Misclassification error is {}".format(1-accurate))
```

```
Misclassification error is 0.0012185215272136896
```

In [ ]:
```
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(confusion_mat)
```

```
Confusion matrix:
[[1197    3]
 [   0 1262]]
```

In [ ]:
```
print("Time cost for modeling is {}".format(end_time - start_time))
```

```
Time cost for modeling is 0.9599082469940186
```

Problem 3

In solving the problem of binary image classification, the performance of these two methods is similar, but the latter consumes significantly more time compared to the former.

Problem 4

In [ ]:
```
train_data_frame = pd.read_csv('MNIST/train_resized.csv')
train_data_frame_4 = train_data_frame.loc[train_data_frame['label'].isin([1,
x_train = train_data_frame_4.iloc[:,1:].values
y_train = train_data_frame_4.iloc[:,0].values
```

In [ ]:
```
test_data_frame = pd.read_csv('MNIST/test_resized.csv')
test_data_frame_4 = test_data_frame.loc[test_data_frame['label'].isin([1,2,5
x_test = test_data_frame_4.iloc[:,1:].values
y_test = test_data_frame_4.iloc[:,0].values
```

In [ ]:
```
len(x_train)
```
Out[ ]:
```
11913
```

In [ ]:
```
param_grid = {'C': [0.000001,0.00001,0.0001]}
svm_model = SVC(kernel='linear')
```

In [ ]:
```
grid_search = GridSearchCV(svm_model, param_grid, cv=5)
```

In [ ]:
```
start_time = time.time()
grid_search.fit(x_train, y_train)
end_time = time.time()
```

In [ ]:
```
best_params = grid_search.best_params_
best_cost = best_params['C']
print(best_params,best_cost)
print(end_time-start_time)
```
```
{'C': 1e-05} 1e-05
18.024085521697998
```

In [ ]:
```
good_model = SVC(kernel='linear', C=0.00001)
good_model.fit(x_train, y_train)
```
Out[ ]:
```
SVC(C=1e-05, kernel='linear')
```

In [ ]:
```
y_pred = good_model.predict(x_test)
accurate = accuracy_score(y_test, y_pred)#test accurate
print("Misclassification error is {}".format(1-accurate))
```
```
Misclassification error is 0.046608406158968
```

In [ ]:
```
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(confusion_mat)
```
```
Confusion matrix:
[[1343   11    1    8]
 [   9 1129   25   22]
 [  15   16 1063   32]
 [  22   18   45 1047]]
```

Problem 5

In [ ]:
```
x_train = train_data_frame.iloc[:,1:].values
y_train = train_data_frame.iloc[:,0].values
```

In [ ]:
```
x_test = test_data_frame.iloc[:,1:].values
y_test = test_data_frame.iloc[:,0].values
```

In [ ]:
```
param_grid = {'C': [0.000001,0.00001,0.0001]}
svm_model = SVC(kernel='linear')
```

In [ ]:
```
grid_search = GridSearchCV(svm_model, param_grid, cv=5)
```

In [ ]:
```
start_time = time.time()
grid_search.fit(x_train, y_train)
end_time = time.time()
```

In [ ]:
```
best_params = grid_search.best_params_
best_cost = best_params['C']
print(best_params,best_cost)
print(end_time-start_time)
```

```
{'C': 0.0001} 0.0001
140.25383281707764
```

In [ ]:
```
good_model = SVC(kernel='linear', C=0.00001)
good_model.fit(x_train, y_train)
```

Out[ ]:
```
SVC(C=1e-05, kernel='linear')
```

In [ ]:
```
y_pred = good_model.predict(x_test)
accurate = accuracy_score(y_test, y_pred)#test accurate
print("Misclassification error is {}".format(1-accurate))
```

```
Misclassification error is 0.06225000000000003
```

In [ ]:
```
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(confusion_mat)
```

```
Confusion matrix:
[[1117    0    1    4    0    8    5    0    5    0]
 [   0 1342    8    5    0    0    1    1    6    0]
 [   7    3 1109    4   18    6   12    8   14    4]
 [   4    8   27 1145    0   39    2   11   16   10]
 [   3    3   11    0 1120    1    7    2    1   27]
 [   9   12    4   46    5 1010   13    1   22    4]
 [   9    1    8    0   12   11 1155    0    4    0]
 [   1    3   17    5   10    5    0 1182    1   40]
 [  10   16   13   26    6   28    4    4 1016    9]
 [   4    7    7    9   31    3    0   28    7 1057]]
```

Problem 5 Deep learning on MNIST Data

In [ ]:
```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import time
import tensorflow as tf
from tensorflow import keras

from sklearn.model_selection import train_test_split
```

In [ ]:
```
def plot_image(image):
    # Remove redundant extra dimension
    if image.shape[-1] == -1:
        image = image.squeeze(axis=1)

    plt.imshow(image, cmap="gray", interpolation="nearest")
    plt.axis("off")
```

In [ ]:
```
def plot_color_image(image):
    # Remove redundant extra dimension
    if image.shape[-1] == 1:
        image = image.squeeze(axis=1)
```

```
    plt.imshow(image, interpolation="nearest")
    plt.axis("off")
```

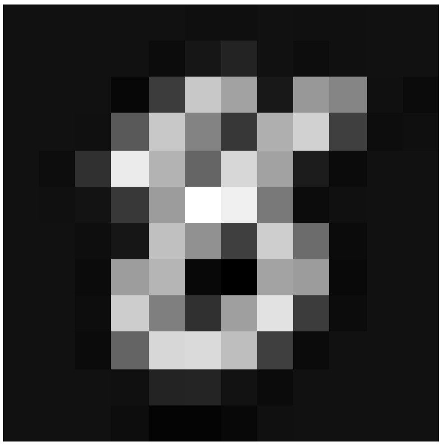
In [ ]:
```
train_data_frame = pd.read_csv('MNIST/train_resized.csv')
x_train = train_data_frame.iloc[:,1:].values
x_train = x_train.reshape(30000, 12, 12)
y_train = train_data_frame.iloc[:,0].values
print(np.shape(x_train[0]))
```

```
(12, 12)
```

In [ ]:
```
test_data_frame = pd.read_csv('MNIST/test_resized.csv')
x_test = test_data_frame.iloc[:,1:].values
x_test = x_test.reshape(12000, 12, 12)
y_test = test_data_frame.iloc[:,0].values
```

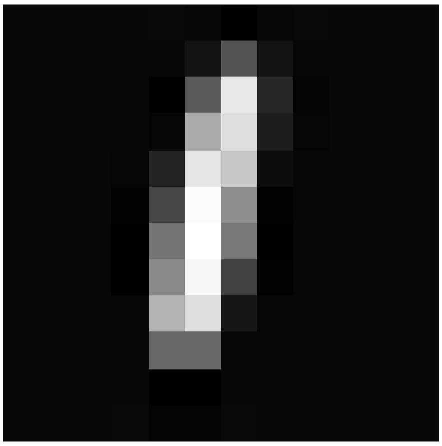
In [ ]:
```
plot_image(x_train[667])
print(y_train[667])
print(f"Shape of X_train: {x_train.shape}")
```

```
8
Shape of X_train: (30000, 12, 12)
```



In [ ]:
```
plot_image(x_test[667])
print(y_test[667])
print(f"Shape of X_train: {x_test.shape}")
```

```
1
Shape of X_train: (12000, 12, 12)
```



In [ ]:
```
x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, tra
```

In [ ]:
```
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
```

```
print(f"Shape of X_train: {x_train.shape}")
print(f"Shape of X_train: {x_test.shape}")
```

```
Shape of X_train: (27000, 12, 12, 1)
Shape of X_train: (12000, 12, 12, 1)
```

In [ ]:
```
model = keras.Sequential([

    # Specify the input shape
    keras.Input(shape=(12, 12, 1)),

    # Conv and pool block 1
    keras.layers.Conv2D(16, kernel_size=(3, 3), activation="relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),

    # Conv and pool block 2
    keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),

    # Flatten and classify using dense output layer
    keras.layers.Flatten(),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax"),
])
```

In [ ]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 10, 10, 16)        160       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 5, 5, 16)         0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 3, 3, 32)          4640      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 1, 1, 32)         0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 32)                0         
                                                                 
 dropout (Dropout)           (None, 32)                0         
                                                                 
 dense (Dense)               (None, 10)                330       
                                                                 
=================================================================
Total params: 5,130
Trainable params: 5,130
Non-trainable params: 0
_________________________________________________________________
```

In this simple CNN model, we set up two convolution layers and two pooling layers. The original image data is compressed, padding, and finally a full-connection layer is added to output the prediction result

In [ ]:
```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer="adam",
              metrics=['accuracy'])
```

In [ ]:
```
"""Running this cell could take several minutes"""
```

```
# Train the model for 10 epochs with batch size 128
batch_size = 128
epochs = 30
history = model.fit(x_train, y_train, batch_size=batch_size,
                    epochs=epochs, validation_data=(x_valid, y_valid))
```

```
Epoch 1/30
211/211 [==============================] - 1s 4ms/step - loss: 5.7852 - acc
uracy: 0.2505 - val_loss: 1.6991 - val_accuracy: 0.4530
Epoch 2/30
211/211 [==============================] - 1s 4ms/step - loss: 1.7104 - acc
uracy: 0.4136 - val_loss: 1.2198 - val_accuracy: 0.6770
Epoch 3/30
211/211 [==============================] - 1s 3ms/step - loss: 1.4614 - acc
uracy: 0.4993 - val_loss: 0.9290 - val_accuracy: 0.7530
Epoch 4/30
211/211 [==============================] - 1s 3ms/step - loss: 1.2826 - acc
uracy: 0.5559 - val_loss: 0.7560 - val_accuracy: 0.8107
Epoch 5/30
211/211 [==============================] - 1s 3ms/step - loss: 1.1760 - acc
uracy: 0.5879 - val_loss: 0.6652 - val_accuracy: 0.8437
Epoch 6/30
211/211 [==============================] - 1s 3ms/step - loss: 1.0982 - acc
uracy: 0.6109 - val_loss: 0.5549 - val_accuracy: 0.8670
Epoch 7/30
211/211 [==============================] - 1s 3ms/step - loss: 0.9987 - acc
uracy: 0.6403 - val_loss: 0.4890 - val_accuracy: 0.8880
Epoch 8/30
211/211 [==============================] - 1s 3ms/step - loss: 0.9265 - acc
uracy: 0.6584 - val_loss: 0.4398 - val_accuracy: 0.8957
Epoch 9/30
211/211 [==============================] - 1s 4ms/step - loss: 0.8731 - acc
uracy: 0.6797 - val_loss: 0.4119 - val_accuracy: 0.9007
Epoch 10/30
211/211 [==============================] - 1s 4ms/step - loss: 0.8239 - acc
uracy: 0.6937 - val_loss: 0.3692 - val_accuracy: 0.9120
Epoch 11/30
211/211 [==============================] - 1s 4ms/step - loss: 0.7826 - acc
uracy: 0.7193 - val_loss: 0.3384 - val_accuracy: 0.9157
Epoch 12/30
211/211 [==============================] - 1s 4ms/step - loss: 0.7537 - acc
uracy: 0.7360 - val_loss: 0.2983 - val_accuracy: 0.9193
Epoch 13/30
211/211 [==============================] - 1s 4ms/step - loss: 0.7040 - acc
uracy: 0.7534 - val_loss: 0.2868 - val_accuracy: 0.9307
Epoch 14/30
211/211 [==============================] - 1s 4ms/step - loss: 0.6589 - acc
uracy: 0.7705 - val_loss: 0.2642 - val_accuracy: 0.9300
Epoch 15/30
211/211 [==============================] - 1s 4ms/step - loss: 0.6372 - acc
uracy: 0.7786 - val_loss: 0.2286 - val_accuracy: 0.9370
Epoch 16/30
211/211 [==============================] - 1s 4ms/step - loss: 0.6003 - acc
uracy: 0.7913 - val_loss: 0.2223 - val_accuracy: 0.9367
Epoch 17/30
211/211 [==============================] - 1s 4ms/step - loss: 0.5852 - acc
uracy: 0.7969 - val_loss: 0.1996 - val_accuracy: 0.9440
Epoch 18/30
211/211 [==============================] - 1s 4ms/step - loss: 0.5460 - acc
uracy: 0.8101 - val_loss: 0.2002 - val_accuracy: 0.9463
Epoch 19/30
211/211 [==============================] - 1s 4ms/step - loss: 0.5271 - acc
uracy: 0.8190 - val_loss: 0.1939 - val_accuracy: 0.9450
Epoch 20/30
211/211 [==============================] - 1s 4ms/step - loss: 0.5027 - acc
uracy: 0.8239 - val_loss: 0.1833 - val_accuracy: 0.9497
Epoch 21/30
211/211 [==============================] - 1s 4ms/step - loss: 0.4968 - acc
uracy: 0.8291 - val_loss: 0.1893 - val_accuracy: 0.9487
Epoch 22/30
```

```
211/211 [==============================] - 1s 4ms/step - loss: 0.4687 - acc
uracy: 0.8390 - val_loss: 0.1710 - val_accuracy: 0.9523
Epoch 23/30
211/211 [==============================] - 1s 4ms/step - loss: 0.4616 - acc
uracy: 0.8429 - val_loss: 0.1715 - val_accuracy: 0.9533
Epoch 24/30
211/211 [==============================] - 1s 4ms/step - loss: 0.4523 - acc
uracy: 0.8464 - val_loss: 0.1597 - val_accuracy: 0.9560
Epoch 25/30
211/211 [==============================] - 1s 4ms/step - loss: 0.4376 - acc
uracy: 0.8514 - val_loss: 0.1554 - val_accuracy: 0.9560
Epoch 26/30
211/211 [==============================] - 1s 4ms/step - loss: 0.4249 - acc
uracy: 0.8556 - val_loss: 0.1496 - val_accuracy: 0.9583
Epoch 27/30
211/211 [==============================] - 1s 4ms/step - loss: 0.4236 - acc
uracy: 0.8587 - val_loss: 0.1596 - val_accuracy: 0.9553
Epoch 28/30
211/211 [==============================] - 1s 4ms/step - loss: 0.4196 - acc
uracy: 0.8599 - val_loss: 0.1497 - val_accuracy: 0.9587
Epoch 29/30
211/211 [==============================] - 1s 4ms/step - loss: 0.4011 - acc
uracy: 0.8670 - val_loss: 0.1398 - val_accuracy: 0.9610
Epoch 30/30
211/211 [==============================] - 1s 4ms/step - loss: 0.3950 - acc
uracy: 0.8691 - val_loss: 0.1462 - val_accuracy: 0.9573
```

After 30 Epoch ,we find that val_accuracy reaches 0.9573,running time for this model is about 1 min

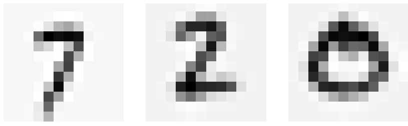
In [ ]:
```
def plot_examples(data, n_rows=4, n_cols=10):
    """Plot a grid of images which are encoded as numpy arrays."""

    # Remove redundant extra dimension
    if data.shape[-1] == 1:
        data = data.squeeze(axis=-1)


    # Size figure depending on the size of the grid
    plt.figure(figsize=(n_cols * 1.2, n_rows * 1.2))

    for row in range(n_rows):
        for col in range(n_cols):

            # Get next index of image
            index = n_cols * row + col

            # Plot the image at appropriate place in grid
            plt.subplot(n_rows, n_cols, index + 1)
            plt.imshow(data[index], cmap="binary")
            plt.axis('off')

    plt.show()
```

Show result

In [ ]:
```
# Sample several test examples
X_test_sample = x_test[3:6]

# Get probability of each class from model
y_proba = model.predict(X_test_sample)
y_pred = np.argmax(y_proba, axis=-1)
```

```
print(y_pred)

plot_examples(x_test[3:6], n_rows=1, n_cols=3)
```

```
[7 2 0]
```



problem2

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
```

In [ ]:
```
# Combine train and test datasets
x_combined = np.concatenate((x_train, x_test), axis=0)
y_combined = np.concatenate((y_train, y_test), axis=0)
# Normalize pixel values
x_combined = x_combined.astype('float32') / 255.0

# Flatten images
x_combined = x_combined.reshape((len(x_combined), -1))
```

In [ ]:
```
# Define Autoencoder architecture
input_dim = x_combined.shape[1]
encoding_dim = 2  # Two-dimensional representation
input_img = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(input_dim, activation='sigmoid')(encoded)
```

In [ ]:
```
# Create Autoencoder model
autoencoder = Model(input_img, decoded)

# Compile the model
autoencoder.compile(optimizer='adam', loss='mse')

# Train the Autoencoder
autoencoder.fit(x_combined, x_combined, epochs=10, batch_size=256, shuffle=
```

```
Epoch 1/10
132/132 [==============================] - 0s 2ms/step - loss: 0.0334 - val
_loss: 0.0329
Epoch 2/10
132/132 [==============================] - 0s 1ms/step - loss: 0.0325 - val
_loss: 0.0322
Epoch 3/10
132/132 [==============================] - 0s 1ms/step - loss: 0.0319 - val
_loss: 0.0317
Epoch 4/10
132/132 [==============================] - 0s 1ms/step - loss: 0.0315 - val
_loss: 0.0313
Epoch 5/10
132/132 [==============================] - 0s 1ms/step - loss: 0.0311 - val
_loss: 0.0310
Epoch 6/10
132/132 [==============================] - 0s 1ms/step - loss: 0.0309 - val
_loss: 0.0308
Epoch 7/10
132/132 [==============================] - 0s 1ms/step - loss: 0.0307 - val
_loss: 0.0306
Epoch 8/10
132/132 [==============================] - 0s 1ms/step - loss: 0.0305 - val
_loss: 0.0304
Epoch 9/10
132/132 [==============================] - 0s 1ms/step - loss: 0.0304 - val
_loss: 0.0303
Epoch 10/10
132/132 [==============================] - 0s 1ms/step - loss: 0.0303 - val
_loss: 0.0302
```

Out[ ]:

```
<keras.callbacks.History at 0x15eae2ca208>
```

In [ ]: