EN 685.621 PA2 - Noboru Hayashi

Problem 1

1: use sklearn.SelectKBest to top 10 best features from the data

```
def selectFeatures(X, y, k=10):
    print('Original shape of X: ', X.shape)
    model = SelectKBest(f_classif, k).fit(X,y)
    features = model.get_support(indices=True)
    X_new = model.transform(X)
    print(k, 'features are selected. Indices of features: ', features)
    print('The shape of new X is: ', X_new.shape)
    return X_new
```

Output:

% python p1.py

Original shape of X: (42000, 60)

10 features are selected. Indices of features: [0 2 4 6 20 21 22 40 41 42]

The shape of new X is: (42000, 10)

2:

A: from sklearn.preprocessing, use normalize

```
def normal(X):
    return normalize(X)
```

B: calculate mahalanobis distance and exclude outliers with alpha = 0.01

```
def mahalanobis_method(class_data):
    x_minus_mu = class_data - np.mean(class_data, axis=0)
    cov = np.cov(x_minus_mu.T)
    inv_covmat = np.linalg.inv(cov)
    left_term = np.dot(x_minus_mu, inv_covmat)
    mahal = np.dot(left_term, x_minus_mu.T)
    md = np.sqrt(mahal.diagonal())
    outlier = []
    C = np.sqrt(chi2.ppf((1-0.01), df=class_data.shape[1]))
    for index, value in enumerate(md):
        if value > C:
            outlier.append(index)
        else:
            continue
    return outlier, md



def removeOutliers(class_data, outlier):
    idx = set(range(len(class_data)))
```

```
    remain = list(idx - set(outlier))

    return [class_data[i] for i in remain]
```
3:

a)
```
def bayes(X, y):
    gnb = GaussianNB().fit(X,y)
    y_pred = gnb.predict(X)
    tot = len(y)
    acc = 1.0* (y==y_pred).sum() / tot
    print('Accuracy of Naive Bayes Classifier is: ', acc)
```

Output: Accuracy of Naive Bayes Classifier is: 1.0

b)
```
def lda(X, y):
    clf = LinearDiscriminantAnalysis().fit(X, y)
    y_pred = clf.predict(X)
    tot = len(y)
    acc = 1.0* (y==y_pred).sum() / tot
    print('Accuracy of LDA is: ', acc)
```

Output: Accuracy of LDA is: 0.7920062380521179

c)
```
def rbfnn(X, y):
    kernel = 1.0 * RBF(1.0)
    gpc = GaussianProcessClassifier(kernel=kernel,random_state=1).fit(X,y)
    print('Accuracy of RBFNN is: ', gpc.score(X,y))
```

Output: Accuracy of RBFNN is: 1.0

d)
```
def sup_vec_mac(X, y):
    kernel = 'poly'
    clf = SVC(kernel=kernel, gamma='auto').fit(X,y)
    y_pred = clf.predict(X)
    tot = len(y)
    acc = 1.0* (y==y_pred).sum() / tot
    print('Accuracy of SVM is: ', acc)
```

Output: Accuracy of SVM is: 1.0

4: Use 50fold cross validation to compare three combinations:

1- Bayes with normalization & outlier removal
2- Bayes with normalization
3- SVM (Poly) with normalization

```
def five_fold_1(X,y):
   clf = GaussianNB()
   scores = cross_val_score(clf, X, y, cv=5)
   print("Bayes with normalization & outlier removal: ")
   print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(),
scores.std()))

def five_fold_2(X,y):
   clf = GaussianNB()
   scores = cross_val_score(clf, X, y, cv=5)
   print("Bayes with normalization: ")
   print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(),
scores.std()))

def five_fold_3(X,y):
   clf = GaussianNB()
   scores = cross_val_score(clf, X, y, cv=5)
   print("SVM(Poly) with normalization: ")
   print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(),
scores.std()))
```

Output:
Bayes with normalization & outlier removal:
1.00 accuracy with a standard deviation of 0.00

Bayes with normalization:
0.79 accuracy with a standard deviation of 0.00

SVM(Poly) with normalization:
0.79 accuracy with a standard deviation of 0.00

5:
By comparing the results from problem 3 & 4, it seems Bayes, RBFNN, and SVM (poly) are better than LDA, and outlier removal is important for the model’s performance. The best results are achieved by Bayes with normalization & outlier removal.

Problem 2:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

if __name__ == '__main__':

   df_train = pd.read_csv('mnist_train.csv')
   labels_train = df_train['label']
   images_train = df_train.iloc[:,1:]/255.0
   images_train = images_train.to_numpy().reshape(df_train.shape[0], 28, 28, 1)


   model = models.Sequential()
   model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
   model.add(layers.MaxPooling2D((2, 2)))
   model.add(layers.Conv2D(64, (3, 3), activation='relu'))
   model.add(layers.MaxPooling2D((2, 2)))
   model.add(layers.Conv2D(64, (3, 3), activation='relu'))
   model.add(layers.Flatten())
   model.add(layers.Dense(64, activation='relu'))
   model.add(layers.Dense(10, activation='softmax'))

   print(model.summary())

   model.compile(optimizer='adam',
                 loss='sparse_categorical_crossentropy',
                 metrics=['accuracy'])
   model.fit(images_train, labels_train, epochs=5)

   df_test = pd.read_csv('mnist_test.csv')
   labels_test = df_test['label']
   images_test = df_test.iloc[:,1:]/255.0
   images_test = images_test.to_numpy().reshape(df_test.shape[0], 28, 28, 1)

   test_loss, test_acc = model.evaluate(images_test, labels_test, verbose=2)

   print(test_acc)
```

Output:
% python p2.py
Epoch 1/5
1875/1875 [==============================] - 23s 12ms/step - loss: 0.1427 - accuracy: 0.9569
Epoch 2/5
1875/1875 [==============================] - 23s 12ms/step - loss: 0.0469 - accuracy: 0.9855
Epoch 3/5
1875/1875 [==============================] - 23s 12ms/step - loss: 0.0322 - accuracy: 0.9903
Epoch 4/5
1875/1875 [==============================] - 22s 12ms/step - loss: 0.0255 - accuracy: 0.9917
Epoch 5/5
1875/1875 [==============================] - 23s 12ms/step - loss: 0.0194 - accuracy: 0.9939
313/313 - 1s - loss: 0.0268 - accuracy: 0.9912
0.9911999702453613

The CNN model's accuracy for train & test datasets reaches 99%. Specifically 99.39% for training, and 99.12% for test dataset.

Problem 3:

Pseudocode for minima, goal based agent:

```
# pseudocode for minimax algorithm
mini_max(board_state, depth, is_Ai) # return [move, score]
  # init the
  if is_Ai then
      best = [null, -inf]
  else
      best = [null, +inf]
  if (depth == 0 or gameover) then
      score = evaluate this board_state for player
      return [null, score]
  # DFS
  for each valid move m for player in board_state s do
      execute move m on s
      [move, score] = mini_max(s, depth - 1, -player)
      undo move m on s
      if is_Ai then
          # for AI, MAX-VALUE
          if score > best.score then best = [move, score]
      else
          # for player, MIN-VALUE
          if score < best.score then best = [move, score]
  return best
end
```