Homework 4 - CS348 Spring 2019

Description - In this assignment, you will run and analyze binary classification using K-nearest neighbors and multiclass classification using a fully connected neural network.

Getting Started - You should complete the assignment using your own installation of Python 3 and the packages numpy, pandas, keras, matplotlib, and seaborn. Download the assignment from Moodle and unzip the file. This will create a directory with this file, 'HW04.ipynb'.

Deliverables - The assignment has a single deliverable: this jupyter notebook file saved as a pdf. Please answer all coding and writing questions in the body of this file. Once all of the answers are complete, download the file by navigating the following menus: File -> Download as -> PDF via LaTeX. Submit the downloaded pdf file on gradescope. Alternatively, you can save the file as a pdf via the following: File -> Print Preview -> Print as pdf.

Data Sets - In this assignment, you will two datasets from the sci-kit learn repository, one on breast cancer and one on handwritten digits.

Academic Honesty Statement - Copying solutions from external sources (books, web pages, etc.) or other students is considered cheating. Sharing your solutions with other students is considered cheating. Posting your code to public repositories such as GitHub is also considered cheating. Any detected cheating will result in a grade of 0 on the assignment for all students involved, and potentially a grade of F in the course.

This academic honesty statement does not restrict you from reading official documentation or using other web resources for understanding the syntax of python, related data science libraries, or properties of distributions.

In [2]:
```
# Do not import any other libraries other than those listed here.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn as skl
import seaborn as sns
import keras

from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_breast_cancer, load_digits
```

Problem 1 - KNN

In this problem you'll use a K-nearest neighbors model to classify whether a tumor is benign or malignant in a breast cancer dataset.

In [42]:
```
# Loading data.
data = load_breast_cancer()
```

In [43]:
```
n = data['data'].shape[0]

index = [346, 446, 385,  90, 353, 333, 408, 398, 151,  11, 326, 164, 244
,
       252, 471, 208,  30, 135,  12, 472, 109, 263,  62, 504, 313, 266,
       301, 558, 460, 213, 498, 526, 444,  10, 264, 239,  44, 561, 543,
        83,  87,  32, 551, 516, 358, 218, 552, 507, 272, 141, 533,  98,
       190, 337, 243, 149,  13, 319, 285, 513, 383, 167, 295, 506, 563,
       303, 162, 507, 451, 264, 270, 332, 566, 385, 334, 562, 191, 241,
       133, 268, 453, 126, 175,  19, 114, 406, 281, 310, 323,  48, 218,
       401, 541, 510, 114, 533,  54, 374, 127, 321, 487,  89, 300, 178,
       356,  89, 430, 333,  37, 471, 486, 301, 183, 107, 164,  31,  77,
       163, 568, 359, 443, 508, 274, 324,  67, 147,  83, 332, 470,  98,
        90, 468,  95, 477, 403, 164, 189, 489, 199, 432, 212,  90,  94,
        86, 184, 319, 448, 382, 296, 323, 551, 363, 386, 273, 407, 254,
        97, 274, 459, 560, 154, 515, 559, 281,  79, 134, 451,   9, 252,
       485, 305, 245, 221, 219, 467,  37, 497, 186, 327, 311, 236, 397,
       189, 468, 334, 158, 231, 100, 542,  40, 410, 481, 359, 304, 269,
       267, 194, 144,  48,  22, 381, 288, 507, 530, 393, 207,  47, 551,
       202, 387, 239,  16, 286, 280, 553,  80, 327,  66, 452, 508, 446,
       418, 280, 426, 468, 233, 284,  51, 117,  40, 502, 410, 109,  81,
       449, 420,  10, 112,  25, 484, 315, 424,  70,  86, 229, 518, 567,
       131, 454, 422, 251, 108,  95, 154, 215, 325, 341, 414, 502, 324,
       364,  54,  82,   1, 177, 442, 144,  86, 337, 538, 453,   0, 251,
       481,  58, 451, 537, 364, 479,  44, 207, 382, 194, 487, 562,   3,
       460, 128, 278, 121, 431, 466, 361, 411, 556, 408, 370, 216, 443,
       124, 408,   9, 316, 267, 380,  29, 549, 481, 137, 439, 328, 381,
       112, 239, 141, 227, 337, 568, 173, 252, 456, 450, 257, 173, 182,
       241, 328, 489,  14, 318, 166, 191, 235,   6,   8,  43, 508, 318,
       465,  68, 488, 156, 238,  30, 484,  52, 249, 107, 259,  29, 359,
       567, 521, 346, 429, 361, 123,  64, 180, 241, 219, 196,  67, 230,
       483, 341, 410, 129,  94, 320, 172, 446, 125, 113, 451, 214, 273,
        87, 198, 217, 284, 565, 172, 328,  57, 251, 513, 463,  43, 334,
        91, 265, 547, 195, 434, 280, 371, 254, 351, 167, 190, 281, 259,
       212, 323, 206, 168,  33, 565, 234, 147, 277,  56, 262, 290, 233,
       178, 153, 423, 134,  84, 146, 358,  85, 112,  27, 563, 447, 197,
       388,  37, 294, 106,  86, 253, 192, 175, 344, 556, 553, 375,   3,
       495,  21,  73, 329, 435, 316, 463, 165, 287, 319, 429, 170, 343,
       379, 546, 209, 524,   8, 538, 483, 131, 284, 338, 460,  89, 370,
        62, 214, 429,  26, 133,  22,  13, 358, 390, 315,  24, 288, 191,
       307, 494, 459, 375, 248,  43, 461, 304, 414,  73, 277,  15, 442,
       261, 353, 465, 271, 542, 152, 431, 538, 305, 345, 132, 440, 111,
       520, 524, 366,  83, 379, 446, 472,  88,  11, 113, 366, 431, 160,
       435, 430, 519, 207, 290, 372, 513, 108, 327, 122, 395, 183, 414,
       269, 326, 329, 169, 384,   5, 522, 161, 213, 197, 532, 451, 446,
       358,  53, 373, 321, 422, 408, 441, 422, 338, 489,  10, 521, 150,
       288, 545, 341,  25, 510, 330, 365,  45, 335,  89]

x = data['data'][index]
y = data['target'][index]

x_train = x[:int(n/2)]#(x[:int(n/2)] - x.min(axis=0))/x.sum(axis=0)
y_train = y[:int(n/2)]
x_val = x[int(n/2):]#(x[int(n/2):] - x.min(axis=0))/x.sum(axis=0)
y_val = y[int(n/2):]
```

In [44]:
```
x_train[0]
```

Out[44]:
```
array([1.206e+01, 1.890e+01, 7.666e+01, 4.453e+02, 8.386e-02, 5.794e-0
2,
       7.510e-03, 8.488e-03, 1.555e-01, 6.048e-02, 2.430e-01, 1.152e+0
0,
       1.559e+00, 1.802e+01, 7.180e-03, 1.096e-02, 5.832e-03, 5.495e-0
3,
       1.982e-02, 2.754e-03, 1.364e+01, 2.706e+01, 8.654e+01, 5.626e+0
2,
       1.289e-01, 1.352e-01, 4.506e-02, 5.093e-02, 2.880e-01, 8.083e-0
2])
```

Part 1 (15 points)

Using sci-kit learn's `KNeighborsClassifier` class, fit a K-nearest neighbors classifier between features `x_train` and targets `y_train`. Without using `KNeighborsClassifier`'s `score()` method or `sklearn.metrics.accuracy_score`, compute the classification accuracy of the model on the training data, `x_train` and `y_train`, as well as the validation data, `x_val` and `y_val`.

Note: The classification accuracy is defined as N_correct / N, where N_correct is the number of correctly classified instances and N is the total number of instances.

In [45]:
```
# Part 1 Solution

# --- write code here ---
model = KNeighborsClassifier(n_neighbors=1)
model.fit(x_train, y_train)

pred1 = model.predict(x_train)
pred2 = model.predict(x_val)

count1 = 0
count2 = 0
for index in range(len(pred)):
    if pred1[index] == y_train[index]:
        count1 += 1
for index in range(len(y_val)):
    if pred2[index] == y_val[index]:
        count2 += 1
print(count1/len(pred1))
print(count2/len(pred2))
```

```
1.0
0.9017543859649123
```

Part 2 (20 points)

For every value of k between 1 and 50, fit a K-nearest neighbors classifier between features `x_train` and targets `y_train` . For each model, compute the classification accuracy of the model on the training data, `x_train` and `y_train` , as well as the validation data, `x_val` and `y_val` . Make a scatterplot with values of k on the horizontal axis and the classification accuracy on the vertical axis, using colors to distinguish between the evaluations on training and validation data.

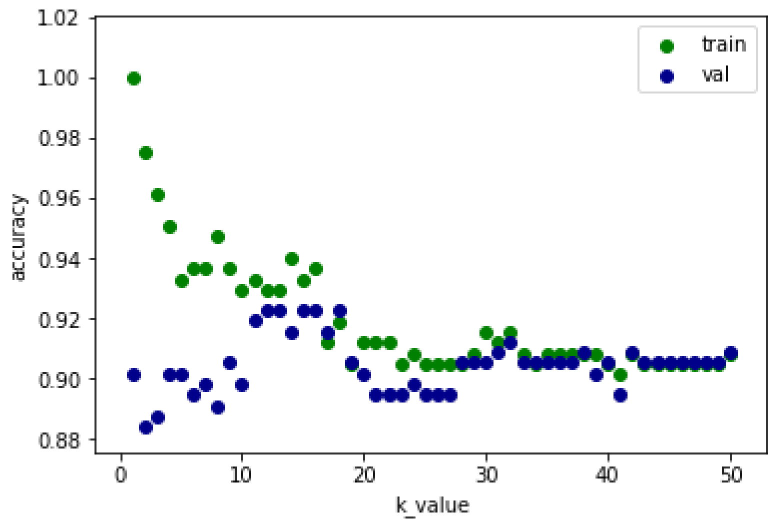
Describe how and why classification accuracy differs between the two datasets when k=1. Which value of k should we choose? Justify your answer in 2-3 sentences.

In [50]:
```
# Part 2 Solution

# --- write code here ---
train = [0]*50
val = [0]*50
for value in range(50):
    model = KNeighborsClassifier(n_neighbors=value+1)
    model.fit(x_train, y_train)

    acc1 = model.score(x_train, y_train)
    acc2 = model.score(x_val, y_val)

    train[value] = acc1
    val[value] = acc2
ls_range = range(1, 51)
plt.scatter(ls_range, train, label='train', color="green")
plt.scatter(ls_range, val, label='val', color="DarkBlue")
plt.xlabel("k_value")
plt.ylabel("accuracy")
plt.legend(loc="upper right")
plt.show()
```



Part 2 Written Response

Type your written response here

k=1 provides model the most flexible fit, which has low bias but high variance. When use k=1, the model learns training data in such a detailed way, that may be affected by outliers. Thus when predicting on x_val dataset, the overfitted model couldn't provide the accurate prediction.

The right fit of the model has a trade-off between bias and variance, thus we should choose the k values which have the same accuracy. Base on the scatterplot, we can choose k values from 42 or 50.

Part 3 (15 points)

Consider two new hypothetical datasets, `x_train_inf` and `x_val_inf` and their corresponding labels `y_train_inf` and `y_val_inf` , each of which contain an infinite number of samples from the same data generating process as parts 1-3. Disregarding computational and storage constraints, would you expect a K-nearest neighbors model trained on the `x_train_inf` and `y_train_inf` to achieve a classification accuracy of 1.0 when evaluated on `x_val_inf` and `y_val_inf` ? Justify your answer in 2-3 sentences.

Part 3 Written Response

Type your written response here

I expect a classification accuracy of 1.0 on x_train_inf and y_train_inf when set k=1, since we use these two datasets to fit our model. However, using other k values may not give us a 1.0 accuracy, high k values give us low variance but high bias, and the accuracy will be affected. For x_val_inf and y_val_inf, I don't expect a classification accuracy of 1.0, because x_val_inf and y_val_inf are not same as the training sets.

Part 4 (5 points - Extra Credit)

Your colleague claims that K-nearest neighbors is an inappropriate model for the raw data, and that the classifier could be improved by scaling each column of `x_train` to be between 0 and 1. Using your colleague's suggestion, train a new model and reproduce the plot from part 2. Explain how and why this data transformation changes the accuracy of the model.

Hint: Whatever transformation you apply to `x_train` should also be applied to `x_val` when evaluating the model.

In [7]:
```
# Part 4 Solution
# --- write code here ---
```

Part 4 Written Response

Type your written response here

Problem 2 - Neural Networks

In this problem you'll use a fully-connected neural network to classify a dataset of handwritten digits.

In [51]:
```
digit_data = load_digits()
```

In [52]:
```
n = digit_data['data'].shape[0]

x = digit_data['data']
y = digit_data['target']

x_train = x[:int(n/2)]
y_train = [keras.utils.to_categorical(y[:int(n/2)], num_classes=10)]
x_val = x[int(n/2):]
y_val = [keras.utils.to_categorical(y[int(n/2):], num_classes=10)]
```

Part 1 (30 points)

Write a function `NN_model(n)` , which returns a compiled keras neural network model with n fully connected layers, each with 64 nodes and a rectified linear unit activation function. After each fully connected layer add a dropout layer with a dropout parameter of 0.1. After the n layers, add a final fully connected layer with 10 nodes and a sigmoid activation. Compile your model with a categorical crossentropy loss, the adam optimization method, and classification accuracy as the only metric.

Hint: You may find it helpful to use the Keras sequential model API. https://keras.io/getting-started/sequential-model-guide/ (https://keras.io/getting-started/sequential-model-guide/)

In [57]:
```
# Part 1 Solution
# --- write code here ---
def NN_model(n):
    model = Sequential()
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.1))
    for index in range(1, n):
        model.add(Dense(64, activation='relu'))
        model.add(Dropout(0.1))
    model.add(Dense(10, activation='sigmoid'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', met
rics=['accuracy'])
    return model
```

Part 2 (20 points)

Train your neural network model on `x_train` and `y_train` with `n=1` for 50 epochs, a batch size of 100, and `verbose=True` . Then evaluate your trained model on `x_val` and `y_val` . Is the classification accuracy monotonic increasing with the number of training steps? Explain why this relationship is/isn't monotonic in 2-3 sentences.

In [58]:
```
# Part 2 Solution
# --- write code here ---
model = NN_model(1)
model.fit(x_train, y_train, epochs=50, batch_size=100, verbose=True)
score = model.evaluate(x_val, y_val)
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorfl
ow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.ma
th_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/50
898/898 [==============================] - 0s 392us/step - loss: 3.8029
- acc: 0.1058
Epoch 2/50
898/898 [==============================] - 0s 24us/step - loss: 2.6737
- acc: 0.1214
Epoch 3/50
898/898 [==============================] - 0s 27us/step - loss: 2.2077
- acc: 0.1670
Epoch 4/50
898/898 [==============================] - 0s 22us/step - loss: 2.1475
- acc: 0.1759
Epoch 5/50
898/898 [==============================] - 0s 23us/step - loss: 2.0799
- acc: 0.1960
Epoch 6/50
898/898 [==============================] - 0s 22us/step - loss: 2.0110
- acc: 0.2216
Epoch 7/50
898/898 [==============================] - 0s 23us/step - loss: 1.8724
- acc: 0.2773
Epoch 8/50
898/898 [==============================] - 0s 15us/step - loss: 1.6292
- acc: 0.3385
Epoch 9/50
898/898 [==============================] - 0s 19us/step - loss: 1.4432
- acc: 0.4254
Epoch 10/50
898/898 [==============================] - 0s 16us/step - loss: 1.1426
- acc: 0.5657
Epoch 11/50
898/898 [==============================] - 0s 20us/step - loss: 0.9792
- acc: 0.6837
Epoch 12/50
898/898 [==============================] - 0s 16us/step - loss: 0.7857
- acc: 0.7561
Epoch 13/50
898/898 [==============================] - 0s 14us/step - loss: 0.6746
- acc: 0.7918
Epoch 14/50
898/898 [==============================] - 0s 15us/step - loss: 0.5395
- acc: 0.8374
Epoch 15/50
898/898 [==============================] - 0s 20us/step - loss: 0.4629
- acc: 0.8597
Epoch 16/50
898/898 [==============================] - 0s 20us/step - loss: 0.4325
- acc: 0.8675
Epoch 17/50
898/898 [==============================] - 0s 16us/step - loss: 0.3269
- acc: 0.8942
Epoch 18/50
```

```
898/898 [==============================] - 0s 22us/step - loss: 0.3042
- acc: 0.9087
Epoch 19/50
898/898 [==============================] - 0s 18us/step - loss: 0.2486
- acc: 0.9198
Epoch 20/50
898/898 [==============================] - 0s 21us/step - loss: 0.2374
- acc: 0.9165
Epoch 21/50
898/898 [==============================] - 0s 23us/step - loss: 0.2202
- acc: 0.9287
Epoch 22/50
898/898 [==============================] - 0s 20us/step - loss: 0.1985
- acc: 0.9410
Epoch 23/50
898/898 [==============================] - 0s 29us/step - loss: 0.1563
- acc: 0.9510
Epoch 24/50
898/898 [==============================] - 0s 20us/step - loss: 0.1768
- acc: 0.9432
Epoch 25/50
898/898 [==============================] - 0s 19us/step - loss: 0.1378
- acc: 0.9588
Epoch 26/50
898/898 [==============================] - 0s 22us/step - loss: 0.1457
- acc: 0.9477
Epoch 27/50
898/898 [==============================] - 0s 22us/step - loss: 0.1421
- acc: 0.9465
Epoch 28/50
898/898 [==============================] - 0s 15us/step - loss: 0.1383
- acc: 0.9566
Epoch 29/50
898/898 [==============================] - 0s 17us/step - loss: 0.0994
- acc: 0.9710
Epoch 30/50
898/898 [==============================] - 0s 17us/step - loss: 0.1156
- acc: 0.9577
Epoch 31/50
898/898 [==============================] - 0s 18us/step - loss: 0.1125
- acc: 0.9610
Epoch 32/50
898/898 [==============================] - 0s 16us/step - loss: 0.0949
- acc: 0.9677
Epoch 33/50
898/898 [==============================] - 0s 22us/step - loss: 0.0856
- acc: 0.9766
Epoch 34/50
898/898 [==============================] - 0s 15us/step - loss: 0.0946
- acc: 0.9688
Epoch 35/50
898/898 [==============================] - 0s 27us/step - loss: 0.0883
- acc: 0.9755
Epoch 36/50
898/898 [==============================] - 0s 15us/step - loss: 0.0783
- acc: 0.9755
Epoch 37/50
```

```
898/898 [==============================] - 0s 20us/step - loss: 0.0764
- acc: 0.9766
Epoch 38/50
898/898 [==============================] - 0s 23us/step - loss: 0.0776
- acc: 0.9811
Epoch 39/50
898/898 [==============================] - 0s 18us/step - loss: 0.0637
- acc: 0.9822
Epoch 40/50
898/898 [==============================] - 0s 16us/step - loss: 0.0819
- acc: 0.9755
Epoch 41/50
898/898 [==============================] - 0s 19us/step - loss: 0.0683
- acc: 0.9800
Epoch 42/50
898/898 [==============================] - 0s 14us/step - loss: 0.0574
- acc: 0.9866
Epoch 43/50
898/898 [==============================] - 0s 21us/step - loss: 0.0657
- acc: 0.9788
Epoch 44/50
898/898 [==============================] - 0s 18us/step - loss: 0.0766
- acc: 0.9766
Epoch 45/50
898/898 [==============================] - 0s 17us/step - loss: 0.0592
- acc: 0.9811
Epoch 46/50
898/898 [==============================] - 0s 20us/step - loss: 0.0629
- acc: 0.9788
Epoch 47/50
898/898 [==============================] - 0s 23us/step - loss: 0.0530
- acc: 0.9844
Epoch 48/50
898/898 [==============================] - 0s 14us/step - loss: 0.0477
- acc: 0.9866
Epoch 49/50
898/898 [==============================] - 0s 20us/step - loss: 0.0533
- acc: 0.9833
Epoch 50/50
898/898 [==============================] - 0s 14us/step - loss: 0.0536
- acc: 0.9822
899/899 [==============================] - 0s 74us/step
```

Part 2 Written Response

Type your written response here

The classification accuracy is monotonic increasing with the number of training steps. Since during the training phase, backpropagation informs each neuron how much it should influence each neuron in the next layer, and finally the network converges to a state. If the classification is not monotonic, the network will unlikely converge to a state and the result could be chaotic.

Part 3 (5 points - Extra Credit)

For every integer value of n between 1 and 10, train a fully connected neural network using `NN_model(n)` on `x_train` and `y_train` and evaluate the trained model on `x_val` and `y_val`. Create a plot identical to the plot in Problem 1 part 2, except with n on the horizontal axis instead of k. Is this enough evidence to select a specific value for n? If not, what additional analysis would you suggest?

In [19]:
```
# Part 3 Solution
# --- write code here ---
```

Part 3 Written Response

Type your written response here