Week 7
Essay Discussion

Ishant Nayer
M10669373
Special Topics in BANA (002)-Python

Dataset

I am using Make_blobs dataset from sklearn's in-built datasets library.

Following are its parameters:

n_samples : int, optional (default=100)
The total number of points equally divided among clusters.

n_features : int, optional (default=2)
The number of features for each sample.

centers : int or array of shape [n_centers, n_features], optional
(default=3) The number of centers to generate, or the fixed center locations.

cluster_std : float or sequence of floats, optional (default=1.0)
The standard deviation of the clusters.

center_box : pair of floats (min, max), optional (default=(-10.0, 10.0))
The bounding box for each cluster center when centers are generated at random.

shuffle : boolean, optional (default=True)
Shuffle the samples.

random_state : int, RandomState instance or None, optional (default=None)

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

Following are the Returns that we get:

X : array of shape [n_samples, n_features]
The generated samples.

y : array of shape [n_samples]
The integer labels for cluster membership of each sample.

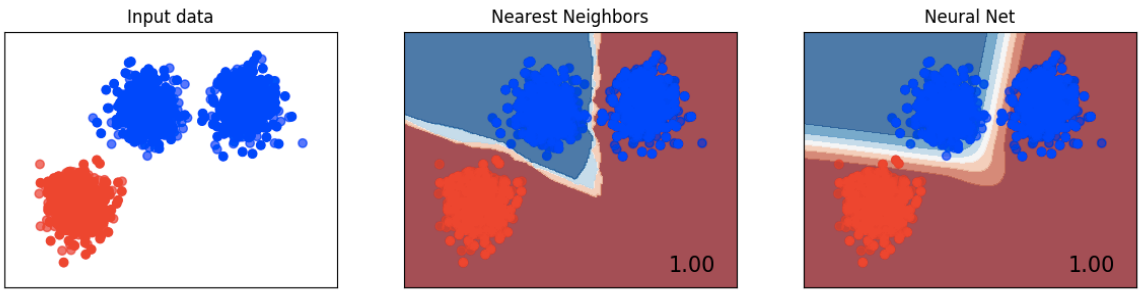
Creating K means and Neural Network

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 25 19:19:09 2017

@author: IshantNayer
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_blobs, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier

h = .02  # step size in the mesh

#Different methodologies adapted
names = ["Nearest Neighbors", "Neural Net"]

#Classifiers needed
classifiers = [
   KNeighborsClassifier(3),
   MLPClassifier(alpha=1)]

#Dataset manipulation
X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                           random_state=1, n_clusters_per_class=1)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_separable = (X, y)

datasets = [make_blobs(n_samples=1500, random_state=170),
            linearly_separable
            ]
figure = plt.figure(figsize=(27, 9))
i = 1

# iterate over datasets
for ds_cnt, ds in enumerate(datasets):
   # preprocess dataset, split into training and test part
   X, y = ds
```

```
X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=.4, random_state=42)

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# just plot the dataset first
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
if ds_cnt == 0:
    ax.set_title("Input data")
# Plot the training points
ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
# and testing points
ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6)
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.set_xticks(())
ax.set_yticks(())
i += 1

# iterate over classifiers
for name, clf in zip(names, classifiers):
    ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    if hasattr(clf, "decision_function"):
        Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    else:
        Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)

    # Plot also the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
    # and testing points
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,
```

```
            alpha=0.6)

        ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xticks(())
        ax.set_yticks(())
        if ds_cnt == 0:
            ax.set_title(name)
        ax.text(xx.max() - .3, yy.min() + .3, ('%.2f' % score).lstrip('0'),
                size=15, horizontalalignment='right')
        i += 1

plt.tight_layout()
plt.show()
```

PLOT



Explanation

- I started off with importing all the python modules which were to be used in the processing of both the models.
- Then classifiers from two models:
  1. Nearest Neighbours
  2. Neural Net

  were imported as well.
- Number of clusters were chosen as 3, alpha was taken to be 1 for Neural net MLP Classifier.
- After slicing and dicing of data we broke it into 2 formats: Train and Test data.
- Our first plot shows that we initially plotted our dataset first by taking a sample from the original dataset. It shows 3 unevenly sized clusters.

- Then other plots were made which also show that our dataset which is divided into 3 intuitive clusters despite unevenly sized blobs.
- Now, particularly in high-dimensional spaces, data can more easily be separated linearly and the simplicity of classifiers such as Neural Nets which might lead to better generalization than is achieved by nearest neighbour classifiers.
- The plots show training points in solid colours and testing points semi-transparent. The lower right shows the classification accuracy on the test set.
- The NN classifier outperformed the KNN classifier on this dataset. This does not mean that NN is universally better, just that it is more suitable for the problems presented in this case.