Department of Computer Engineering
University of Peradeniya

Machine Learning Lab Five **26-28/03/2024**

Objective

- To implement a Radial Basis Functions model on a regression problem and compare its performance with linear regression.
- To use ten-fold cross validation to quote uncertainty in empirical results when comparing the percormances of two machine learning approaches.
- To illustrate how an MLP forms approximations to a Bayes optimal classifier.

1 Radial Basis Functions (RBF)

The RBF model is given by

$$g(\boldsymbol{x}) = \sum_{j=1}^{M} \lambda_j \, \phi(||\boldsymbol{x} - \boldsymbol{m}_j|| \,/\, \sigma).$$

The model has a nonlinear part (with $\boldsymbol{m}_j$ and σ as parameters within a basis function $\phi(.)$) and a linear part with parameters λ_j. In problems where we can make sensible choices of the nonlinear part, the learning problem reduces to a linear problem in the λ_js.
Constructing an $N \times M$ *design matrix* U with terms $u_{ij} = \phi(||\boldsymbol{x}_i - \boldsymbol{x}_j|| \,/\, \sigma$, and the targets in an $N \times 1$ vector $\boldsymbol{f}$, the least squares solution to estimate the unknown λ's is similar to linear regression: $\boldsymbol{l} = \left(U^t \, U\right)^{-1} U^t \, \boldsymbol{f}$, where $\boldsymbol{l}$ is an $M \times 1$ vector containing the unknown λs.

1. Study the skeleton implementation of a Gaussian RBF model given in the Appendix.
2. Make the following improvements to the given implementation:
   - Normalize each feature of the input data to have a mean of 0 and standard deviation of 1.
   - The width parameter of the basis functions σ is set to be the distance between two randomly chosen points. Could this sometimes cause an error? Change it to be the average of several pairwise distances.
   - The locations of the M basis functions $\boldsymbol{m}_j$ are set at random points in the input space. Cluster the data using K-means clustering (with $K = M$) and set the basis function locations to the cluster centres.
   - Split the data into training and test sets, estimate the model on the training set and note the test set performance.
3. Implement ten-fold cross validation, where you split the data into ten parts, train on nine tenths of the data and test on the held out tenth set, repeating the process ten times.
4. Display the distributions of test set results for the RBF and Linear Regression Models as boxplots side by side.

2 Multi-Layer Perceptron (MLP)

The Multi-Layer Perceptron (MLP) is a widely used architecture for pattern classification and non-linear regression. It is far more flexible than the linear and RBF models but comes with difficulties in training. In this task, you are asked to study how well an MLP approximates posterior probabilities of a classification problem.

Snippets of code to generate a multi-class classification problem in two dimensions is given in the Appendix.

1. Set up two classification problems, one relatively easy to learn (i.e.) the classes are far apart and another in which the classe overlap (say, approximately 20% of the data overlap) and difficult to learn. Each class may be either a Gaussian or a Mixture of Gaussians.
2. Split the data into training and test sets and implement a Bayesian classifier and an MLP classifier and compare their performances. Your answer should be in the form of two boxplots obtained by ten-fold cross validation.
3. For one of the partitions (of training - test split), plot the class boundaries from the Gaussian and MLP classifiers. Compare a very simple MLP (with a very small number of hidden nodes) and a complex one.
4. In the piece of code given, the MLP classifier is being called with default settings. Running this will hint at all the possible parameters one is free to set and attempt to improve the model being built as follows:

```
MLPClassifier(activation='relu',
    alpha=0.0001,
    batch_size='auto',
    beta_1=0.9,
    beta_2=0.999,
    early_stopping=False,
    epsilon=1e-08,
    hidden_layer_sizes=(100,),
    learning_rate='constant',
    learning_rate_init=0.001,
    max_iter=200,
    momentum=0.9,
    nesterovs_momentum=True,
    power_t=0.5,
    random_state=None,
    shuffle=True,
    solver='adam',
    tol=0.0001,
    validation_fraction=0.1,
    verbose=False,
    warm_start=False)
```

   Read in the documentation what the role of each of the parameters is, and show the effect of changing any two of them in your report (e.g. How does convergence change of you choose a different value for the `learning_rate_init` parameter?).

Mahesan Niranjan February 2024
Material adapted from module Foundations of Machine Learning, taught at the School of Electronics and Computer Science, University of Southampton.

Appendix: Skeleton Implementation of RBF

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

from sklearn import datasets
from sklearn.linear_model import LinearRegression

diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

def gaussian(x, u, sigma):
    return(np.exp(-0.5 * np.linalg.norm(x-u) / sigma))

N, p = X.shape
print(N, p)

# Space for design matrix
#
M = 200
U = np.zeros((N,M))

# Basis function locations at random
#
C = np.random.randn(M,p)

# Basis function range as distance between two random data
#
x1 = X[np.floor(np.random.rand()*N).astype(int),:]
x2 = X[np.floor(np.random.rand()*N).astype(int),:]
sigma = np.linalg.norm(x1-x2)

# Construct the design matrix
#
for i in range(N):
    for j in range(M):
        U[i,j] = gaussian(X[i,:], C[j,:], sigma)

# Pseudo inverse solution for linear part
#
l = np.linalg.inv(U.T @ U) @ U.T @ y

# Predicted values on training data
#
yh = U @ l
fig, ax = plt.subplots(figsize=(3,3))
ax.scatter(y, yh, c='m', s=3)
ax.grid(True)
ax.set_title("Training Set", fontsize=14)
ax.set_xlabel("True Target", fontsize=12)
ax.set_ylabel("Prediction", fontsize=12)
```

Approximating Bayes Posterior: Data

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

def genGaussianSamples(N, m, C):
    A = np.linalg.cholesky(C)
    U = np.random.randn(N,2)

    return(U @ A.T + m)

NClasses = 3

# Priors
#
w = np.random.rand(NClasses)
w = w / np.sum(w)
N = 1000  # total data (Training = Test)
NPrior = np.floor(w * N).astype(int)

Scale = 10
Means = Scale*np.random.rand(NClasses, 2)

from sklearn.datasets import make_spd_matrix
CovMatrices = np.zeros((NClasses,2,2))
for j in range(NClasses):
    CovMatrices[j,:,:] = make_spd_matrix(2)

AllData_train = list()
for j in range(NClasses):
    AllData_train.append(genGaussianSamples(NPrior[j], Means[j,:], CovMatrices[j,:,:]))

X_train = AllData_train[0]
y_train = np.ones((NPrior[0], 1))
for j in range(NClasses-1):
    Xj = genGaussianSamples(NPrior[j+1], Means[j+1,:], CovMatrices[j+1,:,:])
    X_train = np.append(X_train, Xj, axis=0)
    yj = (j+2)*np.ones((NPrior[j+1], 1))
    y_train = np.append(y_train, yj)

AllData_test = list()
for j in range(NClasses):
    AllData_test.append(genGaussianSamples(NPrior[j], Means[j,:], CovMatrices[j,:,:]))

X_test = AllData_test[0]
y_test = np.ones((NPrior[0], 1))
for j in range(NClasses-1):
    Xj = genGaussianSamples(NPrior[j+1], Means[j+1,:], CovMatrices[j+1,:,:])
    X_test = np.append(X_test, Xj, axis=0)
    yj = (j+2)*np.ones((NPrior[j+1], 1))
    y_test = np.append(y_test, yj)


fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(12,4))
plt.subplots_adjust(wspace=0.3)
for j in range(NClasses):
    Xplt = AllData_train[j]
    ax[0].scatter(Xplt[:,0], Xplt[:,1], s=3)
ax[0].grid(True)
ax[0].set_title("Training Data Distributions")

ax[1].plot(y_train)
ax[1].set_title("Training Targets")

for j in range(NClasses):
    Xplt = AllData_test[j]
    ax[2].scatter(Xplt[:,0], Xplt[:,1], s=3)
ax[2].grid(True)
ax[2].set_title("Test Data Distributions")
```

Approximating Bayes Posterior: MLP Training

```
# Encoding the output
#
from sklearn.preprocessing import OneHotEncoder

onehot_encoder = OneHotEncoder(sparse=False)
y_onehot_train = onehot_encoder.fit_transform(y_train.reshape(-1, 1))

# Training a neural network
#
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier()
clf.fit(X_train, y_onehot_train)

# Predictions, accuracy and confusion matrix
#
from sklearn.metrics import accuracy_score
y_pred_train = clf.predict(X_train)
print(accuracy_score(y_onehot_train, y_pred_train))

N_train = X_train.shape[0]
predicted_class_train = np.zeros((N_train,1))
for j in range(N_train):
    predicted_class_train[j] = (1+np.argmax(y_pred_train[j,:])).astype(int)

from sklearn.metrics import confusion_matrix
print("Confusion Matrix: ")
print(confusion_matrix(y_train, predicted_class_train))
```