**Implementing binary classifiers based on Neural Networks and Nearest Neighbor methods.**

CS 260A - Machine Learning Algorithms

Final Project Report**1**

Amogh Param (704434779)

University of California, Los Angeles

[aparam@cs.ucla.edu](mailto:aparam@cs.ucla.edu)

**Introduction**

**Classification**

**Neural Networks**

**k-Nearest Neighbors**

**Feature Selection**

**Unique Contribution**

**Using Unique features**

**Testing on the IRIS dataset**

**Data preparation.py**

‘DataPreparation.py’ is a class that extracts and structures the given datasets in a format that can be easily handled by the binary classifiers. Additionally, this class is responsible for creating ‘Patient’ objects (described below), assigning them their classification labels and for also calling the feature computation methods of the ‘Patient’ instances.

**Patient.py**

‘Patient.py’ defines the class for the patient data provided in the dataset. Instantiating this class is equivalent to creating an object record for a patient and assigning each of the instance variables to measurements, features and labels that uniquely identify that patient. This class also has methods to compute features like mean, variance, skewness, kurtosis, etc. for each patient.

**Classifier.py**

‘Classifier.py’ is the base class for the 2 binary classifiers (Neural Network and Nearest Neighbor). It provides methods for the required performance evaluation like, computing the classification confusion matrix (TP, FP, TN, FN), accuracy, sensitivity, specificity, precision, recall, f-measure and the ROC characteristics.

**NeuralNetwork.py**

‘NeuralNetwork.py’ is a class that inherits from ‘Classifier.py’ and therefore has all the methods to evaluate its own performance for the given dataset. The Neural Network class is designed to have one input layer, one hidden layer and one output layer, but the number of neurons/nodes in each layer can be changed as required. The input layer includes a bias node. A sigmoid activation function is used for the activations in each layer. Specifically, the hyperbolic tangent function is used. I previously tried experimenting with the exponential sigmoid function but I ran into many ‘overflow’ problems. The hyperbolic tangent function provides an output in the range of -1 to 1; therefore the outputs are scaled to be in the range 0 to 1 (since the problem we’re interested in is binary classification). The weights for the neural networks are initialized to random values before every training phase. One of the primary methods in this class is the ‘backPropagation’ method, which calculates the error between the actual output and the predicted output and updates the weights between nodes of different layers (back propagates the error). A training method predicts the output for every input set and updates weights based on the back-propagated error. Finally a classify method computes the class label that the input set should be classified into.

By changing the parameters in the ‘neuralNetworkConfig’dictionary, we can test the neural network under many different configurations and find the best performing configuration.

**NearestNeighbor.py**

Similar to the neural network described above, ‘NearestNeighbor.py’ inherits from ‘Classifier.py’. In addition to the evaluation methods, this class provides methods to compute the distance between a reference data point (feature vector) and all the other training data points. The ‘findKNearestNeighbors**’** method looks at all the training data points and finds the top k nearest data points with respect to the reference point from the testing set that needs to be classified. The ‘computeClass’ method then looks at the majority class label of the neighbors and assigns that as the class label of the reference data point. This process of classification is done for all the reference data points in the testing set and the final classification is achieved and evaluated.

**Results**

**Dataset 1**

**Dataset 2 (final)**