INDIAN STATISTICAL INSTITUTE

Computing for Data Sciences (PGDBA)

July - December 2018

Assignment V

Deadline: November 04, 2018.

---

1. *Neural Networks* - In the accompanying Jupyter notebook, you will find a simple implementation of a Multi-Layered Perceptron (MLP). The following are the details of the implementation:

   - The MLP consistes of 3 layers: (i) An Input Layer (ii) A Hidden Layer (iii) An Output Layer.
   - The number of neurons in the Hidden Layer is `size_h`+1 , where `size_h` can be set by the user. The number of neurons in the Input Layer is $d+1$, where d is the dimension of the input data. The number of neurons in the Output Layer is c, where c is the total number of classes.
   - The activation function used for every neuron in the Hidden and Output Layers is the sigmoid function $\sigma(z) = \dfrac{1}{1+e^{-z}}$.
   - The cost function that the MLP tries to minimize is $\dfrac{1}{2n}\sum_{i=1}^{n} ||o^{(i)} - y^{(i)}||^2$. Here, $o^{(i)}$ is the output of the network for input data $x^{(i)}$, and $y^{(i)}$ is the one-hot encoded class label vector corresponding to data point $x^{(i)}$.
   - Gradient descent is used to estimate the network parameters, with a constant step size of 0.1.
   - There are two ways in which an MLP can estimate all its parameters. In *online learning*, the network parameters are updated for every data point $x^{(i)}$, i.e., for every $x^{(i)}$, the gradient of the cost with respect to all parameters are computed, after which the parameters are immediately updated using gradient descent. The second way in which the parameters can be estimated is by *batch learning*, where the average gradients computed from the entire data set are used to update the parameters. In the accompanying Jupyter notebook, *batch learning* is used.

   You can go through the implementation to see how the theory translates to the code. Then solve the following assignment:

   (i) Modify the implementation of MLP to train the following network:
   - Use the Rectified Linear Units (ReLU) activation function on the neurons of the hidden layer. The ReLU function is $\text{ReLU}(z) = \max(0, z)$.
   - Keep the sigmoid activation function on the neurons of the output layer.
   - Use the mean cross entropy as the cost function, defined as,

   $$-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\{y_j^{(i)}\log(o_j^{(i)}) + (1-y_j^{(i)})\log(1-o_j^{(i)})\}$$

   (*Hint*: A lot of expressions cancel out, when the derivative of this cost function and the sigmoid activation function are calculated in the chain rule of derviatives.)

   (ii) `sklearn` contains an implementation of Multi-Layered Perceptrons (`sklearn.neural_network.MLPClassifier`). Use `sklearn`'s implementation to train an MLP on the Digits data set (present at `sklearn.datasets.load_digits`). The Digits data set contains 1797

images of handwritten digits 0 to 9. Each image is an 8×8 gray-scale image, present in the data set as a vector of length 64.

- Train an MLP with one hidden layer containing 100 nodes.
- Use ReLU as the activation function.
- Split the data set into 60% training and 40% test data sets.
- From the test results, display 5 correctly classified images, and at most 5 incorrectly classified images.

2. Learning *Decision Trees* - A Decision Tree classifier builds a binary search tree to learn to classify a data set.

   - At each node of the tree, a preset criterion is used to select a feature f and a threshold δ for that feature' values. Data points x_i with $f(x_i) \leq \delta$ are passed to the left child of that node, and data points with $f(x_i) > \delta$ are passed to the right child of that node.
   - The objective of these *splitting* of the data sets is to reach nodes where all data points belong to the same class.
   - Criterions such as the *entropy* of the data set can be used to split the data sets at each node.

   In the accompanying Jupyter notebook, an example is presented of how to train a Decision Tree (present in `sklearn.tree.DecisionTreeClassifier`). The Decision Tree is trained on the Iris data set. How can the learned binary search tree be visualized? In the notebook, we are using the `graphviz` library to display the entire learned tree. `graphviz` is not installed in Anaconda by default, you can install in from the Anaconda prompt by executing the following command.

   ```
   conda install python-graphviz
   ```

   Running the example in the Jupyter notebook will show you the learned Decision Tree. You can go through the code to see how the Decision Tree is trained, and how `graphviz` is used to display the learned tree.

   Your assignment - Train a Decision Tree on the Wisconsin Breast Cancer data set (available in `sklearn.datasets.load_breast_cancer`). Display the learned Decision Tree. The Wisconsin Breast Cancer data set contains the 30 features, which are the medical information of 569 anonymous patients, some of which have malignant tumours while others do not. The binary classification problem on this data set is to accurately identify patients with malignant tumours.

3. Learning *Random Forests* - In this exercise, you will train a Random Forest comprised of Decision Trees.

   - Train a Random Forest with 10 Decision Trees (present in `sklearn.ensemble.RandomForestClassifier`) on the data set present in `synth_data1.txt`.
   - Test data x_{test} is generated in the Jupyter notebook. Predict class labels y_{test} for the test data, and use it to visualize the decision boundary created by Random Forests.
   - Save each Decision Tree using `graphviz`, so that each learned tree in the forest can be visualized.

4. Compare the classification performance of Random Forests, Multi-Layered Perceptrons, and Linear Support Vector Machines on the Digits data set. Use 10-fold cross-validation and measure the average accuracy.

5. *Data Visualization* - Visualizing data sets can inform you about the underlying structures present in the data set. However in the real world, it is quite common to come across high-dimensional data sets, which are impossible to visualize completely. There exists numerous summarizing statistics (mean, variance, etc.) however they often do not capture the complex structures that might be present in the data set. An alternative approach that is often

quite useful, is to use a data dimension reduction technique to project the high-dimensional data to 2 or 3 dimensions, then visualize it. We will look at two such methods, the familiar Principal Component Analysis (PCA), and a method called t-distributed Stochastic Neighbor Embedding (t-SNE).

(i) The Digits data set has 64 features. Use PCA to reduce the data set to 2 dimensions, then visualize the transformed data. A helper plotting function has been included in the Jupyter notebook to color each class separately.

(ii) t-SNE is one of several *manifold*-based data reduction techniques. Manifold-based data reduction techniques project data to lower dimension spaces, while preserving the *local neighbourhood information* in the original space. This implies that points that are close to each other in the original space, should be close in the reduced space. The inverse of the statement holds as well, points that are far apart in the original space should have a large distance between them in the reduced space.

Use t-SNE (present in sklearn.manifold.TSNE) to reduce the data to 2 dimensions. Use the helper plotting function in the accompanying Jupyter notebook to visualize the reduced data. Note the differences between the transformations by TSNE and PCA.