EECS/MSAI 349 Problem Set 4

You can work in teams of 2-3 on this assignment, but each student should independently prepare and submit their own write-up (i.e., we would expect the write-ups to be somewhat different). Submit a single zip file containing:

Ps4.pdf, giving the answers to the questions
Q1.csv, a comma-separated-values file for question 1 below
Q2.csv, a comma-separated-values file for question 2 below
Ps4.py, containing your python code for part II of the assignment

Part I

1. In the second homework, you were asked to try to find two algorithms that gave very different performance on a given data set. In this homework question, you'll instead try to create a data set that gives very different performance for two given algorithms. Specifically, can you create a data set of 1000 examples with at most 1000 attributes such that Weka gives very different 10-fold cross-validation performance (in absolute value) for nearest neighbor (IBk) vs. decision trees (J48)? You only need to create one data set, for which one of the two algorithms wins. You can choose which of IBk or J48 you want to be the winner.

   How large is "very different"? Here is what to aim for in this question and in Question 2:
   - 10%+ difference in accuracy: half credit
   - 20%+ difference in accuracy: 75% credit
   - 30%+ difference in accuracy: full credit

   (however, 40%+ differences are definitely possible for both questions)

   The above are absolute differences, e.g. |J48 accuracy – Ibk accuracy|.

   You should output your data set in CSV format (Weka allows you to open and classify CSV data files, so this format will be convenient).

   Important: you should use the default settings in Weka for each algorithm -- you can find IBk under "lazy" in the "classifiers" section. Report how you generated the data set (including the code if applicable) and a table giving the 10-fold CV accuracy of each algorithm on your data. Include a 2-3 sentence explanation for why the classifiers perform as they do on your data set. Include your data set as q1.csv in your zip file. **(2pt)**

2. Repeat Question 1, except using multi-layer perceptrons (found under "functions" in Weka) vs. Naive Bayes classifiers (found under "bayes" in Weka). Submit your data set as q2.csv in your zip file.

   Yes, you *can* use the same data set in Question 1 as in Question 2, if you like. **(1pt)**

3. Define a data set with four binary attributes and a binary output, such that Naïve Bayes achieves only 2/3 training accuracy but Logistic Regression achieves 1.0 training accuracy. You should use add-one smoothing for Naïve Bayes (for both P(Class) and P(Attribute_i | Class)), and no regularization for logistic regression. Justify your answer. **(1pt)**

4. Consider the following four random variables, which can take values in the given sets:
   Hours of Sleep: {0, 1, 2, 3, 4, 5, 6, 7, 8}
   Studied: {None, Some, Lots}
   LikesMaterial: {None, Some, Lots}
   ExamScore: {A, B, C}

   (a) How many *independent parameters* are needed to specify the joint distribution over these four variables? A joint distribution defined over *k* disjoint events requires *k*-1 independent parameters. **(1/3pt)**
   (b) Now assume you want to model the conditional distribution, P(ExamScore | HoursOfSleep, Studied, LikesMaterial). How many independent parameters does that conditional distribution contain? **(1/3pt)**
   (c) Now imagine modeling P(ExamScore | HoursOfSleep, Studied, LikesMaterial) with the Naïve Bayes assumption, that each of the three variables on the right-hand-side of the pipe is conditionally independent given ExamScore. How many independent parameters are needed to specify the conditional distribution under the Naïve Bayes assumption? **(1/3pt)**

Part II

Start by getting the code and data files.

5. Implement a function named "cosine_similarity" that computes the cosine similarity (a number in [-1.0, 1.0]) between two vectors (represented as numpy arrays or list of numbers). Turn in your code in a file ps4.py. **(1pt)**

6. If we can represent the things we want to compare as vectors, we can use cosine similarity to check how similar they are. For images, a naive choice of representation is to use the RGB value of each pixel as one vector (for example, if the image is 200x100 pixels, the vector representation is 3x200x100). A recent revolution in computer vision is the use of convolutional neural network, which is shown to be able to learn better representations of images in its hidden layers. VGGNet is one of the deep convolutional neural networks used for object recognition. In this part, we will compare the representation of one of the hidden layers in VGGNet with the naive pixel representation.

   Use the function you implemented in #4 to compute cosine similarity between each pair

of the three images under the "cnn" folder. You should use the VGG ("vgg_rep") and pixel representation ("pixel_rep") of each image ("mj1", "mj2", "cat") saved in "cnn_dataset.json".

Report the similarity between each pair using VGG representation and pixel representation respectively (so you should report 6 numbers in total). Which pair is the most similar in VGG representation and which pair is the most similar in pixel representation? **(1 pt)**

7. Based on the result in #5 above, what problem does pixel representation have and why does the convolutional neural network fix it? **(1 pt)**

8. Nearest neighbor can not only be used for classification or regression, but also generating captions for images. The basic idea is simple: you just find the most similar image in your training set, and use its caption as the answer.

   We will use a toy dataset included in the handout "dataset.json" to demonstrate the idea and further compare VGG and pixel representation. (Note that good and robust results require a much larger dataset.)

   The structure of the dictionary saved in "dataset.json" is:
   "train": a list of image names in the training set.
   "test": a list of image names in test set.
   "images": a list of all the image names.
   "captions": a dictionary saving the corresponding captions for each image (using image name as key).

   All the images are saved in "images/" folder to examine.

   The VGG representation and pixel representation are saved in "vgg_rep.npy" and "pixel_rep.npy" (take a look here about how to load the data). Each file saves a numpy matrix. Each row of the matrix corresponds to a vector representation of a image. The order of the rows is the same as the image names in the "images" field of "dataset.json".

   Save your prediction result of the test set in two text files (one caption per line, in the same order that the image name appears in "test" field of "dataset.json"), vgg.txt and pixel.txt. Include the code you wrote to find the nearest neighbor images and captions in ps4.py. Compare the two results from VGG representation and pixel representation – is one better than the other? Justify your answer in 1-2 sentences. **(1pt)**

9. A machine learning model is usually not perfect, and analyzing its errors can usually tell us more about how the model works. From the captions that doesn't describe the image well, pick one example that uses VGG representation and one that uses pixel

representation. Include the image and the generated caption, and describe why you think it generates that wrong caption. **(1 pt)**