Homework 1

DSE I2100, Spring 2020

Michael Grossberg

1. Submit gmail account
2. Submite github account
3. Complete the Data Camp course: Linear Classifiers in Python submit the certificate of completion to blackboard.
4. Complete the Data Camp course: Writing Efficient Python Code
5. Suppose we have two populations of beans. The weights of these beans are normaly distributed, so if μ is the mean weight of one type of beans beans and σ is the standard deviation, so that means that the probabilty density is given by

$$p_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma}(x-\mu)^2}$$

and then the probabilty of measureing a weight in interval $I = [x_1, x_2]$ of bean of that type is given by

$$P_{\mu,\sigma}(I) = \int_{x_1}^{x_2} p_{\mu,\sigma}(x)dx$$

The mean weight of bean type A, μ_A is 5 grams and the standard deviation, σ_A, is 2. The mean weight of bean type B, mu_B, is 4 grams and has a standard deviation σ_B, of 1.4.

Our classifier $C_T(x)$ is determined by a weight threshold T:

$$f\_T(x) = \begin{cases} -1 & x \leq T \\ 1 & T < x \end{cases}$$

The *Bayes error* is the probability that we will misclassify. Assume that there are equally many beans of each type (no prior).

   a. For a given T write down the theoretical expression (in terms of integrals) for the probability that you will classify a point as bean type A when it is bean type B and similarly that it is bean type B when it is bean type A.

   b. In python just using numerical functions (you can take 1000 data points from min of weight x=1 to x=8) compute the theorical probailities from part a. Use matplotlib to make a curves showing the

probability of classifying something class A ($\hat{C} = A$)assuming it is really class B $C = B$, in other words $P(\hat{C} = A|C = B)$ is the figure y-axis, as a function of T, the figure x-axis. Similarly plot $P(\hat{C} = B|C = A)$ as a fucntion of T. Putting these together since $P(A) = P(B) = 1/2$, adding the curves and dividing by 2 you get the probability of miss-classification or Bayed error as a function of T. Plot that as well.

c. Use the numpy random.randn to simulate 10000 data points, 5,000 from bean type A and 5,000 from bean type B. You can now pick 1000 values of T using linspace between $T = 1$ and $T = 8$. For each of these you can compute the miss classification rate. Make the figure. These should match closely your results for b above.

This problem should be a jupyter notebook with markdown explaining each of your steps. In all cases seed your random numbers for reproducebility.

6. We have discussed making fake data many times. We will look at this problem for dimension = 2. First we will use sklearn. Assume we have two classes A and B. Data from class A is normally distributed in $2D$ around the mean $\mu_A = (-1, -1)$ with standard deviation $\sigma_A = 1/8$. Data from class A is normally distributed in $2D$ around the mean $\mu_B = (1, 1)$ with standard deviation $\sigma_B = 1/8$. Use the sklearn function "sklearn.datasets.make_blobs" so make these blobs. In all these cases seed your random numbers.

   a. Assume there are 250 class A points, and 250 class B points. Use matplot lib to plot these points. These clusters should be very well separated and any classifier should do very well on them. Please plot the blobs with class A in blue and class B points as in red. Not that you have overplotting so you cannot see that your blobs are much denser in the middle then at the edges. Decrease the size of the dots and also use a small transparency $alpha << 1$ untill you can clearly see dots in the core as been denser. Always beware of overplotting.

   b. You may not always have access to sklearn. Readd your results in a. using only numpy.random.randn. Without using sklearn you should be able to achive the same results, albiet with more lines of code.

   c. Going back to using sklearn for generating data sets we will now make the data sets less well separated. Increase the standard deviation for A and B to $\sigma_A = \sigma_B = 4$. Again plot the results and becareful with overplotting. It should be clear here that any classifier, no matter how clever, will get very poor perfomance on this data.

   d. For our last example we want to create data which is clearly separable but not linearly separaable. Use sklearn.datasets.make_circles to make two circles in $2D$ each with 500 points. Unfortuately we don't want a little scircle in the middle but a normally disturbed blob. Use fancy indexing to isolate data on the outer circle only (class

B). Then generate a blob in the center (class A) which will have mean $\mu_A = (0, 0)$ and standard deviation $\sigma_A = 1/8$. Again show this data set in red and blue for the classes, showing that they are well separated.

7. In this problem you will write code. You are going to write a one file library a.k.a. a python module. Please read PEP 8 and follow the recommendations. Install Pylint and run it on your file to make sure your code has good style. Use the Numpy Docstringstyle guide and make sure that the paramters for your methods are described (don't use the short summary only.)

   In class we discussed the perceptron with its update rule. In chapter 2 we discuss logistic regression.

   a. You need to implement a class DSELinearClassifier in module "DSEHW1" that takes as a parameter on construction, the activation functions, like so:

   ```
   from DSEHW import DSELinearClassifier

   clfP = DSELinearClassifier(activation='Perceptron')
   clfL = DSELinearClassifier(activation='Logistic')
   clfH = DSELinearClassifier(activation='HyperTan')
   ```

   For the module you implement, the class DSELinearClassifier has a a constructor that takes this activation label. The constructor ("**init**") should be able to take a "random_state" parameter to input a random seed, and an "initial_weight" parameter to set the initial weight (instead of choosing it randomly). You should also be able to set a parameter called "learning_rate".

   b. Write down the update rule by computing the gradient for hyperboilc tantgent and the logistic regression activation function. Logistic regression is in the book. The update rule for the perceptron doesn't use the gradient because it is nondiverentiable. Don't forget to include the learning rate.

   c. Besides the constructor your class DSELinearClassifier should implement a fit method which, like other classifiers in slearn takes X, y to fit. Assume that y is $-1, 1$ for perceptron, $0, 1$ for logistic regression and $-1, 1$ for the hyperboic tangent. The fit should have optional parameters batch_size for (ignored in perceptron), and max_epochs which limits the number of times it goes through the data on a fit. Use the appropriate update rule based on how self.activation was set. Make sure the weights are chosen at random according to the seeding. Make sure that the update rule includes the learning rate.

   d. Store an internal array called "_fit_errors" which saves the error at each step of the fit. Append to the end of this array.

e. This should have a predict method as well that produces predictions $\hat{y}$.

f. Seed your data with seed = 42 (initial random state.) Then compute the classification report from sklearn for each of the three activations on the synthetic data sets we created in the last probelem: two blobs well separated, two blobs poorly separated and a blob and a circle. For each case show the final classifier line, with the class regions colored Red and blue as in the examples in the book, or sklearn. Color the dots as before. As usual split train/test data and show performance on both using classificatio_report for the three data sets.

g. For the three data sets show the fit plot the curve of errors with steps.

h. Try 3 different values of the learning rate each double the last. Plot the path of errors.

i. This time run the three algorithms on the data sets with well separated blobs but initialize the weights to be way off with "initial_weight = (5,5)". With your prior best learning rate, what do the error curves look like?