Scaffolding code is available on [Github](https://github.com/cs472ta/CS472" \t "_blank).  
Read the [tutorial](https://github.com/cs472ta/CS472/blob/master/old/TUTORIAL.md).

1. (40%) Correctly implement and submit your own code for the perceptron learning algorithm. Code Requirements:
   1. A Perceptron Class, with (at least **predict**, **fit**, **score**, and **get\_weights** methods).
   2. Shuffle the data each epoch.
   3. A way to create a random train/test split. Write your own. In the future you can use the scikit-learn version if you want.
   4. Use Stochastic/On-line training updates: Iterate and update weights after each training instance.
   5. Implement a stopping criteria: when your model has trained for a number of epochs with no significant improvement in accuracy, stop training. Note that the weights/accuracy do not usually change monotonically.
   6. Use your perceptron to solve the [Debug](https://learningsuite.byu.edu/cid-9AKBdFBQ8oNg/student/pages/id-6FTA) data. We provide you with several parameters, and you should be able to replicate our results every time. When you are confident it is correct, run your perceptron on the [Evaluation](https://learningsuite.byu.edu/cid-9AKBdFBQ8oNg/student/pages/id-6FTA) data with the same parameters, and **include a screenshot of your executed program showing your final weights in your report.**
   7. For this first lab, **all code for the functions described must be original.** The only package you may use for these methods is NumPy, with no other external calls to scikit-learn or any other ML package.
      1. Your class can still inherit from the relevant scikit-learn learners for good measure, but don't call any of the super methods to accomplish the methods above.
      2. You may use other packages for ARFF loading, graphing, generating a random number, etc, but remember we need to be able to run your code in the environment specified.
2. (10%) Create 2 dataset files, both with 8 instances using 2 real valued inputs (ranging between -1 and 1) with 4 instances from each class.
   1. You can use any format you want (ARFF, CSV, etc).
   2. One data set should be linearly separable and the other not.
   3. Include these two datasets in your report PDF.
3. (10%) Train on both sets (the entire sets) with your perceptron code.
   1. Use a couple different learning rates.
   2. Discuss the effect of learning rate, including how many epochs are completed before stopping. (For these cases, learning rate should have minimal effect, unlike with the Backpropagation lab.)
4. (10%) Graph the instances and decision line for the two cases above (with LR=.1)
   1. For all graphs always label the axes!
5. (20%) Use your perceptron code to learn [this version of the voting data set](https://learningsuite.byu.edu/plugins/Upload/fileDownload.php?fileId=5ecb4ed6-qi3m-Zfnp-kwfc-Yx6fcfe4da52).
   1. This particular task is an edited version of the [standard voting set](https://learningsuite.byu.edu/plugins/Upload/fileDownload.php?fileId=66213f91-qLjh-hFRB-4HML-5T3e648a1236), where we have replaced all the “don’t know” values with the most common value for the particular attribute.
   2. Create a table that reports the final training and test set accuracy and the # of epochs before stopping for each trial.
      1. Try it five times with different random 70/30 splits.
      2. Use your own code to randomize and make splits.
      3. Report the average of these values for the 5 trials in the table.
   3. By looking at the weights, explain what the model has learned and how the individual input features affect the result. Which specific features are most critical for the voting task, and which are least critical?
   4. Make a graph of the average misclassification rate vs epochs (0th – final epoch).
      1. Average the misclassification rate for the training set across your 5 trials.
         1. mean(number misclassified  /  number of total data points) vs epoch
      2. Note that for larger number epochs, only include those runs that trained for at least that length.
      3. To clarify what specific graphs should look like, find examples for this and future projects in "Example Graphs" in "Project Hints."
      4. As a rough sanity check, typical Perceptron accuracies for the voting data set are 90%-98%.
6. (10%) Use the perceptron algorithm from the [scikit-learn toolkit](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html) to learn the voting task above and also one other data set of your choice.
   1. Report and compare your results.
   2. Record your impressions of how it works. Try out some of the hyper-parameters that scikit-learn makes available for the perceptron and discuss your findings.
7. (Optional 5% extra credit) Use the perceptron rule to learn the [iris](https://learningsuite.byu.edu/plugins/Upload/fileDownload.php?fileId=dffa6cff-chF1-VkNA-k5za-PH29e9f038db) task or some other task with more than two possible output values.
   1. Note that the iris data set has 3 output classes, and a perceptron node only has two possible outputs.  You could implement either of the two most common ways to deal with this which we discussed in class. For testing, just execute the novel instance on each model and combine the overall results to see which output class wins.

Note:  In order to help you debug this and other projects we have included [some small examples and other hints](https://learningsuite.byu.edu/cid-9AKBdFBQ8oNg/student/pages/id-9kVm) with actual learned hypotheses so that you can compare the results of your code and help ensure that your code is working properly.  You may also discuss and compare results with classmates.