CS632
Assignment #1
Due: October 10th.

About: In this assignment, you will write your own classifier - according to API that mimics the basics of scikit-learn. There are two parts: coding and conceptual. First, you will implement a Nearest Neighbor classifier, and run an experiment on the Iris dataset. Next, you write a Spam classifier - starting from a zip file of emails the instructor will provide. You will divide the data into train/test, write code to extract features, run experiments, and consider the results.

Skills you will need (to have or learn):

- Understanding the basics of the scikit-learn API
- Understanding how a Nearest Neighbor classifier works
- Understanding basic ML terminology
- Python programming
- Working knowledge of GitHub

Grading: This assignment is worth 10% of your course grade (5% for each part, weighed equally).

Submission instructions: To submit your assignment, please upload your code and write-up to a private repository in your GitHub account. To create private repositories free-of-charge, you can sign up for the student developer pack using your .edu email address. Once your repo is created, please submit a link to this it via BlackBoard before the due date. Please structure your repository as follows, and remember to add the instructor as a collaborator (instructions follow), so they can access it.

1. Create a repository called “cs632”, and use the following directory structure.

cs632/

- hw1/ (a folder for assignment 1)
    - README.md (your write-up, in Markdown format)
    - part1.py (the main file for part one that runs your experiments)
    - part2.py (the main file for part two that runs your experiments)
    - … (any other files and data you use, as you prefer)

2. Share your repository with the instructor, by adding them as a collaborator.

- Visit github.com and navigate to your repository. Select settings → collaborators. Add the instructor as a collaborator (their GitHub username is “random-forests”).

Part 1: Write a Nearest Neighbor Classifier, mimicking the API of scikit-learn.

Part 1a) Coding question

1. Before you begin, please read and try the code from the very first part of this scikit-learn tutorial, in the section titled "Nearest neighbor and the curse of dimensionality" (fortuitously, it shows how to use the built in scikit-learn nearest neighbor classifier on the Iris dataset).

2. Although the example on this page prints out predictions on the test set, it is missing an accuracy score. In the code you submit, use this built in scikit-learn method to print out your accuracy on the test set.

3. Now, you will develop your own Nearest Neighbor classifier to replace the built-in method. *We will not replicate the entire scikit-learn API for classifiers.* Instead, your code should support just the following three methods.

3a. You should have a constructor, e.g., `clf = MyNearestNeighborClassifier(n_neighbors=3)`

Note: This should take `n_neighbors`, a parameter that indicates the number of neighbors to consider.

3b. Fit: `clf.fit(X_train, y_train)`

Note: A nearest neighbor classifier may not do anything inside the fit method, other than storing the training data and labels.

3c. Predict: `clf.predict(X_test, y_test)`

Note: this return the predicted class index for each example from the test set, just like the built-in method.

4. After you have implemented your classifier, calculate the accuracy on the test data using the built in method as before, and compare it to the built in model. Ballpark, you should have similar accuracy. You need not exactly replicate the results of the built-in classifier - but yours should return correct predictions according to your distance metric.

Submission for part 1a

- Code: create a file called "part1.py" inside your hw1 folder that contains a main function. Running this on the command line (e.g., `$python part1.py`) should train your model

on the Iris training set, and print out accuracy on the test set. If there are any special instructions to run your code, or difficulties you encounter - please put them in the README.md file.

Part 1b) Conceptual questions

Please include answers to the following questions (just in a paragraph, or less, each), and include them in your README.md file. Hint: topics like these will almost certainly appear on the midterm.

1. In a Nearest Neighbor classifier, is it important that all features be on the same scale? Think: what would happen if one feature ranges between 0-1, and another ranges between 0-1000? If it is important that they are on the same scale, how could you achieve this?

2. What is the difference between a numeric and categorical feature? How might you represent a categorical feature so your Nearest Neighbor classifier could work with it?

3. What is the importance of testing data?

4. What does “supervised” refer to in “supervised classification”?

5. If you were to include additional features for the Iris dataset, what would they be, and why?

Submission for part 1b

- Conceptual: please include answers to the above questions in the README.md file

Part 2: Spam classifier

Note: although you can work on this section of the assignment even if your classifier from part 1 is not working (by using the built in classifier in scikit-learn instead), you should begin by attempting part 1, and read the scikit-learn tutorial linked there. Also, do your best to answer the conceptual questions first.

Iris is a toy dataset, and these are useful to get started. In reality - of course - you’ll want to classify data for a new problem. In this part of the assignment, you’ll create your own structured dataset (meaning, in a CSV format similar to Iris), train your nearest neighbor classifier on it, and compute your accuracy.

Part 2a) Coding question

1. In this part of the assignment, you will design a spam classifier using a small dataset with 50 examples. To download the data, please clone this repo: https://github.com/random-forests/misc/tree/master/spam_data

    a. You can find labels for each example in labels.txt (0 indicates spam, 1 indicates not-spam).

    b. Notice the email files (e.g., 00000.txt) are unstructured. Meaning, they're just text that hasn't been processed in any way. Sometimes, they includes XML, sometimes they do not.

2. Start by dividing this data into train and test. There's not much of it (normally we'd use many more examples). I recommend a 50/50 split in this case.

3. Using the training set, write code to extract features. Your goal is create a structured representation of this data, in a format similar to Iris. What kind of features can you extract? Anything you like. Here are some suggestions to get you started.

    a. Does the document contain the symbol "$"?
    b. Does the document contain the word "money"?
    c. Does the document contain the word "drugs?"

4. Instead of thinking of predictive words manually, another way to construct a feature representation is to use a Bag of Words. I'd suggest creating a BOW, using the top 100 most common words in the training set.

5. Once you have your feature representation, "train" your nearest neighbor classifier from part 1 of this assignment on the training set, and compute your accuracy on the test set, and print it out.

Submission for part 2a

- Upload a file called "part2.py" to your github repo. Running this on the command line (e.g., `$python part2.py`) should your run pipeline. You may optionally take a data directory as a command line argument, or you may hardcode it and upload the dataset to your repo as well. When this file is run, it should divide the data set into train test, extract features, train the classifier, and report your accuracy.

Part 2b) Conceptual questions

1. What are the strengths and weaknesses of using a Bag of Words? (Tip: would this representation let you distinguish between two sentences the same words, but in a different order?)

2. Which of your features do you think is most predictive, least predictive, and why?

3. Did your classifier misclassify any examples? Why or why not?

Submission for part 2b

- Please include answers to the conceptual questions above in your README.md