Lab 5

Due: Friday 10/11/2024 @ 11:59pm EST

The purpose of labs is to give you some hands on experience programming the things we've talked about in lecture. This lab will focus on dimensionality reduction using random projections. You will turn a monte-carlo procedure into a las vegas one.

Task 0: Setup

Included with this file is a new file called `requirements.txt`. This file is rather special in Python: it is the convention used to communicate dependencies that your code depends on. For instance, if you open this file, you will see entries for `numpy` and `scikit-learn`: two pythn packages we will need in order to run the code in this lab. You can download and install these python packages through `pip`, which is python's package manager. While there are many ways of invoking `pip`, I like to do the following:

```
python3 -m pip install -r requirements.txt
```

(I let `python3` figure out which `pip` is attached to it rather than the more common usage: `pip install -r requirements.txt`)

Task 1: `def randomly_project` (30 points)

In the file `dim_reduction.py`, you will find a function stub called `randomly_project`. This function is where you will implement the monte-carlo random projection. According to the Johnson-Lindenstrauss lemma, if we create a random matrix A where every entry is populated from the unit gaussian (mean 0 and variance 1), then it is possible to use A to project our data into a lower dimensional space.

Let our dataset be $X \in \mathbb{R}^{n \times d}$ where there are n points initially in d-dimensional space. If we wish to project each point into a k dimensional space (ideally $k << d$), if we construct $A \in \mathbb{R}^{d \times k}$ such that every element of A is sampled i.i.d. from a unit gaussian, then we can construct linear function $f := \frac{1}{\sqrt{k}}A$. When applying f to X (i.e. matrix multiplication), we will get a matrix $\hat{X} \in \mathbb{X}^{n \times k}$, where the i^{th} row of $\hat{X}$ contains the i^{th} point in X projected into k dimensions.

Your `def randomly_project` function should perform this task:

1. construct $f := \frac{1}{\sqrt{k}}A$
2. project each point in X into k-dimensional space using f

Note: Your function should return both A as well as $\hat{X}$ (in that order).

Task 2: `check_if_distance_satisfied` (30 points)

Once we have a way of making a random projection, we want to know if the random projection preserves the geometry of the point cloud. In this function you should check that the johnson-lindenstrauss contraints are met:

$$\forall \vec{x}_i, \vec{x}_j \qquad (1-\epsilon)||\vec{x}_i - \vec{x}_j||_2^2 \leq ||f(\vec{x}_i) - f(\vec{x}_j)||_2^2 \leq (1+\epsilon)||\vec{x}_i - \vec{x}_j||_2^2$$

If a pair of points breaks a constraint, your function should return `False`, and `True` if all constraints are met.

Task 3: `def reduce_dims_randomly` (40 points)

This function is the las vegas version. `randomly_project` will do the projection, but the projection may not preserve the geometry of the point cloud (i.e. the output can be wrong). The way we will make this algorithm is as follows:

1. randomly project the data into k dimensions
2. check if the projection satisfies the constraints
3. repeat this process until a projection that satisfies the constraints is produced.

Your function should return three things (in the following order): the value of f that produced a valid projection, the valid projection itself, and the number of iterations it took to produce the valid projection.

Task 4: Submit Your Lab

To complete your lab, please **only turn in the `dim_reduction.py` file** on Gradescope. You shouldn't have to worry about zipping it up or anything, just drag and drop it in.