CS 475 Machine Learning: Homework 4
Clustering, EM, Dimensionality Reduction, Graphical Models
Introduction

Due: Wednesday April 15, 2020, 11:59pm

100 Points Total Version 1.0

Make sure to read from start to finish before beginning the assignment.

1 Partner Policy

You may work with a partner for this assignment. If you choose to do so, you will only make one submission for the two of you on Gradescope (make sure to include your partner when you submit). We *highly* recommend that you do every part of the assignment together instead of splitting it up. For the programming assignment, you can start a Zoom session with your partner and alternate who shares their screen to facilitate pair programming. For the analytical assignment, you can work on the same Overleaf document and think through the questions together.

2 Introduction

Assignments in this course can consist of two parts.

1. **Analytical:** These questions will ask you to consider questions related to the topics covered by the assignment. You will be able to answer these questions without relying on programming.
2. **Programming:** The goal of the programming assignments in this course is to learn about how machine learning algorithms work through hands-on experience. In each homework assignment you will first implement an algorithm and then run experiments with it. You may also be asked to experiment with these algorithms in a Python Notebook to learn more about how they work.

Assignments are worth various points. Typical assignments are worth 100 each, with point totals indicated in the assignment. Each assignment will contain a version number at the top. While we try to ensure every homework is perfect when we release it, errors do happen. When we correct these, we'll update the version number, post a new PDF and announce the change. Each homework starts at version 1.0.

3 Analytical (50 Points)

In addition to completing the analytical questions, your assignment for this homework is to learn LaTeX. All homework writeups must be PDFs compiled from LaTeX. Why learn LaTeX?

1. It is incredibly useful for writing mathematical expressions.
2. It makes references simple.
3. Many academic papers are written in LaTeX.

The list goes on. Additionally, it makes your assignments much easier to read than if you try to scan them in or complete them in Word.

We realize learning LaTeX can be daunting. Fear not. There are many tutorials on the Web to help you learn. We recommend using `pdflatex`. It's available for nearly every operating system. Additionally, we have provided you with the tex source for this PDF, which means you can start your writeup by erasing much of the content of this writeup and filling in your answers. You can even copy and paste the few mathematical expressions in this assignment for your convenience. As the semester progresses, you'll no doubt become more familiar with LaTeX, and even begin to appreciate using it.

Be sure to check out this cool LaTeX tool for finding symbols. It uses machine learning! `http://detexify.kirelabs.org/classify.html`

For each homework assignment we will provide you with a LaTeX template. You **must use the template**. The template contains detailed directions about how to use it.

At this point, please open the file `2020_Homework4_template.pdf` to read and `2020_Homework4_template.tex` respond to the analytical questions.

4 Programming (50 points)

4.1 Python Libraries

We will be using Python 3.7.3. We are *not* using Python 2, and will not accept assignments written for this version. We recommend using a recent release of Python 3.7.x, but any recent Python3 should be fine.

For each assignment, we will tell you which Python libraries you may use. We will do this by providing a `requirements.txt` file. We *strongly* recommend using a *virtual environment* to ensure compliance with the permitted libraries. By strongly, we mean that unless you have a very good reason not to, and you really know what you are doing, you should use a virtual environment. You can also use anaconda environments, which achieve the same goal. The point is that you should ensure you are only using libraries available in the basic Python library or the `requirements.txt` file.

4.2 Virtual Environments

Virtual environments are easy to set up and let you work within an isolated Python environment. In short, you can create a directory that corresponds to a specific Python version with specific packages, and once you activate that environment, you are shielded from the various Python / package versions that may already reside elsewhere on your system. Here is an example:

```
# Create a new virtual environment.
python3 -m venv python3-hw4
# Activate the virtual environment.
source python3-hw4/bin/activate
# Install packages as specified in requirements.txt.
pip install -r requirements.txt
# Optional: Deactivate the virtual environment, returning to your system's setup.
deactivate
```

When we run your code, we will use a virtual environment with *only* the libraries in `requirements.txt`. If you use a library not included in this file, your code will fail when we run it, leading to a large loss of points. By setting up a virtual environment, you can ensure that you do not mistakenly include other libraries, which will in turn help ensure that your code runs on our systems.

Make sure you are using the correct `requirements.txt` file from the current assignment. We may add new libraries to the file in subsequent assignments, or even remove a library (less likely). If you are using the wrong assignments `requirements.txt` file, it may not run when we grade it. For this reason, we suggest creating a new virtual environment for each assignment.

It may happen that you find yourself in need of a library not included in the `requirements.txt` for the assignment. You may request that a library be added by posting to Piazza. This may be useful when there is some helpful functionality in another library that we omitted. However, we are unlikely to include a library if it either solves a major part of the assignment, or includes functionality that isn't really necessary.

In this and future assignments we will allow you to use `numpy` and `scipy`.

"Can I use the command `import XXX`?" For every value of `XXX` the answer is: if you are using a virtual environment setup with the distributed `requirements.txt` file, then you can import and use anything in that environment.

Please verify you use the exact same command line flags we dictate. There are always students with a typo in their command line flags, which means their code fails when we run it.

4.3 How to Run the Provided Framework

The framework operates in two modes: train and test. Both stages are defined in `main.py`, which has the following arguments:

- `--mode` defines which mode to run it, either `train` or `test`
- `--train-data` Required in train mode! A file with labeled training data.
- `--test-data` Required in test mode! A file with labeled (dev) or unlabeled (test) data, to perform inference over.
- `--model-file` Required in both modes! In train mode, this will determine where to save your model after training. In test mode, this will tell the program where to load a pretrained model from, to perform inference.
- `--predictions-file` Required in test mode! Tells the program where to store a model's predictions over the test data.
- `--algorithm` Required in train mode! Which model to train. You need to implement these! This is only required during train-mode, since test-mode loads a pretrained model from a pickled file. The algorithms we'll use are `pegasos, kernel-pegasos`

In addition to these meta arguments, `main` also takes some hyperparameter arguments, namely:

- `--cluster-lambda` The value of λ in λ-means. By default, this value is 0.0.
- `--train-epochs` The number of iterations the model should train for on the entire dataset. By default, this value is 10.
- `--number-of-clusters` The number of clusters K to be used. By default, this value is 3.

Feel free to explore values for the last three hyperparameters to try to find hyperparameters that do the best for your algorithms. Note that when we run your code, we will use predetermined values for these, to ensure consistency amongst everyone's implementation.

4.3.1 Train Mode

The usage for train mode is

```
python3 main.py --mode train --algorithm algorithm_name --model-file model_file --train-data train_file
```

The `mode` option indicates which mode to run (train or test). The `algorithm` option indicates which training algorithm to use. Each assignment will specify the string argument for an algorithm. The `train-data` option indicates the data file to load. Finally, the `model-file` option specifies where to save the trained model.

4.3.2 Test Mode

The test mode is run in a similar manner:

```
python3 main.py --mode test --model-file model_file --test-data test_file --predictions-file predictions_file
```

The `model-file` is loaded and run on the `test-data`. Results are saved to the `predictions-file`.

4.3.3 Examples

As an example, the following trains a perceptron classifier on the speech training data:

```
python3 main.py --mode train --algorithm lambda_means --model-file bio.lambda_means.model \
                --train-data bio.train
```

To run the trained model on development data:

```
python3 main.py --mode test --model-file bio.lambda_means.model --test-data bio.dev \
                --predictions-file bio.dev.predictions
```

4.4 Data Formats

The data are provided in what is known as SVM-light format. Each line contains a single example:

```
0 1:-0.2970 2:0.2092 5:0.3348 9:0.3892 25:0.7532 78:0.7280
```

The first entry on the line is the label. The label can be an integer (0/1 for binary classification, 1-k for k multiclass classification) or a real-valued number (for regression.) The classification label of -1 indicates unlabeled. Subsequent entries on the line are features. The entry `25:0.7532` means that feature 25 has value 0.7532. Features are 1-indexed.

Model predictions are saved as one predicted label per line in the same order as the input data. The code that generates these predictions is provided in the library. The script `compute_accuracy.py` can be used to evaluate the accuracy of your predictions for classification:

```
python3 compute_accuracy.py test_file test_predictions_file
```

We provide this script since it is exactly how we will evaluate your output. Make sure that your algorithm is outputting labels as they appear in the input files. If you use a different internal representation of your labels, make sure the output matches what's in the data files. The above script will do this for you, as you'll get low accuracy if you write the wrong labels.

4.5 Existing Components

The foundations of the learning framework have been provided for you. You will need to complete this library by filling in code where you see a `TODO` comment. You are free to make changes to the code as needed provided you do not change the behavior of the command lines described above. We emphasize this point: **do not change the existing command line flags, existing filenames, or algorithm names.** We use these command lines to test your code. If you change their behavior, we cannot test your code.

The code we have provided is fairly compact, and you should spend some time to familiarize yourself with it. Here is a short summary to get you started:

- `data.py` – This contains the `load_data` function, which parses a given data file and returns features and labels. The features are stored as a sparse matrix of floats (and in particular as a `scipy.sparse.csr_matrix` of floats), which has `num_examples` rows and `num_features` columns. The labels are stored as a dense 1-D array of integers with `num_examples` elements.

- `main.py` – This is the main testbed to be run from the command line. It takes care of parsing command line arguments, entering train/test mode, saving models/predictions, etc. Once again, **do not change the names of existing command-line arguments**.

- `models.py` – This contains a `Model` class which you should extend. Models have (in the very least) a `fit` method, for fitting the model to data, and a `predict` method, which computes predictions from features. You are free to add other methods as necessary.

- `compute_accuracy.py` – This is a script which simply compares the true labels from a data file (e.g., `finance.dev`) to the predictions that were saved by running `classify.py` (e.g., `finance.dev.lambda_means.predictions`).

4.6 Evaluation

For clustering we cannot simply use accuracy against true labels to evaluate the output, since your prediction outputs are simply the indices of clusters (which could be arbitrary). Instead we will evaluate your output using an information-theoretic metric called *variation of information* (VI), which can be used to measure the dependence of the cluster indices with the true labels. The variation of information between two random variables Y and $\widehat{Y}$ is defined as: $H(Y|\widehat{Y}) + H(\widehat{Y}|Y)$. Lower is better. This will have a value of 0 if there is a

one-to-one mapping between the two labelings, which is the best you can do. We have provided a python script to compute this metric: `cluster_accuracy.py`. Additionally, we have provided a script that displays the number of unique clusters: `number_clusters.py`.

We will evaluate your implementation on the standard data sets you have been using (`speech`, `bio`, etc.) as well as the three-class `Iris` dataset, included here. We have removed the `nlp` data set from this assignment for computational efficiency.

4.7 Code Readability and Style

In general, you will not be graded for code style. However, your code should be readable, which means minimal comments and clear naming / organization. If your code works perfectly then you will get full credit. However, if it does not we will look at your code to determine how to allocate partial credit. If we cannot read your code or understand it, then it is very difficult to assign partial credit. Therefore, it is in your own interests to make sure that your code is reasonably readable and clear.

4.8 Code Structure

Your code must support the command line options and the example commands listed in the assignment. Aside from this, you are free to change the internal structure of the code, write new classes, change methods, add exception handling, etc. However, once again, do not change the names of the files or command-line arguments that have been provided. We suggest you remember the need for clarity in your code organization.

4.9 Grading Programming

The programming section of your assignment will be graded using an automated grading program. Your code will be run using the provided command line options, as well as other variations on these options (different parameters, data sets, etc.) The grader will consider the following aspects of your code.

1. **Exceptions:** Does your code run without crashing?
2. **Output:** Some assignments will ask you to write some data to the console. Make sure you follow the provided output instructions exactly.
3. **Accuracy:** If your code works correctly, then it should achieve a certain accuracy on each data set. While there are small difference that can arise, a correctly working implementation will get the right answer.
4. **Speed/Memory:** As far as grading is concerned, efficiency largely doesn't matter, except where lack of efficiency severely slows your code (so slow that we assume it is broken) or the lack of efficiency demonstrates a lack of understanding of the algorithm. For example, if your code runs in two minutes and everyone else runs in 2 seconds, you'll lose points. Alternatively, if you require 2 gigs of memory, and everyone else needs 10 MB, you'll lose points. In general, this happens not because you did not optimize your code, but when you've implemented something incorrectly.

4.10 Knowing Your Code Works

How do you know your code really works? That is a very difficult problem to solve. Here are a few tips:

1. Check results on `easy` and `hard`.
2. Use Piazza. While **you cannot share code**, you can share results. We encourage you to post your results on dev data for your different algorithms. A common result will quickly emerge that you can measure against.
3. Output intermediate steps. Looking at final predictions that are wrong tells you little. Instead, print output as you go and check it to make sure it looks right. This can also be helpful when sharing information on Piazza.
4. Debug. Find a Python debugger that you like and use it. This can be very helpful.

4.11 Debugging

The most common question we receive is "how do I debug my code?" The truth is that machine learning algorithms are very hard to debug because the behavior of the algorithm is unknown. In these assignments, you won't know ahead of time what accuracy is expected for your algorithm on a dataset. This is the reality of machine learning development, though in this class you have the advantage of your classmates, who may post the output of their code to the bulletin board. While debugging machine learning code is therefore harder, the same principles of debugging apply. Write tests for different parts of your code to make sure it works as expected. Test it out on the easy datasets to verify it works and, when it doesn't, debug those datasets carefully. Work out on paper the correct answer and make sure your code matches. Don't be afraid of writing your own data for specific algorithms as needed to test out different methods. This process is part of learning machine learning algorithms and a reality of developing machine learning software.

At this point, please open the file `homework4_coding.pdf` to read the programming assignment. You will modify the Python code provided.

5 Data

The first part of the semester will focus on supervised classification. We consider several real-world classification datasets taken from a range of applications. Each dataset is in the same format (described below) and contains a train, development and test file. You will train your algorithm on the train file and use the development set to test that your algorithm works. The test file contains unlabeled examples that we will use to test your algorithm. It is **a very good idea** to run on the test data just to make sure your code doesn't crash. You'd be surprised how often this happens.

5.1 Biology

Biological research produces large amounts of data to analyze. Applications of machine learning to biology include finding regions of DNA that encode for proteins, classification of gene expression data and inferring regulatory networks from mRNA and proteomic data.

Our biology task of characterizing gene splice junction sequences comes from molecular biology, a field interested in the relationships of DNA, RNA, and proteins. Splice junctions are points on a sequence at which "superfluous" RNA is removed before the process of protein creation in higher organisms. Exons are nucleotide sequences that are retained

after splicing while introns are spliced out. The goal of this prediction task is to recognize DNA sequences that contain boundaries between exons and introns. Sequences contain exon/intron (EI) boundaries, intron/exon (IE) boundaries, or do not contain splice examples.

For a binary task, you will classify sequences as either EI boundaries (label 1) or non-splice sequences (label 0). Each learning instance contains a 60 base pair sequence (ex. ACGT), with some ambiguous slots. Features encode which base pair occurs at each position of the sequence.

5.2 Finance

Finance is a data rich field that employs numerous statistical methods for modeling and prediction, including the modeling of financial systems and portfolios.[1]

Our financial task is to predict which Australian credit card applications should be accepted (label 1) or rejected (label 0). Each example represents a credit card application, where all values and attributes have been anonymized for confidentiality. Features are a mix of continuous and discrete attributes and discrete attributes have been binarized.

5.3 Vision

Computer vision processes and analyzes images and videos and it is one of the fundamental areas of robotics. Machine learning applications include identifying objects in images, segmenting video and understanding scenes in film.

Our vision task is image segmentation. In image segmentation, an image is divided into segments are labeled according to content. The images in our data have been divided into 3x3 regions. Each example is a region and features include the centroids of parts of the image, pixels in a region, contrast, intensity, color, saturation and hue. The goal is to identify the primary element in the image as either a brickface, sky, foliage, cement, window, path or grass. In the binary task, you will distinguish segments of foliage (label 0) from grass (label 1).

5.4 Iris

The iris dataset contains 50 samples of 3 different species of iris (150 samples total). Irises have 3 petals that point upwards and three characteristic sepal petals that point downwards. For each flower, we have measurements of the sepal length, sepal width, petal length, and petal width. You will distinguish between the species of iris using the petal and sepal measurements.

5.5 Synthetic Data: Easy

When developing algorithms it is often helpful to consider data with known properties. We typically create synthetic data for this purpose. To help test your algorithms, we are providing two synthetic datasets. These data are to help development.

The easy data is labeled using a trivial classification function. Any reasonable learning algorithm should achieve near flawless accuracy. Each example is a 10 dimensional instance drawn from a multi-variate Gaussian distribution with 0 mean and a diagonal identity

[1] For an overview of such applications, see the proceedings of the 2005 NIPS workshop on machine learning in finance. `http://www.icsi.berkeley.edu/~moody/MLFinance2005.htm`

covariance matrix. Each example is labeled according to the presence one of 6 features; the remaining features are noise.

5.6 Synthetic Data: Hard

Examples in this data are randomly labeled. Since there is no pattern, no learning algorithm should achieve accuracy significantly different from random guessing (50%). Data is generated in an identical manner as *Easy* except there are 94 noisy features.

6 What to Submit

In each assignment you may submit two different things.

1. **Submit your code (.py files) to gradescope.com as a zip file**. **Your code must be uploaded as code.zip with your code in the root directory**. By 'in the root directory,' we mean that the zip should contain `*.py` at the root (`./*.py`) and not in any sort of substructure (for example `hw4/*.py`). One simple way to achieve this is to zip using the command line, where you include files directly (e.g., `*.py`) rather than specifying a folder (e.g., `hw4`):

   ```
   zip code.zip *.py
   ```

   A common mistake is to use a program that automatically places your code in a subfolder. It is your job to make sure you have zipped your code correctly.

   We will run your code using the exact command lines described earlier, so make sure it works ahead of time, and make sure that it doesn't crash when you run it on the test data. A common mistake is to change the command line flags. If you do this, your code will not run.

   Remember to submit all of the source code, including what we have provided to you. We will include `requirements.txt` but nothing else.

2. **Submit your writeup to Gradescope**. **Your writeup must be compiled from LaTeX and uploaded as a PDF**. The writeup should contain all of the answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and to use the provided LaTeX template for your answers following the distributed template. You will submit this to the assignment called "Homework 2: Supervised Classifiers 2: Analytical".

You will need to create an account on gradescope.com and signup for this class. The course is `https://www.gradescope.com/courses/70713`. Use entry code `MK8J8N`. **You must either use the email account associated with your JHED, or specify your JHED as your student ID.** See this video for instructions on how to upload a homework assignment: `https://www.youtube.com/watch?v=KMPoby5g_nE`.

7 Questions?

Remember to submit questions about the assignment to the appropriate group on Piazza: `https://piazza.com/jhu/spring2020/601475`.