Spring 2023 Practical: Classifying Sounds

Due 11:59 PM on Friday, March 23rd, 2023

This assignment must be completed in groups of 2 to 3 students. You can seek partners via Ed, and course staff will release a team-finding spreadsheet. Section 3 of this document describes the classification problem, and Section 1 includes submission logistics and FAQs. In this same directory, you will find a file called `practical-template.tex`. Use it as a LaTeX template for your writeup, and be sure to read it for grading details.

Harvard College Honor Code: Members of the Harvard College community commit themselves to producing academic work of integrity – that is, work that adheres to the scholarly and intellectual standards of accurate attribution of sources, appropriate collection and use of data, and transparent acknowledgement of the contribution of others to their ideas, discoveries, interpretations, and conclusions. As such, **you may not share your code, or have it viewed by any other student outside your group.**

1 Logistics

1.1 Implementation Details

You can download the train and test datasets here, and clone a starter code repository here. You will be required to submit all of your implementation code to Gradescope.

This assignment can be completed on either your own computer, or on Google Colab. We recommend that you complete the assignment using Google Colab.

- If you choose to complete the assignment on your own computer, you can begin by working from the the `starter_code.ipynb` file, which contains starter code to load the dataset. You can watch a short video walkthrough of downloading the data and running the starter code notebook here.
- If you choose to complete the assignment on Google Colab, begin by copying this Colab notebook, which contains starter code to load the dataset. You can watch a short video walkthrough of the Colab notebook here.

1.2 Submission Details

The main deliverable of this practical is a 3-4 page typewritten document in PDF format to Gradescope. The document must follow the `practical-template.tex` file in this directory *and* follow all instructions outlined in Section 4. All relevant text—including your discussions of why you tried various things and why the results came out the way they did—must be included in the maximum of 4 pages. If you need more space than 4

pages for tables, plots, or figures, that is okay.

You should also submit your code as either a .py or .ipynb file on Gradescope. Make sure that the code is neatly organized to reflect which part is being answered. Before submitting, rerun the entire notebook and display all relevant results.

1.3 Grading

Our grading focus will be on your ability to clearly and concisely describe what you did, present the results, and most importantly *discuss* how and why different choices affected performance. Try to have a model that has at least 25% test accuracy at the end, although you will **not** be penalized if you were unable to achieve this.

Parts A, B1, B2 are each graded on a check, check-minus, and minus basis. A check is provided for successfully and thoughtfully completing the section. A check-minus is provided for completing parts of the section and providing little interpretation. Lastly, a minus is provided for providing little to no work. Part C is for optional exploration, for students who wish to explore the problem further.

See `practical-template.tex` for our desired submission format and more tips for what a full-credit submission looks like. All team members will receive the same practical grade.

1.4 Google Cloud

The Google Cloud Platform (GCP) offers a suite of cloud computing services. **You do not need to set up or use GCP to complete this practical.** Some students prefer to run code remotely on the Cloud instead of locally on their own computers for better job management. For more resources on getting started with GCP, see the Practical Addendum on Ed.

2 Impact Framing

Rainforest Connect (RFCx) is a non-profit organization that used acoustic remote-sensing technology to provide an automated, cost-effective, and inclusive remote-sensing system for monitoring species occurrence over time. They develop and leverage conservation technology in order to enable global partners, governments, corporations, scientists, and ecologists to understand the impacts on ecosystems and guide conservation management. RFCx uses acoustic monitoring to capture trends associated with the entire sound-producing component of the animal community, including fish, bird, amphibian, insect, mammal, and bat species. The acoustic monitoring sounds are then used to create

biodiversity analysis tools, threat detection tools, and a suite of hardware that can mobilize large-scale conservation.

RFCx uses Convolutional Neural Networks, an advanced form of the Neural Networks that we have seen in CS181, to enable real-time validation of species presence based on automated detection of sound spectra. For your CS181 Practical, you will be working with the UrbanSound8k dataset which consists of urban sounds taken around New York. Although this is just a classification project, consider the way that our data and model can be used to solve some real-world problems. Ask yourself the following questions:

1. Can we generate useful information by examining and classifying urban sounds?
2. Who would this information be useful to?
3. How can we use this information to generate ethical systems useful to society?

3 Problem Background

For this practical, you will classify sounds recorded using microphones around New York City into 10 classes. In making your predictions, you will primarily have at your disposal a series of amplitudes sampled for each sound. The classes of sounds under consideration in this practical are: `air_conditioner`, `car_horn`, `children_playing`, `dog_bark`, `drilling`, `engine_idling`, `gun_shot`, `jackhammer`, `siren`, and `street_music`.

The data source is the UrbanSound8k dataset, which contains labeled sound excerpts. These excerpts were created by splicing source audio recordings into shorter two-second non-overlapping clips. While multiple excerpts from the same source may be present within the train or test set, no source will have excerpts in both the train and test set. The dataset was created by NYU's Sounds of New York City (SONYC) initiative. For more information on how the dataset was created, you can read Sections 1 and 3 of this paper.

3.1 Data Files

There are 8 files of interest, which can be downloaded by clicking the download icon next to each file name on this Google Cloud bucket:

- `Xtrain_amp.npy, ytrain_amp.npy` – These files contain information about the 5779 sounds in the training set. The 44,100 columns of `Xtrain_amp.npy` are the sampled amplitudes (2 seconds at 22050 samples/second). Integer labels `ytrain_amp.npy` denote the class of each sound.
- `Xtest_amp.npy, ytest_ampy.npy` – These files contain information about the 1546 sounds in the test set. The 44,100 columns of `Xtest_amp.npy` are the sampled

amplitudes. Integer labels `ytest_amp.npy` denote the class of each sound. **Do NOT use these data for any training or parameter validation!**

- `Xtrain_mel.npy, ytrain_mel.npy` – These files contain the Mel spectrogram representation of the 5779 sounds in the training set. Each sound has a 2D spectrogram of shape (128×87). In short, the original amplitudes are partitioned into 87 time windows, and there are 128 audio-related features computed for each such window. Integer labels `ytrain_mel.npy` denote the class of each sound.

- `Xtest_mel.npy, ytest_mel.npy` – These files contain the Mel spectrogram representation of the 1546 sounds in the test set. Each sound has a 2D spectrogram of shape (128×87). Integer labels `ytest_mel.npy` denote the class of each sound. **Do NOT use these data for any training or parameter validation!**

3.2 Class Distribution

The distribution of sound classes in the training data is approximately as follows. It may be worthwhile to keep in mind that some classes are very infrequent.

0	`air_conditioner`	12.6%
1	`car_horn`	3.54%
2	`children_playing`	12.53%
3	`dog_bark`	9.42%
4	`drilling`	10.93%
5	`engine_idling`	12.98%
6	`gun_shot`	1.49%
7	`jackhammer`	11.85%
8	`siren`	12.03%
9	`street_music`	12.61%

4 Your Task and Deliverables

Below in Parts A-C we list three concrete explorations to complete in the context of this task. Through this process of guided exploration, you will be expected to think critically about how you execute and iterate your approach and describe your solution.

You are welcome to use whatever libraries, online resources, tools, and implementations that help you get the job done. Note, however, that you will be expected to *understand* everything you do, even if you do not implement the low-level code yourself. It is your responsibility to make it clear in your writeup that you did not simply download and run code.

4.1 Evaluation Metrics

In this practical, we would like you to primarily focus on optimizing for accuracy:

$$\text{Classification Accuracy} = \frac{\text{Number Correctly Classified Examples}}{\text{Total Number of Examples}}.$$

In Parts A - C, you will be asked to train several different classification models. **For each model you train, calculate the model's train and test set overall, and per-class classification accuracy** and include these results in your write-up.

Even better, try to run several seeds for each model in a given experiment and report mean and standard deviation of the evaluation metric. This helps you see which variability is due to chance vs. which is attributable to your experimental choices.

You may also explore more sophisticated classification metrics, such as the ROC curve and corresponding AUC score. This metric is useful when you have class imbalance in the dataset. In the binary classification setting, the ROC curve plots a the true positive rate versus its false positive rate for a model's predictions, given the classification threshold used. The AUC score calculates the area under this curve, and it is maximized at 1. Note that

$$\text{True Positive Rate} = \frac{\text{True Positive}}{\text{True Positive + False Negative}}.$$

$$\text{False Positive Rate} = \frac{\text{False Positive}}{\text{True Negative + False Positive}}.$$

In the multi-class setting, we generalize this using the concept of one-versus-rest classification. In this setting, we compute the ROC curve for each one of our classes. For each of the curves, the fixed class is regarded as the positive class and the other classes are taken together as the negative class. We then calculate different the multiclass ROC score using an averaging scheme, which you can read more about implementing here. Typically, we plot overlay all of these curves onto one plot, which can tell us how the model performs for each class.

4.2 Part A: Feature Engineering and Baseline Models

We have provided you with two sets of data files: `amp`, which contain raw amplitude data for each of the sound recordings, and `mel`, which contain derived Mel spectrogram features for each of the sound recordings. Here we first provide more context for each of these feature representations and then provide instructions for your deliverables.

In this practical, our data source is recorded audio clips. Audio data is sampled at discrete time steps. These samples are taken at a specified rate, named the sampling frequency

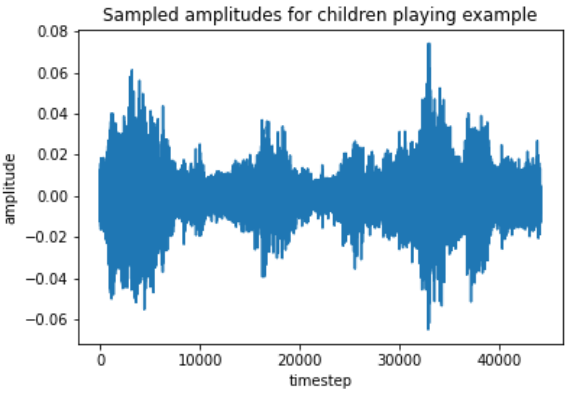

Figure 1: Plot of sound amplitudes for example file with label `children_playing`.

given in Hz. The raw data in the `Xtrain_amp.npy` and `Xtest_amp.npy` files contain the sampled signal amplitudes at each timestep. For example, the Figure 1 shows the signal amplitude versus timestep for an example file with label `children_playing`. A large body of machine learning and signal processing research has explored the predictive value of various audio feature engineering techniques that take as input raw amplitude data. Here we provide some basic intuition behind spectrogram features, which can be implemented using python package `librosa`[1].

A spectrogram is a 2D visual representation of the spectrum of frequencies of a signal as it varies with time[2]. There are several different kinds of spectrograms, most of which are generated by applying a Fourier transform to the sampled amplitudes[3]. Our spectrograms partition the ampltiude arrays into 87 sub-sequences and compute the presence of 128 different frequencies in each of these sub-sequences.

In the practical repository, you will find the file `generate_spectrograms.ipynb`. This notebook contains starter code that was used to transform raw sampled amplitdues to generate a Mel spectrogram, which is a type of short-time fourier transform with frequencies on the Mel (log) scale. To use Mel spectrogram features, you can simply just load the saved `Xtrain_mel.npy` and `Xtest_mel.npy` files.

Your task: Train the following two models.

1. Train a logistic regression model on the raw amplitude features (`Xtrain_amp`, `Xtest_amp`). This will be our first baseline model.

2. Train a logistic regression model on the Mel spectrogram features (`Xtrain_mel`, `Xtest_mel`). This will be our second baseline model.

[1] https://librosa.org/doc/latest/index.html
[2] https://en.wikipedia.org/wiki/Spectrogram
[3] See the following optional educational resources about spectrograms if you're interested: [1, 2].

Discuss which feature representation resulted in higher model performance, and why you hypothesize this feature representation performed better than the other.

4.3 Part B: More Modeling

Now, you will experiment with more expressive nonlinear model classes to maximize accuracy on the audio classification task. Examples of nonlinear models include random forests, KNN, and neural networks.

4.3.1 B1: First Step

First, we will be looking at simple models that are slightly more complicated than a linear model.

Your task: Train at least one nonlinear model on a feature representation of your choice. For model classes with hyperparameters, select a hyperparameter value you intuitively think is appropriate. Compare your results to the logistic regression models in Part A and discuss what your results imply about the task.

4.3.2 B2: More Complicated Models–Hyperparameter Tuning and Validation

In this section, you will explore hyperparameter tuning. Model hyperparameters such as network architecture or random forest maximum tree depth determine the expressivity of the model class. Training hyperparameters such as learning rate, weight decay, or regularization coefficients influence optimization and can encourage desirable properties (such as sparsity) in the final learned models.

Popular hyperparameter tuning techniques include random search, where you train a set of models with hyperparameters chosen uniformly at random from a set of possible values, [4] and grid search, where all possible parameter values are considered exhaustively [5].

Your task: Perform a hyperparameter search to maximize predictive accuracy for two model classes of your choice. You can choose which hyperparameters you search over (feel free to search over multiple simultaneously if you'd like!), but you must search over at least 5 possible values for at least 1 hyperparameter. Explore the changes in performance as you choose different hyperparameter values. In your writeup, discuss your validation strategy and your conclusions.

Optional: Weights & Biases (wandb) is a useful tool used in research to automatically run and evaluate different hyper-parameter configurations. W&B will run and collect key

[4] https://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf
[5] https://scikit-learn.org/stable/modules/grid_search.html

metrics for your sweep, and is easy to install and free to use. This is an **optional** item to include, but can be helpful in practice and future research projects. To start, you can reference the Weights & Biases documentation.

Note: Choose how to present your results of your hyperparameter search in a way that best reflects how to communicate your conclusions.

4.4 Final Write-up and Reflections

To wrap up all your exploratory research into applying different types of models and related training strategies to this use case, your final task is to document and summarize the steps you took in the ML research pipeline, and reflect on the choices that you made.

Your task: For each of the components below, include a paragraph (2-4 sentence) explanation and reflection of what you did for each step. Think through any trade-offs, domain-specific input, real-world considerations, and ethical decisions that you made along the way, in addition to any considerations made from a purely machine learning perspective.

Key Components:

1. **Data Pipeline:** What representations and data processing techniques did you employ in your exploration? Which one stands out as most relevant?
2. **Model Selection:** What was your choice of model, what trade-offs did you make? Did you observe any trends or advantages/drawbacks of any of the methods that you tried?
3. **Model Tuning:** How did you tune or optimize your model? Discuss your thoughts on your choice of problem framing and optimization technique. Did you include additional strategies (validation, cross-validation, ensembling) to further improve your models?
4. **Bias-Variance Trade-off:** Are any of your models over/under-fit? Did you take steps to find an optimal trade-off? What techniques did you employ to minimize over/under-fitting? What decisions did I ultimately make?
5. **Evaluation:** What metrics did you use to evaluate your models' performance? What drawbacks or advantages did those choices hold? Are there any metrics that you considered?
6. **Domain-specific Evaluation:** Are there any insights about the domain or dataset that you used to make key choices in your exploration? Do a gut check — do the choices of problem framing and optimization make sense for this specific problem?

Does your model (the resulting architecture, learned weights, etc.) make intuitive sense for making predictions on the data?

7. **Design Review:** Critique your designs and choices — is there anything that you would have done differently? Are there any foreseeable issues with my choices of dataset, models, optimization strategies, etc.?

8. **Execution & Implementation:** Are there are real-world considerations (deployment, training resources, data availability) that would be relevant to consider? Are there any ethical considerations to be had about collecting data and deploying these models? Should I, or could I, deploy this model?

4.5 Optional Exploration, Part C: Explore some more!

This section is not required to receive a full-credit grade of 18 points on the practical. See Section 1.3 for more details about practical grading.

Your task: Try any combination of the suggestions below, or come up with your own ideas to improve model training or expand your evaluation! In your write-up, discuss what you tried, what happened, and your conclusions from this exploration.

Some ideas:

- **Alternative feature representations:** Try out other popular audio feature representations like Mel-frequency cepstrum coefficients (MFCCs) [6] or others listed in librosa's documentation [7].
- **Use a CNN on the spectrogram data**: CNN architectures have been shown to achieve high classification accuracy when trained on audio spectrogram data [8]!
- **Explore other evaluation metrics**: You may also consider optimizing for or reporting other metrics e.g. precision or recall across various classes.
- **Address class imbalance**: Some classes are very infrequent in the training dataset. Popular techniques to address class imbalance are using a class-weighted loss function [9] or by upsampling infrequent classes during training [10].
- **Go totally Bayesian:** Worried that you're not accounting for uncertainty? You could take a fully Bayesian approach to classification and marginalize out your uncertainty in a generative or discriminative model.

[6] https://librosa.org/doc/main/generated/librosa.feature.mfcc.html
[7] https://librosa.org/doc/main/feature.html
[8] https://research.google/pubs/pub45611/
[9] https://www.kdnuggets.com/2018/12/handling-imbalanced-datasets-deep-learning.html
[10] https://towardsdatascience.com/handling-imbalanced-datasets-in-machine-learning-7a0e84220f28

4.6 Other FAQs

What language should I code in? As you will be submitting your code, you should code in Python.

Can I use {scikit-learn | pylearn | torch | shogun | other ML library}? You can use these tools, but not blindly. You are expected to show a deep understanding of the methods we study in the course, and your writeup will be where you demonstrate this.

What do I submit? You will submit both your write-up on Gradescope and all of your practical code to a supplemental assignment on Gradescope.

Can I have an extension? Yes, your writeup can be turned in late according to standard homework late day policy. You can use at most two late days on the practical.