PMP596 Homework 1 (100%)

Due 10/14, Thursday, before class (6:00 PM)

Submission Instructions

You will need to submit the following materials to canvas:

1. Paper reading report (3 pages max) in PDF format. Remember to put your names in your report. Please see other detailed instructions for the report in the last page.
2. A folder containing all `ipynb` files with your results for the other problems. Please remember to keep all results and logs in the submitted `ipynb` files to get full credit.

All submitted files should be put into one single zip file named as `HW#_xxx.zip`, e.g. HW1_George_Clooney.zip**,** including `HW1_*.ipynb` files with answers (both results and discussions), and the paper reading report.

Problem 1: Data Representation (15%)

Introduction

After running the code in the tutorials, we will go through a definition called Principal Component Analysis (PCA) for dimensionality reduction for high dimensional data. We do dimensionality reduction to convert the high d-dimensional dataset into k-dimensional data where $k < d$. Data variance on one axis may be very large but relatively smaller on another axis. Higher variance usually means greater information in this direction. Therefore, we can skip the dimensions having less variance because having less information.

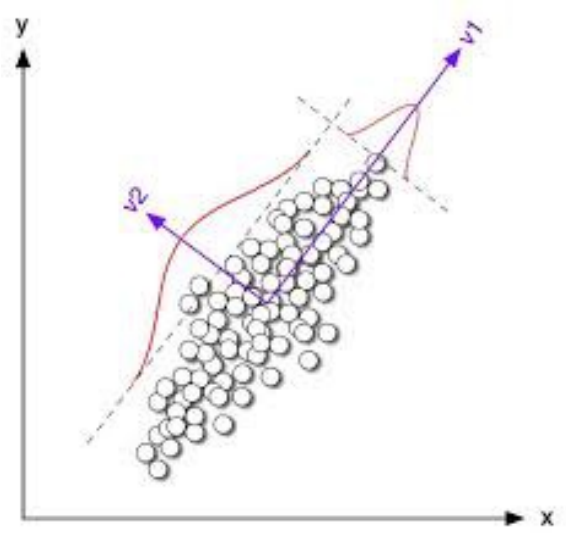


Fig. 1: Variances of 2D data.

Fig. 1 is an example of a set of 2D data. The direction of v1 is maximum while v2 is minimum, so that v1 has more information about the dataset. Thus, the 2D data with (x,y) variables can be converted to 1D variables in the direction of v1.

Projections (Orthogonal)

Input: $x_1, x_2, \ldots, x_n \in R^d$, target dim k.

Output: a k-dim subspace by orthonormal basis $q_1, q, \ldots, q_k \in R^d$.

Orthogonal projection:

$$\underbrace{\left(\sum_{i=1}^{k} \boldsymbol{q}_i \boldsymbol{q}_i^\top\right)}_{\boldsymbol{\Pi}} \boldsymbol{x} \;=\; \sum_{i=1}^{k} \langle \boldsymbol{q}_i, \boldsymbol{x} \rangle \boldsymbol{q}_i \;\in\; \mathbb{R}^d .$$

It can be also represented in terms of coefficients w.r.t. the orthonormal basis $q_1, q, \ldots, q_k \in R^d$:

$$\boldsymbol{\phi}(\boldsymbol{x}) \;:=\; \begin{bmatrix} \langle \boldsymbol{q}_1, \boldsymbol{x} \rangle \\ \langle \boldsymbol{q}_2, \boldsymbol{x} \rangle \\ \vdots \\ \langle \boldsymbol{q}_k, \boldsymbol{x} \rangle \end{bmatrix} \;\in\; \mathbb{R}^k .$$

Minimize residual squared error

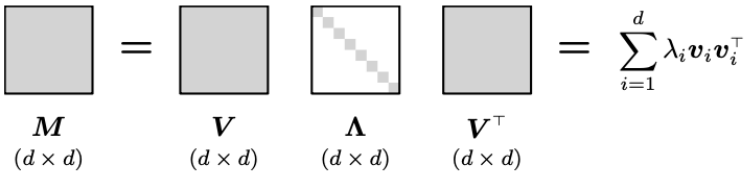
$$\underset{\substack{\boldsymbol{Q} \in \mathbb{R}^{d \times k}: \\ \boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}}}{\arg\min} \; \frac{1}{n} \sum_{i=1}^{n} \left\| \boldsymbol{x}_i - \boldsymbol{Q}\boldsymbol{Q}^\top \boldsymbol{x}_i \right\|_2^2 \quad \equiv \quad \underset{\substack{\boldsymbol{Q} \in \mathbb{R}^{d \times k}: \\ \boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}}}{\arg\max} \; \sum_{i=1}^{k} \boldsymbol{q}_i^\top \left(\frac{1}{n} \boldsymbol{A}^\top \boldsymbol{A} \right) \boldsymbol{q}_i .$$

(where $\boldsymbol{x}_i^\top$ is i-th row of $\boldsymbol{A} \in \mathbb{R}^{n \times d}$).

Solution: k eigenvectors of $A^\top A$ corresponding to k largest eigenvalues.

Eigen Decompositions

Every symmetric matrix $\boldsymbol{M} \in \mathbb{R}^{d \times d}$ guaranteed to have eigendecomposition with real eigenvalues:



real **eigenvalues**: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \quad (\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d))$;
corresponding orthonormal **eigenvectors**: $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d \quad (\boldsymbol{V} = [\boldsymbol{v}_1 | \boldsymbol{v}_2 | \cdots | \boldsymbol{v}_d])$.

PCA step-by-step

Step 1: Standardize the dataset.
Step 2: Calculate the covariance matrix for the features in the dataset.
Step 3: Calculate the eigenvalues and eigenvectors for the covariance matrix.
Step 4: Sort eigenvalues and their corresponding eigenvectors.
Step 5: Pick k eigenvalues and form a matrix of eigenvectors.
Step 6: Transform the original matrix.

Find data representation for MNIST dataset

Handwritten classification is a basic task for early stage image processing. In this problem, we will work with MNIST dataset, which contains 60,000 training and 10,000 testing images of handwritten numbers 0-9.

Fig. 1: Some sample images in MNIST dataset.

(a) Prepare MNIST dataset (3%)

Download MNIST dataset `mnist.mat` from Canvas and follow the instruction in `HW1_P2P3.ipynb` to load and visualize the dataset.

In this part, you will also practice how to load files from your Google Drive to the Google Colab Notebook. To achieve this, you will need to:

- Upload file(s) to your own Google Drive.
- Mount your Google Drive to the corresponding Colab Notebook.
- Find the path of your uploaded file(s) and access them.
- Split the validation set from the training set (train: 50000, valid: 10000, test: 10000).

The detailed instructions for this part is shown in the `HW1_P2P3.ipynb` file.

(b) PCA on MNIST (12%)

Implement PCA on MNIST dataset to reduce the dimension of the digit images.

1) Each data in the MNIST dataset is a 784-dimensional vector (flatten from 28x28). Please use PCA to reduce the dimension to a smaller value k.
2) How to **select** a good reduced dimension k? Keep 80% information after the reduction.
3) Transform all the data into the reduced dimension (train: 60000 x k, test: 10000 x k).

Note that you will use this reduced data for digit classification in the following problem.

Problem 2: Classification on Digit Images (30%)

In this problem, we will **use the data representation we obtained in Problem 2** to classify the digits on the MNIST dataset. You will be asked to implement k-Nearest Neighbor (k-NN) and k-Means Clustering algorithms to classify the digit images in MNIST dataset.

(a) k-NN Classifier (15%)

Follow the steps to build a k-NN Classifier for MNIST dataset. Here, you should use **both raw data** (784-D) **and representative data** (from P2b) for training and inference.

1) Implement a **1-NN** classifier using the `scikit-learn` package: `sklearn.neighbors.KNeighborsClassifier` with Euclidean distance on the selected training set, i.e., randomly select $n \in \{1000, 2000, 5000, 10000, 30000, 60000\}$ entries from the training set.
2) Evaluate the classification accuracies on the validation set with different training set entries. A plot of the accuracy rate (y-axis) as a function of n (x-axis) is called a learning curve. We estimate this curve by using the validation accuracy in place of the true accuracy. **Draw the learning curve** of your results.
3) Adjust the **hyperparameter k** and report the classification accuracies. Which k gives the best performance? Give some explanation and analysis of your conclusion.
4) Run your classifier on the testing set and report accuracy.

(b) k-Means Clustering (20%)

Follow the steps to build a k-Means clustering algorithm for MNIST dataset. Here, you can just use the **representative data** (from P2b).

1) Implement ***k*-Means** clustering algorithm using the `scikit-learn` package: `sklearn.cluster.KMeans` on the **training** set. Use k = 10 as the cluster number.
2) Since k-Means is an unsupervised method, the cluster IDs are not the actual label IDs (i.e., real digits). For each cluster, please show the **distribution of the real digits** (using histogram or table), and use the real digit with maximal frequency (most common) as the prediction result (similar to the idea of **majority vote**). Report the accuracy.
3) The number of classes is 10 but why is the performance of k = 10 not good?
4) Now, you can use different k and add other splitting and grouping strategies to improve your clustering algorithm. But the final results should still be 10 clusters. Note that you **cannot touch** any ground truth labels since it is an unsupervised method. Can you improve the accuracy in question 2)?
5) Infer on the **testing** set using the clustering model(s) created above. Evaluate the distribution mentioned in 2) and report accuracy.

(c) Discussions (5%)

From the results in (b) and (c), discuss:

1) List the differences between supervised and unsupervised learning algorithms?
2) How to choose hyperparameters for a machine learning algorithm?

Problem 3: Sound Clustering (30%)

In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. MFCCs are commonly used as features in speech recognition systems, such as the systems which can automatically recognize numbers spoken into a telephone. MFCCs are commonly derived as follows:

a. Take the Fourier transform of (a windowed excerpt of) a signal.
b. Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows.
c. Take the logs of the powers at each of the mel frequencies.
d. Take the discrete cosine transform of the list of mel log powers, as if it were a signal.
e. The MFCCs are the amplitudes of the resulting spectrum.

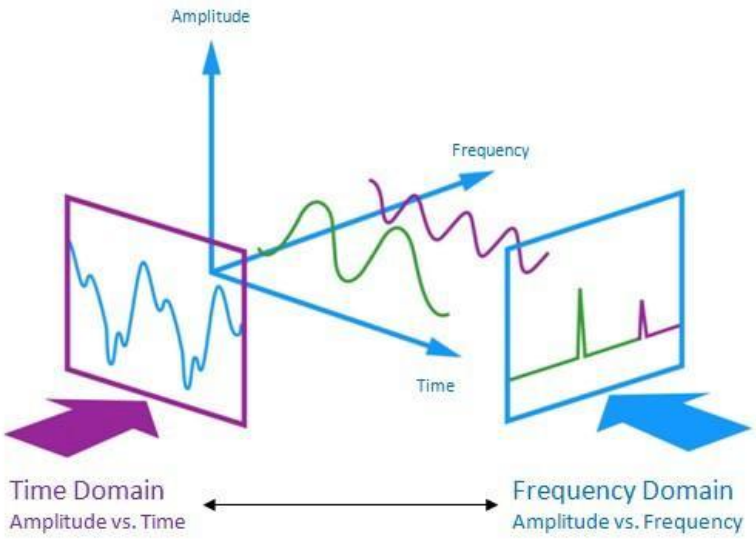
The above sounds scary and tedious? No worries about the theories, Python has integrated huge amounts of libraries to solve all kinds of problems. Let's first walk through a simple process using Python to do the feature extraction for sound (music, speech, etc.) and then classify the data into different clusters.

1. Feature Extraction (10%)

Extraction of features is a very important part in analyzing and finding relations between different things. The data provided of audio cannot be understood by the models directly to convert them into an understandable format feature extraction is used. It is a process that explains most of the data but in an understandable way. Feature extraction is required for classification, prediction and recommendation algorithms.

In P1, we will first extract features of animal sound files that will help us to classify the sound into different clusters. Let's get familiar with the audio signal first.

The audio signal is a 3-dimensional signal in which the three axes represent the time, amplitude and frequency, shown in the Figure. We will be using [librosa] (https://librosa.github.io/librosa/) for analyzing and extracting features of an audio signal. For playing audio, we will use [pyAudio] (https://people.csail.mit.edu/hubert/pyaudio/docs/) so that we can play music directly on Colab.



Audio Signal

Loading an audio

Download three audio files (**Bluejay.mp3**, **Dove.mp3** and **Ducks.wav**) provided on Canvas webpage. Upload your files by clicking **Files -> Upload** to your session storage.

```
# Loading an audio
# let's take bluejay.mp3 as an example
import librosa

audio_path = "Bluejay.mp3"
x , sr = librosa.load(audio_path)
print(type(x), type(sr))
```

.load loads an audio file and decodes it into a 1-dimensional array which is a time series *x* , and *sr* is a sampling rate of *x* . Default *sr* is 22kHz. We can override the *sr* by

```
librosa.load(audio_path, sr=44100)
```

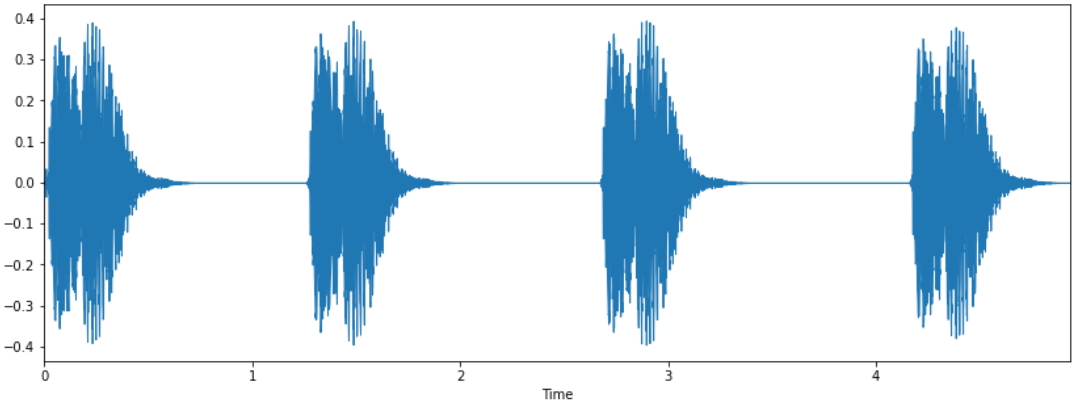
We can also disable sampling by:

```
librosa.load(audio_path, sr=None)
```

Playing an audio

librosa.display is used to display the audio files in different formats such as wave plot, spectrogram, or colormap. Waveplots let us know the loudness of the audio at a given

time. Amplitude and frequency are important parameters of the sound and are unique for each audio. **librosa.display.waveplot** is used to plot waveforms of amplitude vs. time where the first axis is an amplitude and the second axis is time.

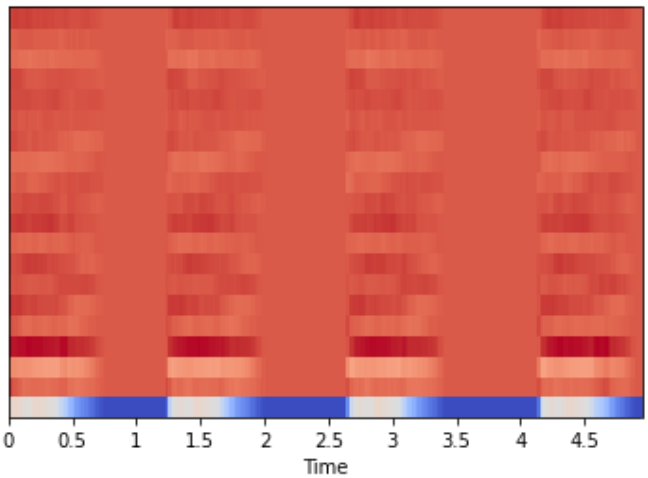


wavelet of blue jay audio file

MFCC – Mel-Frequency Cepstral Coefficients

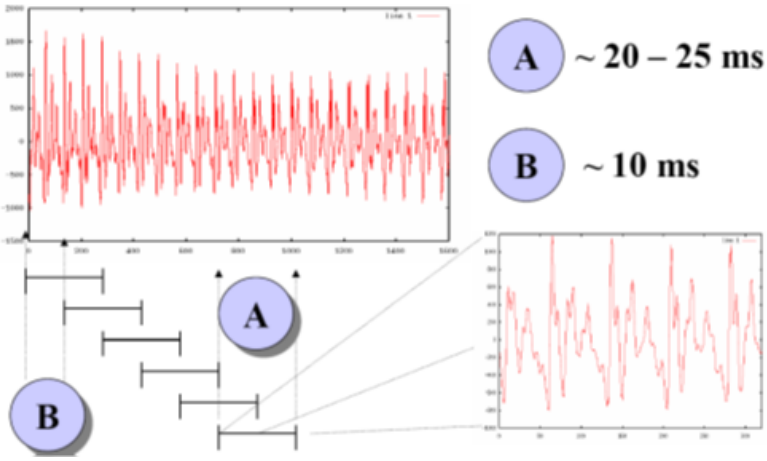
This feature is one of the most important methods to extract a feature of an audio signal and is used majorly whenever working on audio signals. The MFCCs of a signal are a small set of features (usually about 10–20) which concisely describe the overall shape of a spectral envelope.

```
# MFCC – Mel-Frequency Cepstral Coefficients
mfccs = librosa.feature.mfcc(x, sr=sr)
print(mfccs.shape)
# Displaying  the MFCCs:
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
```



MFCC

Questions:



- Q1 (5%): Do the framing of Bird sounds using A=20ms windows, B=10ms, and A-B=10ms of overlapping. 4 seconds of Bird sounds will generate 399 frames.
- Q2 (5%): Generate 13 MFCC coefficients for every frame. Every 4 sec of Bird sound will have a 399x13 MFCC coefficient matrix as a result and plot the 399x13 MFCC coefficients for all three Bird sounds.

2. Training GMM using MFCC features (10%)

Gaussian Mixture Model (GMM) helps to cluster the features. **sklearn.mixture** is a package which enables one to learn Gaussian Mixture Models (diagonal, spherical, tied and full covariance matrices supported), sample them, and estimate them from data. Facilities to help determine the appropriate number of components are also provided. For more details, please refer to [scikit-learn GMM](https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html).

Question:

- Q4 (10%): Find the GMM parameters of the features found in 1. Since there are three Bird sounds, there will be three clusters. You can use the existing GMM function provided from Python.
  (1) Concatenate the MFCCs from all three sound files and become the feature matrix, leaving the last 50 samples for X_test. In the rest sets, define your training/validation set X_train, X_val using train_test_split() and create the labels for each class y_train, y_val.
  (2) Instantiate a Scikit-Learn GMM model by using:

model = sklearn.mixture.GaussianMixture(n_components, covariance_type, reg_covar, verbose, etc.)

(3) Train a model: model.fit(X_train).

(4) Predict the model: y_val_predict= model.predict(X_val)

(5) Calculate the classification accuracy using accuracy_score(y_val_predict, y_val)

3. Training SVM for Bird sound classification (10%)

Support Vector Machine (SVM) helps with classification of the data. For usage and more details, please refer to [scikit-learn SVM](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html).

Questions:

- Q5 (10%): Train the SVM model using the features found in 1. You can use the existing SVM function provided by Python.
  (1) Concatenate the MFCCs from all three sound files and become the feature matrix, leaving the last 50 samples for X_test. In the rest sets, define your training/validation set X_train, X_val using train_test_split() and create the labels for each class y_train, y_val.
  (2) Instantiate a Scikit-Learn SVM model by using:
  model = sklearn.svm.SVC()
  (3) Train a model using model.fit(X_train).
  (4) Predict the model: y_val_predict = model.predict(X_val)
  (5) Calculate the classification accuracy using accuracy_score(y_val_predict, y_val)

4. Training MLP for Bird sound classification (10%)

- Q6 (10%): Train a MLP model following the Pytorch_NN.ipynb tutorial. Calculate and report the classification accuracy.

Problem 4: Paper Reading (25%)

Backpropagation Applied to Handwritten Zip Code Recognition
Y. LeCun, et al.

Abstract: The ability of learning networks to generalize can be greatly enhanced by providing constraints from the task domain. This paper demonstrates how such constraints can be integrated into a backpropagation network through the architecture of the network. This approach has been successfully applied to the recognition of handwritten zip code digits provided by the U.S. Postal Service. A single network learns the entire recognition operation, going from the normalized image of the character to the final classification.

Please download the paper from Canvas.

Requirements: On your own word (we will check the plagiarization) **3 pages max, font size 12, single linspace**

(1) Motivation – Why do the authors want to work on this problem
(2) Contributions – What are the accomplishments they achieved in this paper (others did not achieve)?
(3) Formulations – How do they solve the problems as mentioned/discussed in the introduction or related literature?
(4) Justification – Do the experiments/simulations support their claimed accomplishments?
(5) Your Own Thoughts – What are you most impressed with in this paper?