

# CS 505 Homework 04: Classification

Due Friday 10/27 at midnight (1 minute after 11:59 pm) in Gradescope (with a grace period of 6 hours)

You may submit the homework up to 24 hours late (with the same grace period) for a penalty of 10%.

All homeworks will be scored with a maximum of 100 points; point values are given for individual problems, and if parts of problems do not have point values given, they will be counted equally toward the total for that problem.

Note: I strongly recommend you work in **Google Colab** (the free version) to complete homeworks in this class; in addition to (probably) being faster than your laptop, all the necessary libraries will already be available to you, and you don't have to hassle with `conda`, `pip`, etc. and resolving problems when the install doesn't work. But it is up to you! You should go through the necessary tutorials listed on the web site concerning Colab and storing files on a Google Drive. And of course, Dr. Google is always ready to help you resolve your problems.

I will post a "walk-through" video ASAP on my [Youtube Channel](#).

## Submission Instructions

You must complete the homework by editing **this notebook** and submitting the following two files in Gradescope by the due date and time:

- A file `HW04.ipynb` (be sure to select `Kernel -> Restart and Run All` before you submit, to make sure everything works); and
- A file `HW04.pdf` created from the previous.

For best results obtaining a clean PDF file on the Mac, select `File -> Print Review` from the Jupyter window, then choose `File-> Print` in your browser and then `Save as PDF`. Something similar should be possible on a Windows machine -- just make sure it is readable and no cell contents have been cut off. Make it easy to grade!

The date and time of your submission is the last file you submitted, so if your IPYNB file is submitted on time, but your PDF is late, then your submission is late.

## Collaborators (5 pts)

Describe briefly but precisely

1. Any persons you discussed this homework with and the nature of the discussion;

2. Any online resources you consulted and what information you got from those resources; and
3. Any AI agents (such as chatGPT or CoPilot) or other applications you used to complete the homework, and the nature of the help you received.

A few brief sentences is all that I am looking for here.

<Your answer here>

```
In [47]: import math
import numpy as np
from numpy.random import shuffle, seed, choice
from tqdm import tqdm
from collections import defaultdict, Counter
import pandas as pd
import re
import matplotlib.pyplot as plt

import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F
from torch.utils.data import random_split, Dataset, DataLoader
from torchvision import datasets, transforms
from torch import nn, optim

from torchvision.datasets import MNIST
import torchvision.transforms as T

from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
```



## Problem One: Exploring Shakespeare's Plays with PCA (45 pts)

In this problem, we will use Principal Components Analysis to look at Shakespeare's plays, as we discussed with a very different play/movie in lecture. Along the way, we shall use the tokenizer and the TF-IDF vectorizer from `sklearn`, a common machine learning library.

Note: There is a library for text analysis in Pytorch called `Torchtext`, however, in my view this will be less well-developed and less well-supported than the rest of Pytorch, so we shall use `sklearn` for this problem.

## Part A: Reading and exploring the data (5 pts)

The cells below read in three files and convert them to `numpy` arrays (I prefer to work with the arrays rather than with `pandas` functions, but it is your choice).

- The file `shakespeare_plays.csv` contains lines from William Shakespeare's plays. The second column of the file contains the name of the play, the third the name of the player (or the indication `<Stage Direction>`, and the fourth the line spoken:

0	1	2	3		
1	Henry IV Part 1	<Stage Direction>	ACT I		
2	Henry IV Part 1	<Stage Direction>	SCENE I. London. The palace.		
3	Henry IV Part 1	<Stage Direction>	Enter KING HENRY, LORD JOHN OF LANCASTER,		
4	Henry IV Part 1	KING HENRY IV	So shaken as we are, so wan with care,		
5	Henry IV Part 1	KING HENRY IV	Find we a time for frightened peace to pant,		

- The file `play_attributes.csv` stores the genres and chronology of Shakespeare's plays; the first column is the name of the play, the second the genre, and the third its order in a chronological listing of when it was first performed. The plays are in the same (arbitrary) order as in the first file.

Name	Genre	Chronology
Henry IV Part History		15
Henry VI Part History		1
Henry VI Part History		2
Henry VI Part History		3
All's Well Th Comedy		25
As You Like It Comedy		20

- The file `player_genders.csv` stores the name of a major character (defined somewhat arbitrarily as one whose total lines contain more than 1400 characters) in the first column and their gender in the second.

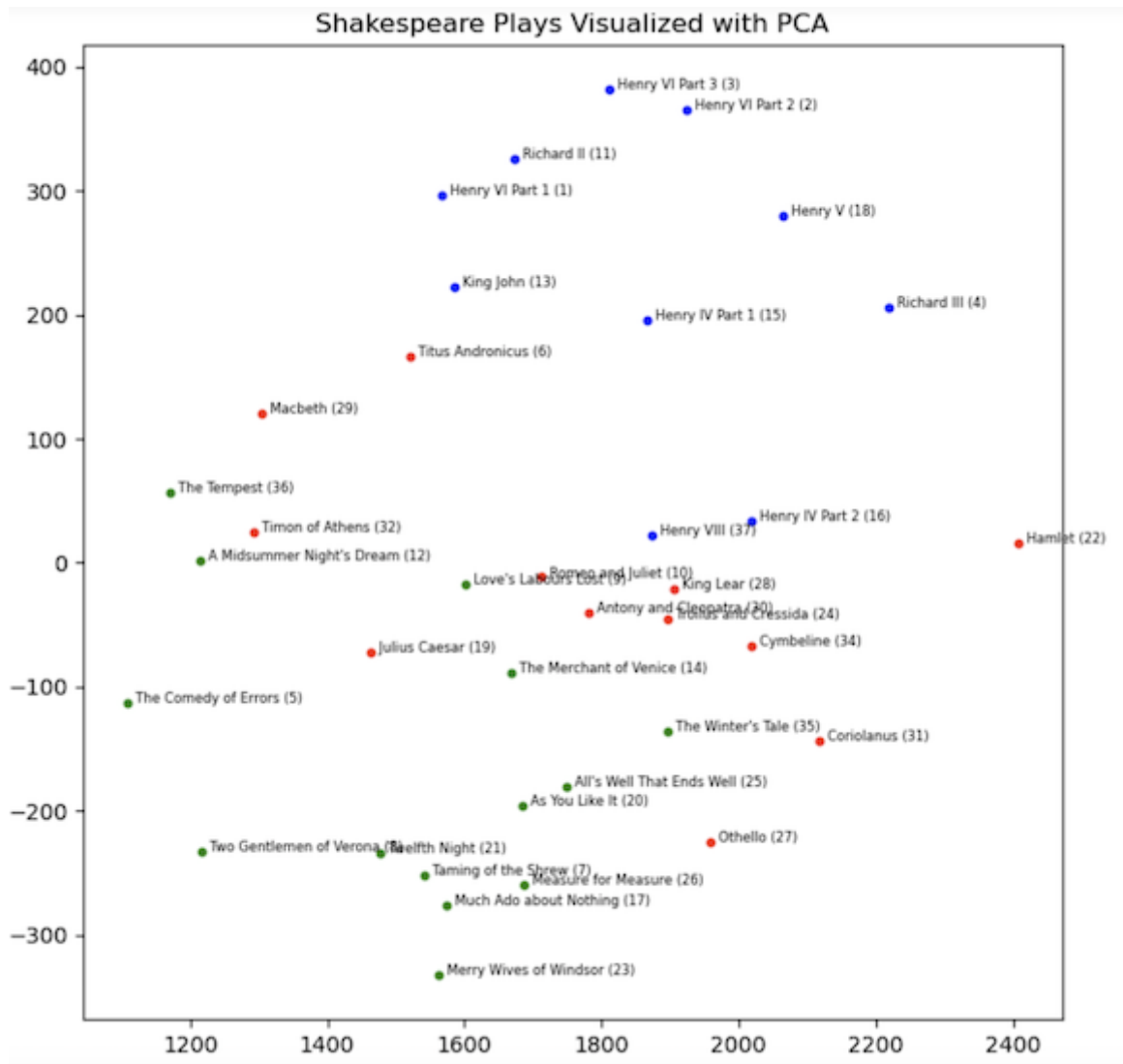
Name	Gender
AARON	male
ACHILLES	male
ADAM	male
ADRIANA	female
ADRIANO DE	male
AEGEON	male
AEMELIA	female

**To Do:** For each of the arrays, print out the the `shape` and the first line.

```
In [48]: plays_array = pd.read_csv('https://www.cs.bu.edu/fac/snyder/cs505/shakespeare_r  
player_genders_array = pd.read_csv('https://www.cs.bu.edu/fac/snyder/cs505/play  
play_attributes_array = pd.read_csv('https://www.cs.bu.edu/fac/snyder/cs505/pla
```

## Part B: Visualizing the Plays (8 pts)

1. Create an array containing 36 strings, each being the concatenation of all lines spoken. Be sure to NOT include stage directions! You may wish to create an appropriate dictionary as an intermediate step.
2. Create a document-term matrix where each row represents a play and each column represents a term used in that play. Each entry in this matrix represents the number of times a particular word (defined by the column) occurs in a particular play (defined by the row). Use `CountVectorizer` in `sklearn` to create the matrix. Keep the rows in the same order as in the original files in order to associate play names with terms correctly.
3. From this matrix, use `TruncatedSVD` in `sklearn` to create a 2-dimensional representation of each play. Try to make it as similar as possible to the illustration below, including (i) appropriate title, (ii) names of each play, followed by its chronological order, and (iii) different colors for each genre. Use a `figsize` of `(8,8)` and a `fontsize` of `6` to provide the best visibility. You can follow the tutorial [here](#) to create the visualization (look at the "PCA" part).
4. Now do the same thing all over again, but with TF-IDF counts (using `TFIDFVectorizer` in `sklearn`).
1. Answer the following in a few sentences: What plays are similar to each other? Do they match the grouping of Shakespeare's plays into comedies, histories, and tragedies here? Which plays are outliers (separated from the others in the same genre)? Did one of TF or TF-IDF provided the best insights?



## Part C: Visualizing the Players (8 pts)

Now you must repeat this same kind of visualization, but instead of visualizing plays, you must visualize players. The process will be essentially the same, starting with an array of strings representing the lines spoken by each player. Use one of TF or TF-IDF, and use different colors for the genders.

Use a figsize of (8,8) and a fontsize of 4 to make this a bit more visible.

Again, comment on what you observe (it will not be as satisfying as the previous part).

## Part D: DIY Word Embeddings (8 pts)

In this part you will create a word-word matrix where each row (and each column) represents a word in the vocabulary. Each entry in this matrix represents the number of times a particular word (defined by the row) co-occurs with another word (defined by the column) in a sentence (i.e., line in `plays`). Using the row word vectors, create a

document-term matrix which represents a play as the average of all the word vectors in the play.

Display the plays using `TruncatedSVD` as you did previously.

Again, comment on what you observe: how different is this from the first visualization?

#### Notes:

1. Remove punctuation marks . , ; : ? ! but leave single quotes.
2. One way to proceed is to create a nested dictionary mapping each word to a dictionary of the frequency of words that occur in the same line, then from this to create the sparse matrix which is used to create the average document-term matrix which is input to `TruncatedSVD`.
3. If you have trouble with the amount of memory necessary, you may wish to eliminate "stop words" and then isolate some number (say, 5000) of the remaining most common words, and build your visualization on that instead of the complete vocabulary.

## Part E: Visualizing the Plays using Word2Vec Word Embeddings (8 pts)

Now we will do the play visualization using word embeddings created by Gensim's `Word2Vec`, which can create word embeddings just as you did in the previous part, but using better algorithms.

You can read about how to use `Word2Vec` and get template code here:

<https://radimrehurek.com/gensim/models/word2vec.html>

I strongly recommend you follow the directions for creating the model, then using `KeyedVectors` to avoid recomputing the model each time.

Experiment with the `window` (say 5) and the `min_count` (try in the range 1 - 5) parameters to get the best results.

Display the plays using `PCA` instead of `TruncatedSVD`.

Again, comment on what you observe: how different is this from the other visualizations?

## Part F: Visualizing the Players using Word2Vec Word Embeddings (8 pts)

Now you must repeat Part C, but using these `Word2Vec` embeddings.

Use a figsize of (8,8) and a fontsize of 4 to make this a bit more visible.

Again, comment on what you observe. How is this different from what you saw in Part C?

## Problem Two: Classifying Text with a Feed-Forward Neural Network (50 pts)

In this problem, you must create a FFNN in Pytorch to classify emails from the Enron dataset as to whether they are spam or not spam ("ham"). For this problem, we will use `Glove` pretrained embeddings. The dataset and the embeddings are in the following location:

<https://drive.google.com/drive/folders/1cHR4VJuuN2tEpSkT3bOaGkOJrvIV-ISR?usp=sharing>

(You can also download the embeddings yourself from the web; but the dataset is one created just for this problem.)

### Part A: Prepare the Data (10 pts)

Compute the features of the emails (the vector of 100 floats input to the NN) vector based on the average value of the word vectors that belong to the words in it.

Just like the previous problem, we compute the 'representation' of each message, i.e. the vector, by averaging word vectors; but this time, we are using `Glove word embeddings` instead. Specifically, we are using word embedding 'glove.6B.100d' to obtain word vectors of each message, as long as the word is in the 'glove.6B.100d' embedding space.

Here are the steps to follow:

1. Have a `basic idea` of how Glove provides pre-trained word embeddings (vectors).
2. Download and extract word vectors from 'glove.6B.100d'.
3. Tokenize the messages ( `spacy` is a good choice) and compute the message vectors by averaging the vectors of words in the message. You will need to test if a word is in the model (e.g., something like `if str(word) in glove_model ...`) and ignore any words which have no embeddings.

### Part B: Create the DataLoader (15 pts)

Now you must separate the data set into training, validation, and testing sets, and build a 'Dataset' and 'DataLoader' for each that can feed data to train your model with Pytorch.

Use a train-validation-test split of 80%-10%-10%. You can experiment with different batch sizes, starting with 64.

Hints:

1. Make sure `__init__`, `__len__` and `__getitem__` of the your defined dataset are implemented properly. In particular, the `__getitem__` should return the specified message vector and its label.
2. Don't compute the message vector when calling the `__getitem__` function, otherwise the training process will slow down A LOT. Calculate these in an array before

creating the data loader in the next step.

3. The data in the `.csv` is randomized, so you don't need to shuffle when doing the split.

## Part C: Build the neural net model (25 pts)

Once the data is ready, we need to design and implement our neural network model.

The model does not need to be complicated. An example structure could be:

1. linear layer 100 x 15
2. ReLU activation layer
3. linear layer 15 x 2

But feel free to test out other possible combinations of linear layers & activation function and whether they make significant difference to the model performance later.

In order to perform "early stopping," you must keep track of the best validation score as you go through the epochs, and save the best model generated so far; then use the model which existed when the validation score was at a minimum to do the testing. (This could also be the model which is deployed, although we won't worry about that.) Read about `torch.save(...)` and `torch.load(...)` to do this.

Experiment with different batch sizes and optimizers and learning rates to get the best validation score for the model you create with early stopping. (Try not to look *too hard* at the final accuracy!) Include your final performance charts (using `show_performance_curves` ) when you submit.

Conclude with a brief analysis (a couple of sentences is fine) relating what experiments you did, and what choices of geometry, optimizer, learning rate, and batch size gave you the best results. It should not be hard to get well above 90% accuracy on the final test.