Large-Scale Data Mining: Models and Algorithms **ECE 219 Winter 24**
Prof. Vwani Roychowdhury
UCLA, Department of ECE

Project 1: End-to-End Pipeline to Classify News Articles

Due January 26, 2024, by 11:59 PM

Overview

Statistical classification broadly refers to the task of *learning to identify* a subset of categories that pertain to a data point (sample of text, an image, a video clip, a time-signal etc..) from a predefined (generally human-guided) larger set of categories. The model attempts to master the task given a training data set (*that is kept separate from an evaluation set*) in which each data point is pre-labeled with their "correct" category membership/s. In this project, we deal with the classification of text data. The project consists of building an end-to-end pipeline to classify samples of news articles and involves the following ML components:

1. **Feature Extraction:** Construction of TF-IDF representations of textual data;
2. **Dimensionality Reduction:** Principal Component Analysis (PCA) and non-Negative Matrix Factorization (NMF) - Generally necessary for classical ML methods
3. **Application of Simple Classification Models:** Applying common classification methods to the extracted features such as Logistic/Linear Classification and Support Vector Machines;
4. **Evaluation the Pipeline:** Evaluating and diagnosing classification results using Grid-Search and Cross Validation;
5. **Replace corpus-level features with pretrained features:** To apply pre-training to a downstream classification task and evaluate comparative performance.

Getting familiar with the dataset

Please access the dataset at this link. We are using a custom dataset that was designed specifically for this quarter [1]. **Note:** Do not attempt to open the downloaded file in Excel - the formatting of file when visualized in Excel might suggest the data is corrupted, but this is not true. Consider exploring the dataset using Pandas. You might find the following Pandas functions helpful (in no specific order): `read_csv`, `head`, `hist`, `shape`.

QUESTION 1: Provide answers to the following questions:

- *Overview:* How many rows (samples) and columns (features) are present in the dataset?

[1] This dataset was extracted from the search feature in The GDELT Project and a recursive crawler that traverses resulting news links

- *Histograms:* Plot 3 histograms on : (a) The total number of alpha-numeric characters per data point (row) in the feature `full_text`: i.e count on the x-axis and frequency on the y-axis; (b) The column `leaf_label` – class on the x-axis; (c) The column `root_label` – class on the x-axis.

- *Interpret Plots:* Provide qualitative interpretations of the histograms.

The two sets of labels `leaf_label` and `root_label` are hierarchically arranged as follows:

root_label	**sports**					**climate**				
leaf_label	basketball	baseball	tennis	football	soccer	forest fire	flood	earthquake	drought	heatwave

Binary Classification

For the first part of the project, we will be using only the `full_text` column as the *raw* features per sample (row) and the `root_label` column as the label for each sample. The `root_labels` are well-separated. Before continuing on, *please set the random seed as follows to ensure consistency:*

```
import numpy as np
import random

np.random.seed(42)
random.seed(42)
```

1 Splitting the entire dataset into training and testing data

In order to measure the performance of our binary classification model, we split the dataset into a training and a testing set. The model is trained on the training set and evaluated on the testing set. **Note:** Do not train on the testing set. We create the sets with a Pandas dataframe input that contains the entire dataset `df`. **Please make sure that the random seeds are set and the fraction of the test set is** 0.2:

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df[["full_text","root_label"]], test_size=0.2)
```

`train` and `test` contain the dataframes containing specific rows pertaining to the training data and testing data respectively.

QUESTION 2: Report the number of training and testing samples.

2 Feature Extraction

The primary step in classifying a corpus of text is choosing a good representation of each data point. Since the `full_text` column contains the *raw* features to describe each data point, we seek *a feature extraction module that encodes raw text features into processed computationally compatible features.*

A good representation should retain enough class-discriminating information post-processing to competitively perform classification, yet in the meantime, be concise to avoid computational intractability and over fitting.

A first model: The Bag-of-Words (BOW): One feature extraction technique is the "Bag of Words" model, where a document – in this case the `full_text` feature of one data point – is represented as a histogram of word frequencies, or other statistics, within a fixed vocabulary of words. The vocabulary is aggregated from the training set (process described below).

Compiling the Vocabulary: (a) Each raw feature (text segment) is split into sentences; (b) Each sentence is split into words; (c) The list of words PER sentence are passed jointly into a position tagger to identify the nouns, verbs, adjectives etc.; (d) These tags are used to *lemmatize*/stem [2] each word; (e) Words are filtered: Very rare or very frequent words (the number of documents they occur in, or the number of times they occur within a document) are removed, digits and punctuation-dominant words are removed, and stopwords, words contained in a database of common words are removed; (f) Remaining words are added to the vocabulary. Say that the selected set of words that compose the vocabulary form a set $\mathcal{W}$. For a dataset $\mathcal{D}$, the processed features can be collectively represented as a data matrix $X \in \mathbb{R}^{|\mathcal{D} \times |\mathcal{W}|}$. So each row captures the count of each word (the histogram) per data point.

An Example: If a test data point's raw feature contains the text,

> *"On Saturday, the NHL hockey team went to the school to volunteer and educate. Outreach is required for hockey, which is dying in popularity in recent years."*

and $\mathcal{W} = [``hockey'', ``volunteer'', ``sport'']$, the row in X corresponding to this data point would be $[2, 1, 0]$ because "hockey" appears twice, "volunteer" appears once, and "sport" does not appear at all (though it might appear in another data point. *Remember* the vocabulary is aggregated *across* the training set.)

During Testing: Each text sample in the testing set is similarly processed to those during training; **however words are no longer added to $\mathcal{W}$ – this was fixed during training.** Instead *if a word that exists in the vocabulary occurs in the processed words from a testing sample, its count is incremented in the resulting feature matrix X.*

To avoid adding to the vocabulary during testing *and* to avoid training on the test set in general please note *as a rule*: For most of this project you will be using the NLTK and sci-kit learn (sklearn) libraries for text processing and classifier design. In sklearn, in particular, *only* use the functions `fit_transform` and `fit` on the training set and *only* use `transform` on the testing set.

The better model: The Term Frequency-Inverse Document Frequency Model (TF-IDF): *"document" and "data point" are used interchangeably* While Bag-of-Words model continues to be used in many feature extraction pipelines, a *normalized* count vector that not only counts the number of times a word occurs in a document (*the term frequency*) but also scales the resulting count by the number of documents the word appears in (*the document frequency*) might provide a more valuable feature extraction approach.

The focus on the document frequency (and more correctly the inverse of it) encourages the

[2] *Lemmatization:* Lemmatization uses a prelearned vocabulary and morphological information of words in-sentence context, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.

Stemming: Stemming is a crude heuristic that removes the rightmost characters of words in the hope of mapping multiple word derivatives to the same root word.

If confronted with the word "saw", stemming might return just "s", whereas lemmatization would attempt to return either "see" or "saw" depending on whether the use of the word was as a verb or a noun.

feature vector to *discriminate* between documents *as well as* represent a document. TF-IDF does not only ask “What is the frequency of different words in a document?” but rather “What is the frequency of words in a document specific to that document and which differentiates it from other documents? A human reading a particular news article in the sports section will usually ignore the contextually dominant words such as “sport”, “competition”, and “player” despite these words being frequent in every news article in the sports section.

Such a context-based conditioning of information is widely observed. The human perception system usually applies a saturating function (such as a logarithm or square-root) to the actual input values into the vision model before passing it on to the neuronal network in the brain. This makes sure that a contextually dominant signal does not overwhelm the decision-making processes in the brain. The TF-IDF functions draw their inspiration from such neuronal systems. Here we define the TF-IDF score to be:

$$\text{TF-IDF}(d, t) = \text{TF}(t, d) \times \text{IDF}(t)$$

where $\text{TF}(d, t)$ represents the frequency of word (processed, lemmatized, otherwise filtered) t in document d, and inverse document frequency is defined as:

$$\text{IDF}(t) = \log\left(\frac{n}{\text{DF}(t)}\right) + 1$$

where n is the total number of documents, and $\text{df}(t)$ is the document frequency, *i.e.* the number of documents that contain the word t.

```python
import re
def clean(text):
        text = re.sub(r'^https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
        texter = re.sub(r"<br />", " ", text)
        texter = re.sub(r"&quot;", "\"",texter)
        texter = re.sub('&#39;', "\"", texter)
        texter = re.sub('\n', " ", texter)
        texter = re.sub(' u '," you ", texter)
        texter = re.sub('`',"", texter)
        texter = re.sub(' +', ' ', texter)
        texter = re.sub(r"(!)\1+", r"!", texter)
        texter = re.sub(r"(\?)\1+", r"?", texter)
        texter = re.sub('&amp;', 'and', texter)
        texter = re.sub('\r', ' ',texter)
        clean = re.compile('<.*?>')
        texter = texter.encode('ascii', 'ignore').decode('ascii')
        texter = re.sub(clean, '', texter)
        if texter == "":
            texter = ""
        return texter
```

QUESTION 3: Use the following specs to extract features from the textual data:

- Before doing anything, please clean each data sample using the code block provided above. This function helps remove many but not all HTML artefacts from the crawler's output. You can also build your own cleaning module if you find this function to be ineffective.
- Use the “english” stopwords of the `CountVectorizer`

- Exclude terms that are numbers (e.g. “123”, “-45”, “6.7” etc.)
- Perform lemmatization with `nltk.wordnet.WordNetLemmatizer` and `pos_tag`
- Use `min_df=3`

Please answer the following questions:

- What are the pros and cons of lemmatization versus stemming? How do these processes affect the dictionary size?
- `min_df` means *minimum document frequency*. How does varying `min_df` change the TF-IDF matrix?
- Should I remove stopwords before or after lemmatizing? Should I remove punctuations before or after lemmatizing? Should I remove numbers before or after lemmatizing? Hint: Recall that the full sentence is input into the Lemmatizer and the lemmatizer is tagging the position of every word based on the sentence structure.
- Report the shape of the TF-IDF-processed train and test matrices. The number of rows should match the results of Question 2. The number of columns should roughly be in the order of $k \times 10^3$. This dimension will vary depending on your exact method of cleaning and lemmatizing and that is okay.

The following functions in sklearn will be useful: `CountVectorizer`, TfidfTransformer, About Lemmatization and for the daring, Pipeline. Please refer to the discussion section notebooks for more guidance.

3 Dimensionality Reduction

After applying the above operations, the dimensionality of the representation vectors (TF-IDF vectors) is large. Classical learning algorithms, like the ones required in this section, however, may perform poorly with such high-dimensional data. Since the TF-IDF matrix is sparse and low-rank, as a remedy, one can *project the points from the larger dimensional space to a lower dimension.*

In this project, we use two dimensionality reduction methods: Latent Semantic Indexing (LSI) and Non-negative Matrix Factorization (NMF), both of which minimize the Mean Squared residual Error (MSE) between the original TF-IDF data matrix and a reconstruction of the matrix from its low-dimensional approximation. Recall that our data is the term-document TF-IDF matrix, whose rows correspond to TF-IDF representation of the documents, *i.e.*

$$\mathbf{X} = \begin{bmatrix} \texttt{tfidf}(d_1, t_1) & \cdots & \texttt{tfidf}(d_1, t_m) \\ \texttt{tfidf}(d_2, t_1) & \cdots & \texttt{tfidf}(d_2, t_m) \\ \vdots & \vdots & \vdots \\ \texttt{tfidf}(d_n, t_1) & \cdots & \texttt{tfidf}(d_n, t_m) \end{bmatrix}$$

.

Latent Semantic Indexing (LSI): The LSI representation is obtained by computing left and right singular vectors corresponding to the top k largest singular values of the term-document TF-IDF matrix $\mathbf{X}$.

We perform Singular Value Decomposition (SVD) on the matrix $\mathbf{X}$, resulting in $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$ where $\mathbf{U}$ and $\mathbf{V}$ orthogonal. Let the singular values in $\mathbf{\Sigma}$ be sorted in descending order, then the first k columns of $\mathbf{U}$ and $\mathbf{V}$ are called $\mathbf{U}_k$ and $\mathbf{V}_k$ respectively. $\mathbf{V}_k$ consists of the principle components of matrix $\mathbf{X}$ in the feature space.

Then we use $(\mathbf{X}\mathbf{V}_k)$ (which is also equal to $(\mathbf{U}_k\mathbf{\Sigma}_k)$) as the dimension-reduced data matrix, where rows still correspond to documents, only now each data point can be represented in a (far) lower dimensional space. In this way, the number of features is reduced. LSI is similar to Principal Component Analysis (PCA), and you can see the lecture notes for their relationships.

Having obtained $\mathbf{U}$ and $\mathbf{V}$, to reduce the *test* data, we ONLY multiply the test TF-IDF matrix $\mathbf{X}_t$ by $\mathbf{V}_k$, *i.e.* $\mathbf{X}_{t,\text{reduced}} = \mathbf{X}_t\mathbf{V}_k$. By doing so, we actually project the test TF-IDF vectors onto the previously learned principle components from training, and use the projections as the dimensionality-reduced data.

Non-negative Matrix Factorization (NMF): NMF tries to approximate the data matrix $\mathbf{X} \in \mathbb{R}^{n\times m}$ ($n = |D|$ docs and $m = |\mathcal{W}|$ terms) with $\mathbf{WH}$ ($\mathbf{W} \in \mathbb{R}^{n\times r}$, $\mathbf{H} \in \mathbb{R}^{r\times m}$). Concretely, it finds the non-negative matrices $\mathbf{W}$ and $\mathbf{H}$ s.t. $\|\mathbf{X} - \mathbf{WH}\|_F^2$ is minimized ($\|\mathbf{A}\|_F \equiv \sqrt{\sum_{i,j} A_{ij}^2}$).

Then we use $\mathbf{W}$ as the dim-reduced data matrix, and in the `fit` step, we calculate both $\mathbf{W}$ and $\mathbf{H}$. The intuition behind this is that we are trying to describe the documents (the rows in $\mathbf{X}$) as a (non-negative) linear combination of r topics:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{\mathsf{T}} \\ \vdots \\ \mathbf{x}_n^{\mathsf{T}} \end{bmatrix} \approx \mathbf{WH} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1r} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{mr} \end{bmatrix} \begin{bmatrix} \mathbf{h}_1^{\mathsf{T}} \\ \vdots \\ \mathbf{h}_r^{\mathsf{T}} \end{bmatrix}$$

Here we see $\mathbf{h}_1^{\mathsf{T}}, \ldots, \mathbf{h}_r^{\mathsf{T}}$ as r "topics", each of which consists of m scores, indicating how important each term is in the topic. Then $\mathbf{x}_i^{\mathsf{T}} \approx w_{i1}\mathbf{h}_1^{\mathsf{T}} + w_{i2}\mathbf{h}_2^{\mathsf{T}} + \cdots + w_{ir}\mathbf{h}_r^{\mathsf{T}}$, $i = 1, \ldots, n$.

Now how do we calculate the dim-reduced test data matrix? Again, we try to describe the document vectors (rows by our convention here) in the test data (call it $\mathbf{X}_t$) with (non-negative) linear combinations of the "topics" we learned in the `fit` step. The "topics", again, are the rows of $\mathbf{H}$ matrix, $\{\mathbf{h}_i^{\mathsf{T}}\}_{i=1}^r$. How do we do that? Just solve the optimization problem

$$\min_{\mathbf{W}_t \geq 0} \|\mathbf{X}_t - \mathbf{W}_t\mathbf{H}\|_F^2$$

where $\mathbf{H}$ is fixed as the $\mathbf{H}$ matrix we learned in the fit step. Then $\mathbf{W}_t$ is used as the dim-reduced version of $\mathbf{X}_t$.

QUESTION 4: Reduce the dimensionality of the data using the methods above:

- Plot the *explained variance ratio* across multiple different $k = [1, 10, 50, 100, 200, 500, 1000, 2000]$ for LSI and for the next few sections choose $k = 50$. What does the explained variance ratio plot look like? What does the plot's concavity suggest?
- With $k = 50$ found in the previous sections, calculate the reconstruction residual MSE error when using LSI and NMF – they both should use the same $k = 50$. Which one is larger, the $\|\mathbf{X} - \mathbf{WH}\|_F^2$ in NMF or the $\left\|\mathbf{X} - \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^{\mathsf{T}}\right\|_F^2$ in LSI and why?

4 Classification Algorithms

In this part, you are asked to use the dimensionality-reduced training data from LSI with your choice of k to train (different types of) classifiers, and evaluate the

trained classifiers with test data. Your task would be to classify the documents into two classes (for now a binary classification task) `sports` versus `climate`.

Classification Measures: Classification quality can be evaluated using different measures such as **precision**, **recall**, **F-score**, etc. Refer to the discussion material to find their definition.

Depending on application, the true positive rate (TPR) and the false positive rate (FPR) have different levels of significance. In order to characterize the trade-off between the two quantities, we plot the receiver operating characteristic (ROC) curve. For binary classification, the curve is created by plotting the true positive rate against the false positive rate at various threshold settings on the probabilities assigned to each class (let us assume probability p for class 0 and $1-p$ for class 1). In particular, a threshold t is applied to value of p to select between the two classes. The value of threshold t is swept from 0 to 1, and a pair of TPR and FPR is got for each value of t. The ROC is the curve of TPR plotted against FPR.

Support Vector Machines (SVM): Linear Support Vector Machines are seemingly efficient when dealing with sparse high dimensional datasets, including textual data. They have been shown to have good generalization and test accuracy, while having low computational complexity.

These models learn a vector of feature weights, $\mathbf{w}$, and an intercept, b, given the *training* dataset. Once the weights are learned, the label of a data point is determined by thresholding $\mathbf{w}^\mathsf{T}\mathbf{x} + b$ with 0, *i.e.* $sign(\mathbf{w}^\mathsf{T}\mathbf{x} + b)$. Alternatively, one produce probabilities that the data point belongs to either class, by applying a logistic function instead of hard thresholding, *i.e.* calculating $\sigma(\mathbf{w}^\mathsf{T}\mathbf{x} + b)$.

The learning process of the parameter $\mathbf{w}$ and b involves solving the following optimization problem:

$$\begin{aligned}\min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + \gamma\sum_{i=1}^{n}\xi_i \\ s.t. \quad & y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, \\ & \forall i \in \{1, \ldots, n\}\end{aligned}$$

where $\mathbf{x}_i$ is the ith data point, and $y_i \in \{0, 1\}$ is its class label.

Minimizing the sum of the slack variables corresponds to minimizing the loss function on the training data. On the other hand, minimizing the first term, which is basically a regularization term, corresponds to maximizing the margin between the two classes. Note that in the objective function, each slack variable represents the amount of error that the classifier can tolerate for a given data sample. The trade-off parameter γ controls relative importance of the two components of the objective function. For instance, when $\gamma \gg 1$, misclassification of individual points is highly penalized. This is called "Hard Margin SVM". In contrast, a "Soft Margin SVM", which is the case when $\gamma \ll 1$, is very lenient towards misclassification of a few individual points as long as most data points are well separated.

QUESTION 5: Compare and contrast hard-margin and soft-margin linear SVMs:

- Train two linear SVMs:
  - Train one SVM with $\gamma = 1000$ (hard margin), another with $\gamma = 0.0001$ (soft margin).
  - Plot the ROC curve, report the **confusion matrix** and calculate the **accuracy**, **recall**, **precision** and **F-1 score** of both SVM classifiers on the testing set. Which one performs better? What about for $\gamma = 100000$?
  - What happens for the soft margin SVM? Why is the case? Analyze in terms of the confusion matrix.

    - Does the ROC curve reflect the performance of the soft-margin SVM? Why?

- Use **cross-validation** to choose γ (use average validation [3] accuracy to compare): Using a 5-fold cross-validation, find the best value of the parameter γ in the range $\{10^k | -3 \leq k \leq 6, k \in \mathbb{Z}\}$. Again, plot the ROC curve and report the confusion matrix and calculate the **accuracy**, **recall precision** and **F-1 score** of this best SVM.

Logistic Regression: Logistic regression is a probability model that can be used for binary classification. In logistic regression, a logistic function ($\sigma(\phi) = 1/(1 + \exp(-\phi))$) acting on a linear function of the features ($\phi(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x} + b$) is used to calculate the probability that a data point belongs to a particular binary class, and during the training process, parameters $\mathbf{w}$ and b that maximize the likelihood of predicting the correct labels on the training data are learned.

One can also add a regularization term to the objective function, so that the goal of the training process is not only in maximizing the likelihood, but also in minimizing the regularization term, which is often some norm of the parameter vector $\mathbf{w}$. Adding regularization helps prevent ill-conditioned results and over-fitting, and facilitates generalization. A coefficient is used to control the trade-off between maximizing likelihood and minimizing the regularization term.

QUESTION 6: Evaluate a logistic classifier:

- Train a logistic classifier without regularization (you may need to come up with some way to approximate this if you use `sklearn.linear_model.LogisticRegression`); plot the ROC curve and report the confusion matrix and calculate the **accuracy**, **recall precision** and **F-1 score** of this classifier on the testing set.

- Find the optimal regularization coefficient:
    - Using 5-fold cross-validation on the dimension-reduced-by-SVD training data, find the optimal regularization strength in the range $\{10^k | -5 \leq k \leq 5, k \in \mathbb{Z}\}$ for logistic regression with L1 regularization and logistic regression with L2 regularization, respectively.
    - Compare the performance (accuracy, precision, recall and F-1 score) of 3 logistic classifiers: w/o regularization, w/ L1 regularization and w/ L2 regularization (with the best parameters you found from the part above), using test data.
    - How does the regularization parameter affect the test error? How are the learnt coefficients affected? Why might one be interested in each type of regularization?
    - Both logistic regression and linear SVM are trying to classify data points using a linear decision boundary. What is the difference between their ways to find this boundary? Why do their performances differ? Is this difference statistically significant?

Naïve Bayes Model: Scikit-learn provides a suite of Naïve Bayesian classifiers including `MultinomialNB`, `BernoulliNB`, and `GaussianNB`. Naïve Bayes classifiers use the assumption that features are statistically independent of each other when conditioned by the class the data point belongs to, in order to simplify the calculation for the Maximum a posteriori (MAP) estimation of the labels. That is,

$$P(x_i \mid y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m) = P(x_i \mid y) \qquad i \in \{1, \ldots, m\}$$

where x_i's are features and y is the label of the data point. The `MultinomialNB`, `BernoulliNB`, and `GaussianNB` use different underlying probability models.

[3] Validation is done on a subset of the previously obtained training data; NEVER on the testing data. k-fold CV conventionally means that the 5 folds are chosen without replacement - so each data point has a chance to be in the validation set. The same `train_test_split` function used to splice the larger dataset can be used to splice the training dataset into a training set per batch and a small validation set.

QUESTION 7: Evaluate and profile a Naïve Bayes classifier: Train a `GaussianNB` classifier; plot the ROC curve and report the confusion matrix and calculate the **accuracy**, **recall**, **precision** and **F-1 score** of this classifier on the testing set.

Grid Search of Parameters

Now we have gone through the complete process of training and testing individual parts of an end-to-end pipeline that performs binary classification.

Optimizing the hyperparameters for individual parts however does not imply that serially combining these optimized parts creates the optimal end-to-end pipeline. In this section, we combine everything we have done in previous sections into a GridSearch task.

In a "grid search", each hyperparameter to be tuned is placed along one axis of a grid. And the validation accuracy is compared for each combination of hyperparameters - essentially filling out the grid.

Note: This part of the project takes a long time to run because the number of options at each stage of the classification pipeline increases the number of possibilities combinatorially. Please give yourself and your computer time to complete this task.

QUESTION 8: In this part, you will attempt to find the best model for binary classification.

- Construct a `Pipeline` that performs feature extraction, dimensionality reduction and classification;
- The evaluation of each combination is performed with 5-fold cross-validation (use the average validation set accuracy across folds).
- In addition to any other hyperparameters you choose, your gridsearch must at least include:

Table 1: Minimum set of hyperparameters to consider for pipeline comparison

Module	**Options**
Loading Data	Clean the data
Feature Extraction	`min_df = 3` vs `5` while constructing the vocabulary; AND use Lemmatization vs Stemming as a compression module
Dimensionality Reduction	LSI ($k = [5, 30, 80]$) vs NMF ($k = [5, 30, 80]$)
Classifier	SVM with the best γ previously found vs Logistic Regression: L1 regularization vs L2 regularization, with the best regularization strength previously found vs `GaussianNB` Note: You can once again find the optimal hyperparameters for each classifier, but this is not *required*.
Other options	Use default

- What are the 5 best combinations? Report their performances on the testing set.

Hint: see these links for useful libraries: Examples and some more examples.

Multiclass Classification

We have so far been dealing with classifying the data points into two classes. In this part, we explore multiclass classification techniques through different algorithms. **You will need to use the column of labels marked `leaf_label` for this section.**

Some classifiers perform the multiclass classification inherently. As such, naïve Bayes algorithm finds the class with the maximum likelihood given the data, regardless of the number of classes. In fact, if the probability of each class label is computed in the usual way, then the class with the highest probability is picked; that is

$$\hat{c} = \arg\min_{c \in \mathcal{C}} P(c \mid \mathbf{x})$$

where c denotes a class to be chosen, and $\hat{c}$ denotes the optimal class.

For SVM, however, one needs to extend the binary classification techniques when there are multiple classes. A natural way to do so is to perform a one versus one classification on all $\binom{|\mathcal{C}|}{2}$ pairs of classes, and given a document the class is assigned with the majority vote. In case there is more than one class with the highest vote, the class with the highest total classification confidence levels in the binary classifiers is picked.

An alternative strategy would be to fit one classifier per class, which reduces the number of classifiers to be learned to $|\mathcal{C}|$. For each classifier, the class is fitted against the union of all the other classes. Note that in this case, *the unbalanced number of documents in each class should be handled.* By learning a single classifier for each class, one can get insights on the interpretation of the classes based on the features.

QUESTION 9: In this part, we aim to learn classifiers on the documents belonging to unique classes in the column `leaf_label`.

Perform Naïve Bayes classification and multiclass SVM classification (with both One VS One and One VS the rest methods described above) and report the **confusion matrix** and calculate the **accuracy**, **recall**, **precision** and **F-1 score** of your classifiers. How did you resolve the class imbalance issue in the One VS the rest model?

In addition, answer the following questions:

- In the confusion matrix you should have an 10×10 matrix where 10 is the number of unique labels in the column `leaf_label`. **Please make sure that the order of these labels is as follows:**

  ```
  map_row_to_class = {0:"basketball", 1:"baseball", 2:"tennis",
   ↪  3:"football", 4:"soccer", 5:"forest fire", 6:"flood",
   ↪  7:"earthquake", 8:"drought", 9:"heatwave"}
  ```

  Do you observe any structure in the confusion matrix? Are there distinct visible blocks on the major diagonal? What does this mean?

- Based on your observation from the previous part, suggest a subset of labels that should be merged into a new larger label and recompute the **accuracy** and plot the **confusion matrix**. How did the accuracy change in One VS One and One VS the rest?

- Does class imbalance impact the performance of the classification once some classes are merged? Provide a resolution for the class imbalance and recompute the **accuracy** and plot the **confusion matrix** in One VS One and One VS the rest?.

Word Embedding

In the previous parts of this project we relied on the TF-IDF vector representation of text for our feature extraction. There are two disadvantage of such a representation:

1. **TF-IDF vectors are not learned from a larger universal set of documents:** The fixed vocabulary is generated from the training corpus: The representation generation process is not well-transferable to other tasks or datasets. *As humans, we do not learn to identify critical features of a document from a limited set of documents; rather we have a more general corpus from which we model our feature extractors and apply them on a specific document.*

2. TF-IDF does not attempt to condition the representation of a word with respect to its context (words around it) within a document. It calculates the scores based on relative occurrences of words in the text.

In this section we extract word features using a framework called GLoVE: Global Vectors for Word Representation. These features are trained on a much larger text corpus to take into account the broader contexts of words while forming *word representations*, making the final vectors informative and useful in a downstream task without any supervision or knowledge of the data used in the task.

A word representation is most often a vector of fixed dimension that captures the meaning of a target word by relating its numerical vector to other vectors in an estimated vector space. GLoVE in particular, measures the *similarity* between a pair of words by their representations' Euclidean distance in the vector space of representations.

QUESTION 10: Read the paper about GLoVE embeddings - found here and answer the following subquestions:

(a) Why are GLoVE embeddings trained on the ratio of co-occurrence probabilities rather than the probabilities themselves?

(b) In the two sentences: "James is **running** in the park." and "James is **running** for the presidency.", would GLoVE embeddings return the same vector for the word **running** in both cases? Why or why not?

(c) What do you expect for the values of,

$$||\texttt{GLoVE["woman"] - GLoVE["man"]}||_2,\ ||\texttt{GLoVE["wife"] - GLoVE["husband"]}||_2 \text{ and } ||\texttt{GLoVE["wife"] - GLoVE["orange"]}||_2 \text{ ? Compare these values.}$$

(d) Given a word, would you rather stem or lemmatize the word before mapping it to its GLoVE embedding?

Please download the glove.6B.zip embedding file from the link below: `https://nlp.stanford.edu/projects/glove/`. Note that the zip file contains 4 different versions of GLoVE. Use glove.6B.300d.txt for this project. The code below shows how to use GLOVE embedding:

```
embeddings_dict = {}
dimension_of_glove = 300
with open("glove/glove.6B.300d.txt", 'r') as f: # if 'r' fails with unicode
 ↪  error, please use 'rb'
```

```
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], "float32")
        embeddings_dict[word] = vector
```

GLoVE embedding is a representation for **each word** while TF-IDF vector is **per text segment**. A text segment consists of many words.

Simple concatenation of all of the embeddings of words in that segment will result in a high-dimensional vector representation of varying sizes. Therefore, we need to design a feature construction procedure to make use of the word embeddings. The goals include:

1. The representation of segments of text of varying lengths – we plotted the histogram in Question 1 – should be mapped into the same dimension to facilitate downstream applications;
2. The representation should be a sufficient description of the content of a document, so as to support the classification.

QUESTION 11: For the binary classification task distinguishing the "sports" class and "climate" class:

(a) Describe a feature engineering process that uses GLoVE word embeddings to represent each document. **You have to abide by the following rules**:

- A representation of a text segment needs to have a vector dimension that CANNOT exceed the dimension of the GLoVE embedding used per word of the segment.
- You cannot use TF-IDF scores (or any measure that requires looking at the complete dataset) as a pre-processing routine.
- **Important:** In this section, feel free to use raw features from any column in the original data file not just `full_text`. The column `keywords` might be useful... or not. Make sure that your result achieves an accuracy of at least **92%**.
- To aggregate these words into a single vector consider normalization the vectors, averaging across the vectors.

(b) Select a classifier model, train and evaluate it with your GLoVE-based feature. If you are doing any cross-validation, please make sure to use a limited set of options so that your code finishes running in a reasonable amount of time.

If your model is performing poorly, try to improve your feature engineering pipeline and also revisit your hyperparameters.

QUESTION 12: Plot the relationship between the dimension of the pre-trained GLoVE embedding and the resulting accuracy of the model in the classification task. Describe the observed trend. Is this trend expected? Why or why not? In this part use the different sets of GLoVE vectors from the link.

Visualize the set of normalized GLoVE-based embeddings of the documents with their binary labels in a 2D plane using the UMAP library. Similarly generate a set of normalized **random** vectors of the same dimension as GLoVE and visualize these in a 2D plane with UMAP.

QUESTION 13: Compare and contrast the two visualizations. Are there clusters formed in either or both of the plots? We will pursue the clustering aspect further in the next project.

Submission

Your submission should be made to both of the two places: BruinLearn and Gradescope within BruinLearn.

BruinLearn Please submit a zip file containing your **report**, and your **codes** with a **readme file** on how to run your code to CCLE. The zip file should be named as

"Project1_UID1_UID2_..._UIDn.zip"

where UIDx's are student ID numbers of the team members. **Only one submission per team is required**. If you have any questions, please ask on Piazza or through email. Piazza is preferred.

Gradescope Please submit your report to Gradescope as well. Please specify your group members in Gradescope. It is very important that you assign each part of your report to the question number provided in the Gradescope template.

FAQ

- **My NMF objective function is not converging to a minimum fast/it is timing out. Help!**: If the iterative algorithm is taking too long to converge, please increase the default tolerance parameter and/or reduce the maximum number of iterations that the algorithm can run for.
- **My Gridsearch is taking too long. Help!**: If the GridSearch takes too long, you are welcome to use RandomizedSearch instead that samples the parameters in the grid space instead of running the experiments exhaustively. Note that we are not responsible for the correctness of the randomized search; i.e if the sampling is too low, you are not likely to get a set of parameters that are truly optimal.
- **In Q9, how many classes should we merge?** Scan your confusion matrices from the previous sections to see if any *pair* of classes stand out as being similar. Of course the combined label *needs to make semantic sense as well*; there is no use in combining all the classes into 1 for example, and claiming that the performance of your classifier is optimal. I recommend finding a pair of labels *within* a binary class that can be merged.
- **I am finding it difficult to run experiments on my local machine.** You are highly encouraged to learn to use Google Colab for your projects in this course. While Project 1 is doable on your local machine, GridSearch is expensive and will take on average 4 hours to complete (anywhere from 1/2 hour to 11 hours have been reported). Furthermore, future projects will occasionally use Neural Networks which are optimized to run on GPUs. These are provided for free on Google Colab.