Large-Scale Data Mining: Models and Algorithms **ECE 219 Spring 2018**
Prof. Vwani Roychowdhury TA: Tianyi Wang
UCLA, Department of ECE Behnam Shahbazi

Project 1: Classification Analysis on Textual Data

Due Friday Apr. 20, 2018 by 11:59 pm

Introduction

Statistical classification refers to the task of identifying a category, from a predefined set, to which a data point belongs, given a training data set with known category memberships. Classification differs from the task of clustering, which concerns grouping data points with no predefined category memberships, where the objective is to seek inherent structures in data with respect to suitable measures. Classification turns out as an essential element of data analysis, especially when dealing with a large amount of data. In this project, we look into different methods for classifying textual data.

In this project, the goal includes:

1. To learn how to construct tf-idf representations of textual data.
2. To get familiar with various common classification methods.
3. To learn ways to evaluate and diagnose classification results.
4. To learn two dimensionality reduction methods: PCA & NMF.
5. To get familiar with the complete pipeline of a textual data classification task.

Getting familiar with the dataset

We work with "20 Newsgroups" dataset, which is a collection of approximately 20,000 documents, partitioned (nearly) evenly across 20 different newsgroups (newsgroups are discussion groups like forums, which originated during the early age of the Internet), each corresponding to a different topic.

One can use `fetch_20newsgroups` provided by `scikit-learn` to load the dataset. Detailed usages can be found at `http://scikit-learn.org/stable/datasets/twenty_newsgroups.html`

In a classification problem one should make sure to properly handle any imbalance in the relative sizes of the data sets corresponding to different classes. To do so, one can either modify the penalty function (*i.e.* assign more weight to errors from minority classes), or alternatively, down-sample the majority classes, to have the same number of instances as minority classes.

QUESTION 1: To get started, plot a histogram of the number of training documents per category to check if they are evenly distributed.

Note that the data set is already balanced (especially for the categories we'll mainly work on) and so in this case we do not need to balance. But in general, as a data scientist you need to be aware of this issue.

Binary Classification

To get started, we work with a well separable portion of data, and see if we can train a classifier that distinguishes two classes well. Concretely, let us take all the documents in the following classes:

Table 1: Two well-separated classes

Computer Technology	`comp.graphics`	`comp.os.ms-windows.misc`	`comp.sys.ibm.pc.hardware`	`comp.sys.mac.hardware`
Recreational Activity	`rec.autos`	`rec.motorcycles`	`rec.sport.baseball`	`rec.sport.hockey`

Specifically, use the settings as the following code to load the data:

```
categories = ['comp.graphics', 'comp.os.ms-windows.misc',
              'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
              'rec.autos', 'rec.motorcycles',
              'rec.sport.baseball', 'rec.sport.hockey']

train_dataset = fetch_20newsgroups(subset='train', categories=categories,
                                   shuffle=True, random_state=42)
test_dataset  = fetch_20newsgroups(subset='test', categories=categories,
                                   shuffle=True, random_state=42)
```

1 Feature Extraction

The primary step in classifying a corpus of text is choosing a proper document representation. A good representation should retain enough information that enable us to perform the classification, yet in the meantime, be concise to avoid computational intractability and over fitting.

One common representation of documents is called "Bag of Words", where a document is represented as a histogram of term frequencies, or other statistics of the terms, within a fixed vocabulary. As such, a corpus of text can be summarized into a term-document matrix whose entries are some statistic of the terms.

First a common sense filtering is done to drop certain words or terms: to avoid unnecessarily large feature vectors (vocabulary size), terms that are too frequent in almost every document, or are very rare, are dropped out of the vocabulary. The same goes with special characters, common stop words (e.g. "and", "the" etc.), In addition, appearances of words that share the same stem in the vocabulary (e.g. "goes" vs "going") are merged into a single term.

Further, one can consider using the normalized count of the vocabulary words in each document to build representation vectors. A popular numerical statistic to capture the importance of a word to a document in a corpus is the "Term Frequency-Inverse Document Frequency (TF-IDF)" metric. This measure takes into account count of the words in the document, as normalized by a certain function of the frequency of the individual words in the whole corpus. For example, if a corpus is about computer accessories then words such as "computer" "software" "purchase" will be present in almost every document and their frequency is not a distinguishing feature for any document in the corpus. The discriminating words will most likely be those that are specialized terms describing different types of accessories and hence will occur in fewer documents. Thus, a human reading a particular document will usually ignore the contextually dominant words such as "computer", "software" etc. and give more importance to specific words. This is like when going into a very bright room or looking at a bright object, the human perception system usually applies a saturating function (such as a logarithm

or square-root) to the actual input values before passing it on to the neurons. 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 term 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 term t.

QUESTION 2: Use the following specs to extract features from the textual data:

- Use the default 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`

Report the shape of the TF-IDF matrices of the train and test subsets respectively.

Please refer to the official documentation of `CountVectorizer` as well as the discussion section notebooks for details.

2 Dimensionality Reduction

After above operations, the dimensionality of the representation vectors (TF-IDF vectors) ranges in the order of thousands. However, learning algorithms may perform poorly in high-dimensional data, which is sometimes referred to as "The Curse of Dimensionality". Since the document-term TF-IDF matrix is sparse and low-rank, as a remedy, one can select a subset of the original features, which are more relevant with respect to certain performance measure, or transform the features into a lower dimensional space.

In this project, we use two dimensionality reduction methods: Latent Semantic Indexing (LSI) and Non-negative Matrix Factorization (NMF), both of which minimize mean squared residual between the original data and a reconstruction 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.*

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

(which is the case for the output of `CountVectorizer` and `TfidfTransformer`).

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 SVD to the matrix $\mathbf{X}$, resulting in $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, $\mathbf{U}$ 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 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 that they can have (far) lower dimension. 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 learnt $\mathbf{U}$ and $\mathbf{V}$, to reduce the *test* data, we just 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 to the principle components, and use the projections as the dimension-reduced data.

NMF

NMF tries to approximate the data matrix $\mathbf{X} \in \mathbb{R}^{n\times m}$ (*i.e.* we have n docs and m 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. 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:

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

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

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, $\{\mathbf{h}_i^\top\}_{i=1}^r$. 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 3: Reduce the dimensionality of the data using the methods above

- Apply LSI to the TF-IDF matrix corresponding to the 8 categories with $k = 50$; so each document is mapped to a 50-dimensional vector.
- Also reduce dimensionality through NMF and compare with LSI:

  Which one is larger, the $\|\mathbf{X} - \mathbf{WH}\|_F^2$ in NMF or the $\left\|\mathbf{X} - \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top\right\|_F^2$ in LSI? Why is the case?

3 Classification Algorithms

In this part, you are asked to use the dimension-reduced training data from LSI 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 "Computer Technology" vs "Recreational Activity". Refer to Table 1 to find the 4 categories of documents comprising each of the two classes. In other words, you need to combine documents of those sub-classes of each class to form the set of documents for each class.

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.

SVM

Linear Support Vector Machines have been proved efficient when dealing with sparse high dimensional datasets, including textual data. They have been shown to have good generalization accuracy, while having low computational complexity.

Linear Support Vector Machines aim to 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 $\mathbf{w}^\top \mathbf{x} + b$ with 0, *i.e.* $sign(\mathbf{w}^\top \mathbf{x} + b)$. Alternatively, one produce probabilities that the data point belongs to either class, by applying a logistic function instead of hard thresholding, *i.e.* calculating $\sigma(\mathbf{w}^\top \mathbf{x} + b)$.

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

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

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

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, 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 tradeoff 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, which 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 4: Hard margin and soft margin linear SVMs:

- Train two linear SVMs and compare:
  - 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 classifier. Which one performs better?
  - What happens for the soft margin SVM? Why is the case?
- Use cross-validation to choose γ:

  Using a 5-fold cross-validation, find the best value of the parameter γ in the range $\{10^k | -3 \leq k \leq 3, k \in \mathbb{Z}\}$. 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

Although its name contains "regression", logistic regression is a probability model that is used for binary classification.

In logistic regression, a logistic function ($\sigma(\phi) = 1/(1+\exp(-\phi))$) acting on a linear function of the features ($\phi(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$) is used to calculate the probability that the data point belongs to class 1, and during the training process, $\mathbf{w}$ and b that maximizes the likelihood of the training data are learnt.

One can also add regularization term in the objective function, so that the goal of the training process is not only maximizing the likelihood, but also 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 facilitate generalization ability of the classifier. A coefficient is used to control the trade-off between maximizing likelihood and minimizing the regularization term.

QUESTION 5: Logistic classifier:

- Train a logistic classifier; plot the ROC curve and report the confusion matrix and calculate the **accuracy**, **recall precision** and **F-1 score** of this classifier.
- Regularization:
  - Using 5-fold cross-validation, find the best regularization strength in the range $\{10^k | -3 \leq k \leq 3, k \in \mathbb{Z}\}$ for logistic regression with L1 regularization and logistic regression 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, 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?

Naïve Bayes

Scikit-learn provides a type of classifiers called "naïve Bayes classifiers". They include `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, 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, *i.e.* components of a data point, and y is the label of the data point.

Now that we have this assumption, a probabilistic model is still needed; the difference between `MultinomialNB`, `BernoulliNB`, and `GaussianNB` is that they use different models.

QUESTION 6: 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.

Grid Search of Parameters

Now we have gone through the complete process of training and testing a classifier. However, there are lots of parameters that we can tune. In this part, we fine-tune the parameters.

QUESTION 7: Grid search of parameters:

- Construct a `Pipeline` that performs feature extraction, dimensionality reduction and classification;
- Do grid search with 5-fold cross-validation to compare the following (use test accuracy as the score to compare):

Table 2: Options to compare

Procedure	Options
Loading Data	remove "headers" and "footers" vs not
Feature Extraction	`min_df` = 3 vs 5; use lemmatization vs not
Dimensionality Reduction	LSI vs NMF
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`
Other options	Use default

- What is the best combination?

Hint: see
http://scikit-learn.org/stable/auto_examples/plot_compare_reduction.html
and
http://www.davidsbatista.net/blog/2018/02/23/model_optimization/

Multiclass Classification

So far, we have been dealing with classifying the data points into two classes. In this part, we explore multiclass classification techniques through different algorithms.

Some classifiers perform the multiclass classification inherently. As such, naïve Bayes algorithm finds the class with maximum likelihood given the data, regardless of the number of classes. In fact, 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 learnt to $|\mathcal{C}|$. For each classifier, the class is fitted against 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 8: In this part, we aim to learn classifiers on the documents belonging to the classes:

`comp.sys.ibm.pc.hardware`, `comp.sys.mac.hardware`,
`misc.forsale`, `soc.religion.christian`

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.

Submission

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 had any questions, please ask on Piazza or through email.