THE UNIVERSITY OF SYDNEY

ASSIGNMENT

TWO

Article Classification

Due date: 5pm Monday of week 9 (2018-05-07)

This assignment is worth 20% of your final assessment.

2.1 Objectives

1. Train and evaluate a statistical text classification system.
2. Analyse errors produced by a statistical text classification system.
3. Reason about how feature extraction approaches may enhance the representation of language for text classification, and validate this through empirical experimentation.
4. Consider how different classification tasks (e.g. sentiment vs. topic classification) may require different feature engineering.

2.2 Task

In this assignment you will train a classifier from the articles annotated in Assignment 1. You will:

1. use an existing implementation of a maximum entropy or logistic regression classifier;[1] week5
2. use a *sparse representation* of a feature vector;[2] sparse: does not contain 0
3. implement a baseline *bag-of-words* approach to feature extraction;[3]
4. quantify the classifier's performance with *ten-fold cross validation*; 2.6.1 K-fold Cross-Validation
5. *analyse errors* made by the baseline system to identify its limitations;
6. propose enhancements to the *feature extraction* approach and describe how they might improve the model;
7. perform a series of *experiments* exploring the effect of each enhancement on performance;
8. apply the same feature representations to a *second classification task* and comment on how they perform;
9. describe your experiments and results in a report.

In particular, you will report on at least *five* (5) experiments with enhancing the feature extraction, incorporating ideas from at least *three* (3) of the categories in subsection 2.2.1.

[1] These are equivalent for our purposes.
[2] We will not assess whether you actually use a sparse representation in your code, but it will likely avoid large training times.
[3] With either binary or count values. You may reuse an existing implementation of bag of words.

2.2.1 Categories of feature extraction enhancement

Bag of words ignores a lot of valuable content in the language. For example, it disregards: word ordering; compositional meaning and multi-word expressions; polysemy; the fact that words share stems but are then inflected; synonymy and other semantic relations; the structure of a document (are all words, sentences or paragraphs equally important?); the relationship between a document and its title.

This section lists some feature extraction approaches that you may consider as extensions of the bag-of-words baseline. The following are just examples for each category. You are welcome to report experiments not listed here.

Your experiments should incorporate at least three of these categories.

N-gram specification character n-grams; word n-grams for $n > 1$; words only from title / lead paragraph / all paragraphs; cleaning article text to reduce noise from boilerplate material.

Normalisation and weighting remove stop words; expand abbreviations; normalise numbers; stem words; lemmatise words; apply TF-IDF weighting.

Syntax and stylometry use POS tags or syntactic parses as an alternative representation, or to disambiguate lexical features (e.g. distinguishing *set-NOUN* from *set-VERB* from *set-ADJ*); group words together that form named entities or multi-word expressions. Stylometric features include: number of tokens; number of distinct tokens per token; average word length.

Lexical and multi-word semantics map words to WordNet synsets; map the document to titles of similar documents or categories from Wikipedia ("explicit semantic analysis"); disambiguate named entities using a named entity disambiguation system; represent documents by their latent topic signatures (via Latent Dirichlet Allocation or PCA); sentiment lexicon features.

Some of these enhancements will replace the baseline features, while others will add features to the bag of words, or to the features of the previous experiment. Please state whether you are adding new features or replacing, and why.

Some of these enhancements should make use of existing language processing resources, such as Stanford CoreNLP or spaCy for part of spech tagging; the WordNet interface in NLTK; MALLET topic modelling; a Porter stemmer implementation; a stop word list; or DBPedia Spotlight for named entity disambiguation. Please use Canvas Discussion to discuss and enquire about the various online resources you can use. Please make sure you mention the resources used when describing the features in your report.

You should motivate any experiments: what kind of effect do you think the change will have, and why? Ideally, you should motivate experiments by identifying the kinds of errors made by a simpler system.

2.3 Data

The data will be provided in CSV files with the following columns:

class The gold standard class for the article.

url The URL of the article. This is provided as metadata, but you need not use it.

title The title text of the article. This is provided both for feature extraction, and so you can refer to examples in your report.

body The body text of the article. The body will contain newline characters, so be sure to use a standard CSV reading tool. This should be the primary source of textual features.

You will receive two datasets:

`topic.csv` Articles annotated in Assignment 1. This is a multiclass classification task.

`virality.csv` Articles annotated as High or Low according to whether or not the article was in the top 25% of articles by Facebook shares on that web site for that month. This is a binary classification task.

Most of your analysis and experimentation should be performed on `topic.csv`. You will assess whether your experimental results on `topic.csv` are echoed when the same experiments are performed on `virality.csv`.

2.4 Code

You are free to use a programming language of your choice (or multiple) to implement the assignment.

The assessment of this assignment is not about the quality of your code. Rather, it is about how well you can set up, evaluate and analyse a typical statistical natural language processing experiment.

However, the *correctness* of your code is critical in producing meaningful experimental results.

2.4.1 Structure

To help us examining your code:

1. Please include in it a function/class/file for each experiment, named `experiment0` (or `Experiment0`) being the baseline, `experiment1`, `experiment2`, etc. This naming should correspond to feature extraction experiments detailed in your report.

   NB: These units need not run and evaluate the experiment in entirety. It would be better to structure your code in a modular way, so that only the aspect of the code that changes from experiment to experiment is contained in each `experiment*` unit.

2. Please also add a comment `CROSS VALIDATION` where you split the dataset into training and test sets (or where you use a library to do so).

3. Please list in a `README` file what software libraries need to be installed to run your code, and note how you would execute the experiments.

4. Please write your code assuming that the source data CSV files are available in the same directory as your `README`.

2.4.2 Avoiding plagiarism

If you copy non-trivial code from any source, even if you modify it, please clearly comment where the copied portion starts, where it ends, and where you copied it from. If in doubt, provide attribution!

2.4.3 Code submission instructions

You will submit your code to Canvas Assignments as a gzipped-tarball (.tgz) or a zipfile (.zip) by the due date. (Please do not use RAR or other compression formats.)

This will include your code and `README`. Please do *not* include the data we provide in your zip/tarball.

You may want to use a private repository on github.sydney.edu.au to build and track your code development. If, for instance, your GitHub.Sydney username is `jblo9999` and your repository is named `comp5046-a2`, you can download a zip copy of the master branch with the URL `https://github.sydney.edu.au/jblo9999/comp5046-a2/archive/master.zip`, and then upload this to Canvas.

2.5 Report

You will describe your experiments in a **3-4-page report (not including tables and/or diagrams in the page count)**.

An important part of this assignment is learning to identify and describe *relevant details*. There are an almost limitless combination of measures you can use or experiments you can do to analyse how your system performs. However, space *is* limited, so you must be selective. Once you have a baseline implementation, asking the right questions and using statistics that answer them concisely is the key to good marks.

2.5.1 Structure

Please include the following sections:

Baseline system

1. Briefly describe your baseline system.

2. Evaluate your baseline system quantitatively on the topic dataset. You must report at least accuracy averaged across the 10 folds. Other metrics may help you later identify in what way your experimental features affected performance.
3. Analyse the kinds of errors that the baseline system produces, referring to quantitative results and to examples from the data. Here you might want to explore the test instances that were correctly and incorrectly classified in one fold of cross validation.

Experiments

For each experiment:

1. Name the experiment in correspondence to code (e.g. `experiment1`).
2. State which category (subsection 2.2.1) the experiment falls under, if applicable, to help markers.
3. Describe how you alter the baseline system in this experiment.
4. Describe why you think this change might be helpful, or how you think it will affect results.

Results

1. Report average accuracy under 10-fold cross validation for each experiment. You may report other evaluation metrics in order to support your Discussion.
2. Identify the best-performing experiment(s) using accuracy or another metric.

Discussion

In about 150 words, answer the question: what kinds of errors did your best-performing experiment help to reduce? Make sure to refer to examples from the data and quantitative results.

Virality results

1. Report accuracy averaged over 10 folds for the baseline and each experiment when re-trained on the virality dataset.
2. Analyse whether the relative performances of each experiment match those on the topic dataset.

2.5.2 Report submission instructions

You will submit your report (as a DOC, PDF, etc.) to Turnitin through Canvas Assignments by the due date. *Your submission is not complete until you receive a "digital receipt".*

2.6 Algorithms and concepts

2.6.1 K-fold Cross-Validation

Once you have a system that classifies Wikipedia articles, you must estimate what accuracy you could expect from it on new articles that were not part of the training data. Training articles cannot be used because we wish to know how well the model *generalises*. A system that simply memorised the training examples would get 100% accuracy on them (assuming no two identical instances have different labels), but would not be useful on new data.

One way to estimate the performance of a system is to set aside some portion of the labelled data as *test data*, which the model is not trained on. However, a substantial number of instances must be set aside, because we want a large sample size so that we can measure the system's accuracy precisely. If we have only a small pool of labelled data, the loss of this data from the training set can hurt the system a lot.

Cross-validation is the practice of assessing how well a statistical learner *generalises* by training on only a portion of the instances and testing on the held-out remainder. K-fold cross-validation is a way to use more data for training while still estimating the system's robustness. The idea is to divide the data into k sections, and then train and evaluate the classifier k times, each time leaving aside a different section for evaluation.

Let's say we had 100 labelled instances. If we set aside only 10 for testing, most of our measurements would be statistically insignificant, as the sample size would be too small. However, setting aside 30 or 40 for testing to make more reliable accuracy estimates would leave us with only 60 or 70 training instances.

example

Using 10-fold cross validation, we instead divide the data into 10 sets of 10 documents each. We train the classifier on sets 1-9, and evaluate it on set 10, remembering the result, and then do the same again, training on all sets but number 9, and evaluating on section 9. Once we have evaluated 10 times, holding aside each section once, we will have 10 accuracy measures, which we can then average. We can also analyse the *variance* of these 10 accuracy measures, giving us an idea of the reliability of our accuracy estimates. This is important, because your classifier's performance will often vary simply by chance as you change your experimental settings.

When instances in a test set are correlated (i.e. not just a random sample from the whole dataset), you can get poor estimates (and high variance across folds). In the worst case, you might land up with training or test sets missing instances of one or more classes. Be sure to avoid this.

2.6.2 Maximum Entropy Classification

Classification with generalised linear models such as Maximum Entropy (basically the same as Logistic Regression) and Perceptron is introduced in lectures 5 and 6. Speech and Language Processing 3rd Ed chapter 7 goes into detail about Logistic Regresssion. MaxEnt / Logistic Regression is able to estimate a probability for each class, but here we use it to derive categorical predictions.

Prediction

The *decoding* or *prediction* process for classification with linear models involves multiplying a vector of weights $\mathbf{w}$ by a vector of features $\mathbf{f}(x, y)$ for each instance-class combination. The class y that gives the highest dot product is chosen as the prediction.

$$y^* = \underset{y}{\text{argmax}}\, \mathbf{w} \cdot \mathbf{f}(x, y) = \underset{y}{\text{argmax}} \sum_j w_j \cdot f_j(x, y) \tag{2.1}$$

Or, in pseudocode:

- $best \leftarrow -\infty$
- for each class y
  - $score \leftarrow 0.0$
  - for each feature f_j and its weight w_j
    - $score \leftarrow score + f_j(x, y) \times w_j$
  - if $score > best$, save $best \leftarrow score$ and $y^* \leftarrow y$

(If the features are the same for each class, this is equivalent to learning a *matrix* W of weights, with shape (#classes, #featuresperclass), which is multiplied by a feature vector $\mathbf{f}(x)$ to get a score for each class.)

Here is an example showing this decoding process for classifying John, represented by a feature vector of 4 elements per class for classes PER, LOC and ORG (shown in the first two rows). Here elements with value 0 are explicitly shown, unlike when using a sparse vector representation.

	PER				LOC				ORG			
$\mathbf{f}(PER\|John)$	1	0	1	0	0	0	0	0	0	0	0	0
$\mathbf{f}(LOC\|John)$	0	0	0	0	1	0	1	0	0	0	0	0
$\mathbf{w}$	3	1	1	0	-2	2	-1	0	-1	1	1	0

Once you have these values, you can add up all the associated weights for each active feature:

$$score(John, PER, \mathbf{w}) \quad = \quad (1 * 3) + (1 * 1) = 4 \tag{2.2}$$

$$score(John, LOC, \mathbf{w}) = (1 * -2) + (1 * -1) = -3 \quad (2.3)$$

In this case, y^* is PER.

Optimisation

The goal of training a maximum entropy model is to set the weights **w**. This is done to minimize the amount of error on the training data.

This does not mean the model generalises well to new data. To maximise generalisation performance (i.e. reduce overfitting), we make use of *regularised models* which don't just try to reduce the error, but also include a penalty that encourages most weights to be small.

One or two parameters, *regularisation coefficients*, need to be set to describe how many features must be small. Your system's performance may vary a lot depending on these coefficients, so the coefficient that maximises cross-validated performance should ideally be selected within each experiment. We will *not* assess this, but it may impact your system performance.

Thus if you want your evaluation to truly reflect the benefit of changing the feature set, make sure to optimise the regularisation coefficient using cross-validation within each training set. This is done, for instance, by `sklearn.linear_model.LogisticRegressionCV` in Python, `cv.glmnet` with `family='multinomial'` in R, `CVParameterSelection` over `Logistic`'s `R` parameter in Weka.

Minimising the amount of error (i.e. the logistic regression loss) may also not match the metric you measure performance by, e.g. an extrinsic measure of performance. This is mitigated by selecting the model that maximises your *extrinsic* measure under cross validation.

2.6.3 Evaluation Metrics

We use evaluation metrics to summarise performance. This allows us to select a "best system". It also allows us to compare the relevant strengths of alternative models in more detail.

For multiclass classification, the most useful measures to *maximise* are accuracy, or macro-averaged recall.[4] You may also consider calculating averaged F_1 with some classes (e.g. "other") removed.

Useful measures for analysis are per-class precision, recall, confusion matrices, etc.

Accuracy is the number of instances where your classifier agrees with the gold standard, divided by the total number of instances.

Precision, Recall and F-score are standard evaluation metrics for information retrieval and extraction, originally designed for binary classification/retrieval tasks. See assignment 1.

With multiple classes, there are two main ways to calculate overall P, R and F measures:

1. A **micro-average** gives equal weight to each *instance* being classified by pooling the contingency values across all classes;
2. A **macro-average** gives equal weight to each *class* being classified, but for multiclass (not multilabel) problems is identical to accuracy unless some "negative" class(es) is removed from the calculation.

Resources like The Stanford IR Book may be useful for describing these metrics, but are defined in terms of multi-label classification, which has some different properties. See also the scikit-learn documentation.

2.6.4 Analysis

While it is interesting to know how well your system performed (and especially how close it came to human performance), it is most useful to understand why it performed as it did, and its limitations.

You should use metrics to guide a qualitative understanding of the system's successful and failed predictions. Give specific examples. Inspecting errors can guide the design of further features and experimentation.

[4] Accuracy is identical to a weighted average over per-class recall, where the rate is the prevalence of each class in the gold standard.

You might also look at which features are most important. One approach to this involves looking at the largest-weighted (positively or negatively) features, though it is hard to account for how features interact. Another approach is an *ablation analysis* where you evaluate how performance changes when you remove each group of features.