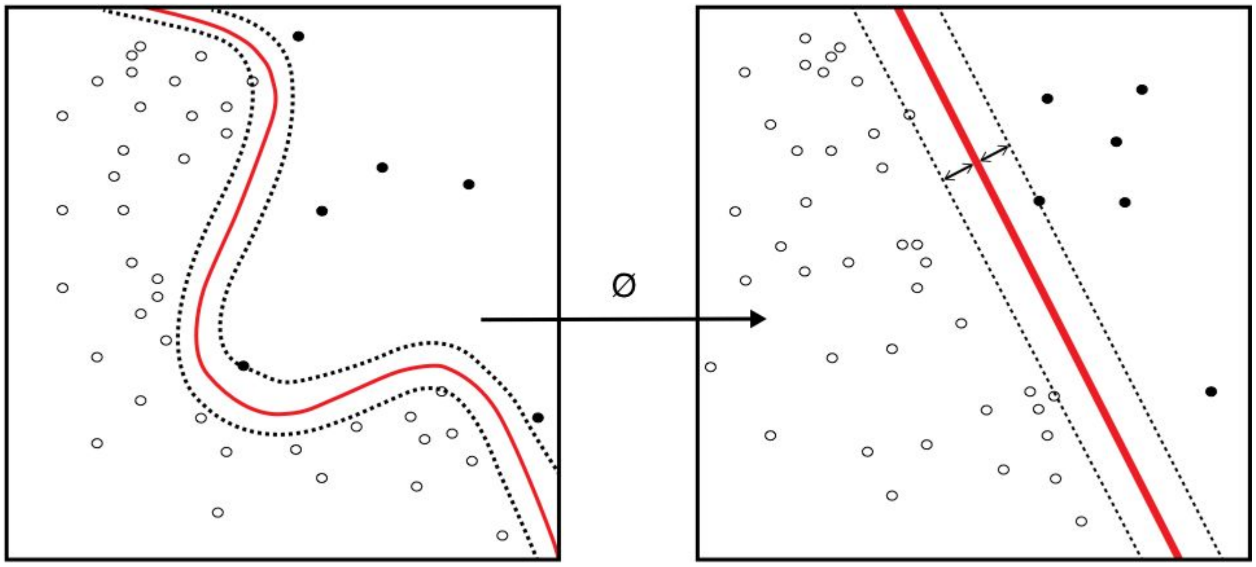
VIETNAMESE NEWS CLASSIFICATION

Why text data is linearly separable?



Report of Group III

July 7th, 2018

LE Ta Dang Khoa
BACH Quang Minh
NGUYEN Nhat Quang

I. INTRODUCTION

In this project, we investigate the task of classifying Vietnamese texts using SVM. More specifically, we're interested in doing binary classification of Vietnamese news texts.

Besides the usual context of performance comparison, we would like to propose and prove a very common heuristic in text-classification:

"Text data is often linearly separable."

Such heuristic is always quoted to justify the choice of linear-SVM in text-classification, often without proofs. Since proofs are desirable, we would like to contribute our humble effort in providing one.

However, our goal is not to do that rigorously, but to provide an intuitive explanation on why the heuristic is correct. This comprises of 3 tasks:

1. First, we build several classifiers and contrast their results to have an empirical proof.
2. Second, to establish confidence in such proof, we'll explain why our design decisions make sense along the way.
3. Third, we develop a theoretical justification for this empirical evidence to explain why this is not mere coincidence.

The bottom line is both good results on the practical side (which is the main goal of the final project), and good explanations for this very popular heuristic.

Remark:

For the theoretical justification, we will lay out the theoretical foundations and reasonings **during the presentation**, since the main project of this report is to explain, in details, the empirical process.

II. EXPERIMENTAL PLAN

For raw data, we use the same crawled data as other groups: "*Vietnamese News in 5 Categories*". But for the dataset, we subset that crawled data and divide into two classes, **Technology** and **Non-Technology**, each comprises of more than 9000 texts.

Next, we **pre-process** in this order:

1. Segmentation using "*VnCoreNLP*" library in Java.
2. Remove special characters and numbers, leaving only lower-cased words.
3. Remove stopwords using a reliable "*vietnamese-stopwords*" list.
4. Shuffle the pre-processed dataset and do a 80/20 train-test split.

The next step is **vectorization** and **feature-selection**. For the former, we use *TF-IDF representation*. For the latter, we use and contrast the very popular methods of *Word Thresholding* and *Mutual Information.*

Finally, we implement and contrast the two approaches of SVM: *Linear-SVM* and *RBF-kernel*. The metrics are prediction accuracy, time of **hyper-parameter tuning** (with *randomized search*) and time of **model training**.

These contrasts of feature-selection methods and learners will produce empirical results useful for our proof.

III. EMPIRICAL EXPERIMENT

A. Data Preparation & Data Pre-processing

For SVM to work well, we need a balanced dataset of medium size. That's why we draw around 9000 texts in each interested class, **Tech** and **Non-Tech**. That's data preparation. For pre-processing, we do as follows:

1. **Do Vietnamese text-segmentation**
   - **Reason**: One property of text-data is the *appearance-or-not* of a word implies text's meaning. This works well in English, but not in Vietnamese. For example, the single word "*hospital*" in English corresponds to a two-word "*bệnh viện*" in Vietnamese.

     Hence, this is required to level the playing-field between Vietnamese and English text-classification.

   - **Implement**: Write a Java program that utilizes VnCoreNLP to read all raw texts in the original directory and write-out segmented texts in another directory.

2. **Removing all digits from segmented texts**
   - **Reason**: Numbers in document are meaningful to retrieving information but not useful to classify between documents.
   - **Implement**: Define a function that mapping all number to null string and remove.

3. **Removing special characters**
   - **Reason**: Special characters can be punctuations and any character that contains no meaning. This is a list of special characters that we used in this step:

     ″ … ‘ ’ [ ] { } ` ~ | ' ! @ # $ % ^ & * ( ) + = < > ? , . / : \ “ ” ; "

   - **Implement**: Using regular expression to replace special characters with null string.
   - **Notes**: Firstly, some characters may look identical to human eyes but not to machine. Secondly, because the resulting tokens from "*VNcoreNLP*" using "_" to combining 2 syllables into a word, for example: "chính_phủ", therefore "_" is excluded in the special characters list in this step.

4. **Removing patterns that make a meaningful word become noise**
   - **Reason**: It is observed that some tokens still attached with some “weird” patterns to the head of a word and make them become noise to classifier. For example:

     _-sỹ_thanh

     \u200b\u200bcông

   - **Implement:** Because we only want to remove the “weird” patterns at the beginning of a word, therefore, we define a regular expression to remove these “weird” patterns if any appeared at the beginning of a word.

5. **Removing stopwords**
   - **Reason:** Like english, some words are meaningless for classification but they appeared in almost any documents; thus, classifier cannot make use of them perform the task.
   - **Implement**: The list of stopword are taken from this github. We use the file: vietnamese-stopwords-dash.txt for stopword vocabulary because of the result of the tokenizer. Furthermore, To ensure that after removing “weird” patterns, the remainings are still contain special characters or an alphabet (a-z), we add these to expand the stopword vocabulary.

The result of the preprocessing is acceptable. Due to the time limit, not all “weird” patterns and special characters are found and removed. However, these can be justified in the next step: vectorization and feature selection.

B. Vectorization & Feature Selection

For transforming a text to *TF-IDF representation*, we use *scikit-learn* **CountVectorizer** to build a term-frequency matrix, then feed that matrix to a **TfidfTransformer** to produce TF-IDF representations.

During this process, we apply feature-selection after having the term-frequency matrix. For *Word Thresholding*, we set **min_df** and **max_df** in *CountVectorizer*, and for *Mutual Information*, we use **mutual_info_classif** of *scikit-learn*.

Also, for SVM to work well, we should scale the TF-IDF representation also. In this project, we use **sublinear-tf scaling** so that no single-word can out-weight others simply occurring very often.

1. **Word Thresholding:**

Assumptions:

- A document can be said to belong to a specific topic if it contains words found in topic vocabulary. However, if a word appears too little in a collection, it cannot be used as a feature to classify between topics. Vice versa, If a word appear too much in a collection, it cannot used as a distinct feature to distinguish between topics.

For our problem:

- **min_df:** is set to equal 6. It means that a word is removed if it appears in less than 6 documents. We choose 6 because it can remove leftover noise from preprocessing step.
- **max_df:** is set to equal 0.75. It means that a word is removed if it appears in more than 75% number of documents in the dataset. We justify 0.75 = 0.5 + 0.25 as follows:
  - i. Our dataset is balancedly drawn, which mean that number of documents of TECH is approximately equal to that of NON-TECH. (around 50% of total documents)
  - ii. It is reasonable to say that if a word in TECH topic appears in about more than a half of number of document in NON-TECH (25% of total documents), it is not distinguishable enough.

After applying word threshold:

- **Number of features:** 16419
- The word threshold are able to remove some special characters such as

아 무 도 행 복 할 수

航 空 站

文 字

2. **Mutual Information:**

The mutual information measures the dependence between two random variables. It quantifies the amount of information obtained about one random variable, through the other random variable.

Formally, the mutual information of two discrete random variables X and Y can be defined as:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right),$$

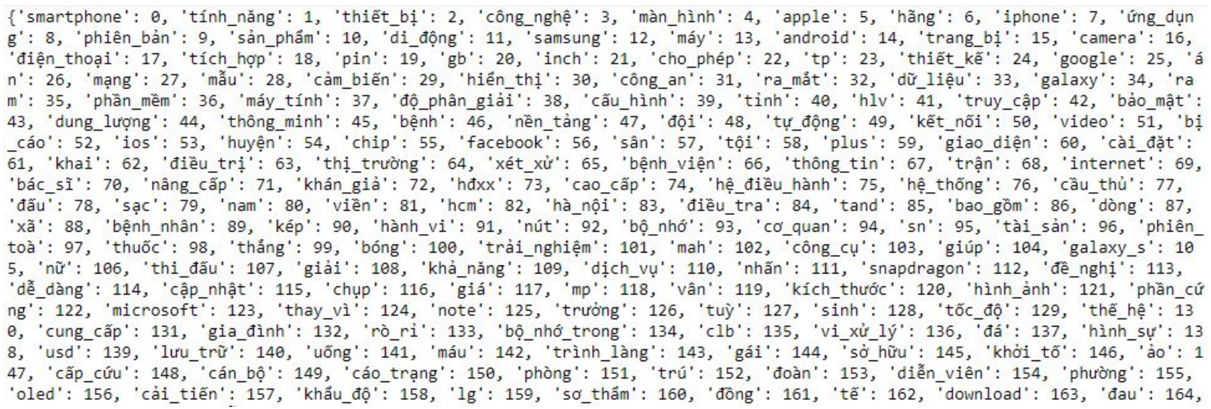
where $p(x,y)$ is the joint probability function of X and Y, and $p(x)$ and $p(y)$ are the marginal probability distribution functions of X and Y respectively.

In our case, the random variable X could describe the present of a feature (a word) in a document. And the random variable Y could represent the target class of a document, whose values are discrete.

It is hard to set a threshold for the mutual information values. Therefore, the approach is:

- ➢ Compute the mutual information values on the train set.
- ➢ Sort the features by their mutual information values descending
- ➢ Construct a grid as [1, 5, 10, 50, 100, 500, 1000, 5000, 10000] to build 9 dictionaries with the corresponding number of features
- ➢ Let a cross-validation process to decide the best dictionary.

After running the cv process, we get a list of 5000 features with the highest mutual information values among 81192 features. Taking a glance at these features, it is found that most of them mentioned to the topic of interest, which is TECH.



```
{'smartphone': 0, 'tính_năng': 1, 'thiết_bị': 2, 'công_nghệ': 3, 'màn_hình': 4, 'apple': 5, 'hãng': 6, 'iphone': 7, 'ứng_dụn
g': 8, 'phiên_bản': 9, 'sản_phẩm': 10, 'di_động': 11, 'samsung': 12, 'máy': 13, 'android': 14, 'trang_bị': 15, 'camera': 16,
'điện_thoại': 17, 'tích_hợp': 18, 'pin': 19, 'gb': 20, 'inch': 21, 'cho_phép': 22, 'tp': 23, 'thiết_kế': 24, 'google': 25, 'á
n': 26, 'mạng': 27, 'mẫu': 28, 'cảm_biến': 29, 'hiển_thị': 30, 'công_an': 31, 'ra_mắt': 32, 'dữ_liệu': 33, 'galaxy': 34, 'ra
m': 35, 'phần_mềm': 36, 'máy_tính': 37, 'độ_phân_giải': 38, 'cấu_hình': 39, 'tỉnh': 40, 'hlv': 41, 'truy_cập': 42, 'bảo_mật':
43, 'dung_lượng': 44, 'thông_minh': 45, 'bệnh': 46, 'nền_tảng': 47, 'đội': 48, 'tự_động': 49, 'kết_nối': 50, 'video': 51, 'bị
_cáo': 52, 'ios': 53, 'huyện': 54, 'chip': 55, 'facebook': 56, 'sân': 57, 'tội': 58, 'plus': 59, 'giao_diện': 60, 'cài_đặt':
61, 'khai': 62, 'điều_trị': 63, 'thị_trường': 64, 'xét_xử': 65, 'bệnh_viện': 66, 'thông_tin': 67, 'trận': 68, 'internet': 69,
'bác_sĩ': 70, 'nâng_cấp': 71, 'khán_giả': 72, 'hđxx': 73, 'cao_cấp': 74, 'hệ_điều_hành': 75, 'hệ_thống': 76, 'cầu_thủ': 77,
'đấu': 78, 'sạc': 79, 'nam': 80, 'viền': 81, 'hcm': 82, 'hà_nội': 83, 'điều_tra': 84, 'tand': 85, 'bao_gồm': 86, 'dòng': 87,
'xã': 88, 'bệnh_nhân': 89, 'kép': 90, 'hành_vi': 91, 'nút': 92, 'bộ_nhớ': 93, 'cơ_quan': 94, 'sn': 95, 'tài_sản': 96, 'phiên_
toà': 97, 'thuốc': 98, 'thắng': 99, 'bóng': 100, 'trải_nghiệm': 101, 'mah': 102, 'công_cụ': 103, 'giúp': 104, 'galaxy_s': 10
5, 'nữ': 106, 'thi_đấu': 107, 'giải': 108, 'khả_năng': 109, 'dịch_vụ': 110, 'nhấn': 111, 'snapdragon': 112, 'đề_nghị': 113,
'dễ_dàng': 114, 'cập_nhật': 115, 'chụp': 116, 'giá': 117, 'mp': 118, 'vân': 119, 'kích_thước': 120, 'hình_ảnh': 121, 'phần_cứ
ng': 122, 'microsoft': 123, 'thay_vì': 124, 'note': 125, 'trưởng': 126, 'tuỳ': 127, 'sinh': 128, 'tốc_độ': 129, 'thế_hệ': 13
0, 'cung_cấp': 131, 'gia_đình': 132, 'rò_rỉ': 133, 'bộ_nhớ_trong': 134, 'clb': 135, 'vi_xử_lý': 136, 'đá': 137, 'hình_sự': 13
8, 'usd': 139, 'lưu_trữ': 140, 'uống': 141, 'máu': 142, 'trình_làng': 143, 'gái': 144, 'sở_hữu': 145, 'khởi_tố': 146, 'ảo': 1
47, 'cấp_cứu': 148, 'cán_bộ': 149, 'cáo_trạng': 150, 'phòng': 151, 'trú': 152, 'đoàn': 153, 'diễn_viên': 154, 'phường': 155,
'oled': 156, 'cải_tiến': 157, 'khẩu_độ': 158, 'lg': 159, 'sơ_thẩm': 160, 'đông': 161, 'tế': 162, 'download': 163, 'đau': 164,
```

The figure shows the first 164 features in the 5000-feature list.

C. Hyper-parameter Tuning & Model Training:

We use randomized search for hyper-parameter tuning. The cross-validation is set at 5-fold, which will perform on just a random (representative) subset of the training set. After finding the optimal set parameters, we'll train the model on the full training set, and apply that model on the test set to find the prediction accuracy.

1. THE LINEAR KERNEL:

Thi Linear-SVM assumes the data at hand is linearly separable. However, to deal with some outliers, it introduces the **C hyper-parameter** to regularize the algorithm tolerance of such outliers. A larger **C** means less tolerance and we have a "harder" margin SVM.

For the tuning subset, we random sample 2000 texts in the training dataset. For the distribution of *C*, we use uniform distribution in the range of 1 to 10. This produces optimal *C* of **1.2** and at training accuracy of **99.4%**.

2. THE RBF KERNEL:

RBF kernel is a function to measure the distance between 2 vectors in an infinite-dimensional space, where the 2 vectors are non-linear transformed

$$K(x,x') = \exp\left(-\gamma||x - x'||^2\right)$$

The parameter needed to define is γ. Intuitively, with a small γ, two vectors are still consider similar even the distance $||x - x'||^2$ are large. And with a large γ, the two vectors are considered similar only they are close to each other.

To tune our classifier, we randomly select a subset of 2000 documents from our collection and use Randomized Search Cross Validation from sklearn for selecting optimal Gamma γ and penalty parameter C.

- The gamma γ is drawn from reciprocal distribution of $(a = 0.001,\ b = 0.1)$
- The penalty C is drawn from uniform distribution of $(a = 1,\ b = 10)$

Other specification for RandomizedSearchCV:

- Number of cross-validation: 5
- Number of iteration: 60
- Verbose: 2

	RBF with Word Threshold	**RBF with Mutual Information**
Accuracy on Train	99.11%	98.89%
Optimal Gamma γ	0.0518	0.0469
Optimal Penalty C	7.2605	10.3501

3. **<u>ON RANDOMIZED SEARCH</u>**:

1. Why did we pick **randomized-search** instead of **grid-search**?

Because randomized-search guarantees 5% within the true optimal in 95% of the time, using just 60 iterations. On the other hand, grid-search doesn't guarantee anything.

2. Why use uniform-distribution on (1, 10) for **C**?

*First, we want the parameter to jump in the interval of **tenth**. Second, since C controls the violations of outliers, we want to have equal chance of selecting any number within the search range, we pick the uniform distribution. The range (1, 10) is implied by best practices (using the typical scale of TF-IDF).*

3. Why use reciprocal-distribution on (0.001, 0.1) for **gamma**?

*Again, we want the parameter to jump in the interval of **tenth**. Second, since C controls the exponents of the RBF function, we want to have equal chance of selecting any number **<u>regardless of scale</u>** within the search range, we pick the reciprocal distribution. The range (0.001, 0.1) gives us natural exponent closes to 1, which prevents the decision boundary from fluctuating strongly.*

IV. EMPIRICAL RESULTS

KERNEL	ACCURACY	RUNNING TIME
Linear - Word Threshold - Features: **16 419**	**97.64%**	Tune ~ **2 mins** Train = **0.5s** Prediction* = **0.007s**
RBF - Word Threshold - Features: **16 419**	**97.69%**	Tune ~ **23.7 mins** Train = **56.80s** Prediction* = **10.57s**
RBF - Mutual Information (MI) - Features: **5 000**	**97.63%**	Tune ~ **9.9 + 2.95** mins** Train = **37.61s** Prediction* = **8.22s**

* : Prediction time is measure for predicting 3723 documents in test set.
**: One running time for tuning mutual information, please see section 2 of the Comments.

Comments:

1. The accuracies of the three pipelines are very similar. The RBF with Word Threshold feature selection receives the highest score, while the RBF with MI sees the lowest score.

2. The real difference is in the running time. In terms of tuning, training and prediction time, Linear SVM with Word Threshold showed to be the fastest approach, while the other two need a significantly extra amount of time to finish the process.

Moreover, the RBF with MI **heuristically** compute the MI values **ONLY once**. A rigorous approach would require this tuning to be included in the randomized search, a tremendous amount of time is expected.

3. Based on these results, a heuristic for text classification is to use a simple Linear SVM with Word Threshold feature selection. Because it is the best **effort-saving** and **time-saving** method for building a classifier with an **adequate level of accuracy**.

V. CONCLUSIONS

There is a common heuristic in text-classification:

"Text data is often linearly separable."

And in this project, again we reinforce the heuristic on the domain of Vietnamese news classification.

In this project, we build binary SVM classifiers for 2 classes: TECH and NON-TECH with different kernel and different method of feature selections:

1. Linear Kernel with Word Threshold
2. RBF Kernel with Word Threshold
3. RBF Kernel with Mutual Information

The results show that the accuracies of the classifiers are not very different from each other; however, linear kernel requires a minimal resource when tuning (2 minutes), training (0.5 seconds) and predicting (0.007 seconds). This shows how effective linear kernels are in text classification.

END