COMS4045A/7066A
Natural Language Processing/Technology (NLP) 2022
Project

October 2022

Table of Contents

General Project Details

1. Projects should be done and submitted individually.
2. Select one out of the three projects.
    a. Spam Classification with Naïve Bayes
    b. Language generation with N-grams
    c. Named-entity recognition (Token classification) using Logistic Regression or Neural Networks
3. If you have any questions, please visit the lab sessions, or post your questions on the 'Lab session' channel on Teams.

Submission Instructions:

1. You are to submit only one project.
2. Each project is 100 marks.
3. Submit a Jupyter notebook. Your notebook should:
    a. Clearly state the project you are working on.
    b. Contain code cells and answers to all questions.
    c. Contain your name and student number.
4. All code cells and answers to questions, must be clearly marked with the full question number.
5. Your notebook should be able to run as-is without any errors (e.g. commands to import or download libraries, packages etc. Please ensure that you verify this, your notebook will be graded based on the output generated as it was submitted. E.g., no import issues will be fixed.
6. Zip up your notebook alongside the datasets provided and other files if needed,
7. Save your zip file and your notebook with your ***Studentnumber.***
8. You are allowed to use any libraries you wish while following the given instructions for each project. Ensure that the libraries are publicly available or provided.

Submission Date:

The due date for the project is **17th November**, 2022.
Extension: Submissions will still be accepted until 23rd November, 2022, but for each day beyond 17th November, you will **lose 5 points**.
There will be no further extensions and **no submissions will be accepted beyond the 23rd of November**.

Credit: Tutors
Michael Beukman;
Manuel Fokam

Project 1 – Spam Classification Using Naïve Bayes

Summary:
The overall aim of this project is to train and optimize a spam classifier using the Naive Bayes classification model. In summary, you will have to

- Implement a Naive Bayes model
- Investigate the effect of using different features
- Evaluate the performance of different models
- Optimize the model for better performance.

Dataset:
The dataset for this project was adapted from the original dataset here. However, use the given datafiles for this project: *train.csv*, *test.csv* and *val.csv* files.

Evaluation:
For each evaluation task that needs to be carried out, evaluate the given model using model accuracy, f1 score, precision, and recall.

Tasks to be graded:

1. Preprocess the data, tokenize the text, and get a list of words for each sentence. [10 marks]
2. Train a standard Naive Bayes model. Call this **Model1** [10 marks].
3. Train **4** additional Naïve Bayes models with the following variations:
    a. Use only the 10 most frequent words as features. Call this **Model2** [10 marks].
    b. Use only the 100 most frequent words as features. Call this **Model3** [10 marks].
    c. Remove the 100 most frequent words from the features. Call this **Model4** [10 marks].
    d. Use only the subject line (see the data) as the feature set. Call this **Model5** [10 marks].
4. Evaluate the performance of the first model and all 4 variations using the validation set.
    a. Calculate the evaluation metrics [10 marks].
    b. Compare all 5 models [5 marks].
    c. Make a recommendation about which model to use with reasons. [5 marks].
5. Now, evaluate all the models with the test set
    a. Calculate the evaluation metrics [10 marks].
    b. Is your recommendation (in 4c above) still is valid? Explain [5 marks].
6. Implement at least one more variation to improve the performance of the best model. For instance, use frequent n-grams as features. The improvement should be on at least one of the evaluation metrics. Cleary describe the improved model and explain how it is an improvement. [5 marks]

Project 2 – Language Generation Using N-Gram Models

Summary:
The goal of this project is to use a language model to generate new text and evaluate how different parameters affect the generation. Specifically, you are given 4 datasets, the plays of Shakespeare, some Monte Python scripts, Euclid's Elements, and the Git source code. Further, you should investigate the effect of changing the size of the n-grams.

Dataset:

1. euclid_elements/book.txt
2. monte_python/scripts.txt
3. The concatenation of all the .txt files in the shakespeare/ directory.
4. git/source.c

Tasks to be graded:
The following subtasks will be graded:

1. Read in all the text files given in the dataset, storing each separately (as 4 separate variables) [10 marks].
2. Create a variable that is the concatenation of the entire dataset [5 marks].
3. Perform preprocessing on the extracted texts (5 variables) from Tasks 1 and 2. The result of this should be a 2D list. Each element in this list should be a "sentence", consisting of a list of words/tokens. The steps are as follows: [15 marks]
    a. First, change the text to lowercase.
    b. Then, to obtain the sentences, a simple technique would be to split the text on two newlines ("\n\n") to obtain a list of sentences.
    c. Each sentence can then be split on whitespace (e.g. string.split() in Python).

    If you want, you can use more complex tokenisations.
4. For each dataset and $n \in \{2, 3, 4\}$, create an n-gram language model and train it on the preprocessed texts [30 marks].
    a. In all you should have 15 models.
    b. If you encounter memory problems, you can use the lab computers or Kaggle notebooks or Google Colab to train the models. Otherwise, use only a smaller part of each text file to train your model (if you do, note this in the submitted notebook.)
5. Generate 2 sentences using each of the above models using different initial contexts (start words) [30 marks].
    a. In all you should generate 30 sentences.
6. Write a short paragraph comparing these sentences based on the training data and size of the n-gram [10 marks].

Hints and Resources:

- consider using *nltk.lm*.
- https://www.nltk.org/api/nltk.lm.html
- https://www.kaggle.com/code/valkling/monty-python-scripts-database-to-text/data

Project 3 - Named Entity Recognition Using Logistic Regression or Neural Networks.

Summary:
This project will focus the task of *Named Entity Recognition* (NER). This is effectively performing a multi-class classification on each word.

This is a token-classification task, where for each word in a sentence, we must classify it as being one of the following:

- **DATE:** A date/time
- **PER:** A person
- **LOC:** A location
- **ORG** An organisation
- "Other": None of the above.

The approach here will be simple:

- Perform standard classification using the word vectors as input.
- Use surrounding words, around the candidate as extra features.
- Make use of pre-trained word vectors - the FastText [1] word vectors to represent the words as numerical quantities (embeddings).

Dataset:

We will specifically consider the Swahili language in this project, but feel free to explore other languages as well. The dataset is provided in the *train.txt*, *text.txt* and *dev.txt* files attached to the project. It was obtained from the *MasakhaNER* [2] dataset, which has NER datasets for many languages, including Swahili.

Tasks to be graded:

1. Read in the provided `train.txt` file. The format is as follows: Each row consists of one word, followed by a space, followed by a label (e.g. PER, O, etc.). Sentences are separated by two newlines. Read this data into a list of sentences, with each sentence being a list of words. Also create a parallel list with the same structure, consisting of the labels. You might need to study the data first. [5 marks]

2. Read in the FastText vectors from the given file.[1] The structure here is: Each row contains a different word. Each line has the word, followed by a space, followed by 100 space-separated floating point numbers, which are the components of this word vector. [5 marks]

3. Create a function that maps between a word and a vector. If the word is not found in the list of vectors, try again with the lowercase version of the word. If neither of these is found, return a vector full of zeros. [10 marks]

4. Create a function that creates features for your dataset. This should return an array of shape (N, F), where N is the number of words and F is the feature dimension. This function should take in a parameter, *context*, which determines how you will define your features. [10 marks]

   (a) Context should influence how many words on either side of the current word should be included in the features for that word. For example, given a sentence "The cat is on the mat.", a context of 0 would yield a vector of $vec(cat)$ for the word cat (where *vec* maps between the word and its numerical representation as obtained in the previous step). A context of 1 would instead give $concat(vec(The), vec(cat), vec(is))$. Do not go outside sentence boundaries, and use the zero vector as padding. E.g. if the context is 2, the feature for cat would be $concat(\mathbf{0}, vec(The), vec(cat), vec(is), vec(on))$. Ensure that the zero vector has the same dimensionality as all of the others.

5. Do something similar for the label, so that you have an array of shape $(N,)$ containing the class indices of each label (these can be arbitrary but be consistent). [10 marks]

6. Create a neural network that predicts the class given the features. If you prefer you can instead use a simple, logistic regression approach without neural networks. [14 marks]

7. Train one model for each context between 0 (the features are only the specific word's vector) and 3 (the features contain the vectors from the current word and its 6 nearest neighbours); i.e., 4 models. You can choose the

hyperparameters and network architecture, but justify your choices. For example, you train for 20 epochs because the loss plateaus after that. [16 marks]

8. Compare these models based on their F1 score, as well as their accuracy both on the training set and the validation set (dev.txt). Ensure that you use the same preprocessing & feature extraction approach for the validation data as you do for the training data. [10 marks]

9. Finally, answer the following questions: Is there a notable difference between the following? Describe one potential reason per observation [20 marks]

   (a) The F1 score and accuracy?

   (b) The train and validation metrics?

   (c) The models trained with different contexts?

   (d) Your results and those obtained by Adelani et al. [2]? (hint: look at table 5 for their results)

Resources

1. https://github.com/masakhane-io/masakhane-ner
2. https://fasttext.cc/docs/en/unsupervised-tutorial.html
3. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
4. The data is located at https://drive.google.com/file/d/18dUXtHE-6Acm4n8-I5C1oPEqTE06dMfo/view?usp=sharing

References

[1] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, 2017. [Online]. Available: https://doi.org/10.1162/tacl_a_00051

[2] D. I. Adelani, J. Z. Abbott, G. Neubig, D. D'souza, J. Kreutzer, C. Lignos, C. Palen-Michel, H. Buzaaba, S. Rijhwani, S. Ruder, S. Mayhew, I. A. Azime, S. H. Muhammad, C. C. Emezue, J. Nakatumba-Nabende, P. Ogayo, A. Anuoluwapo, C. Gitau, D. Mbaye, J. Alabi, S. M. Yimam, T. Gwadabe, I. Ezeani, R. A. Niyongabo, J. Mukiibi, V. Otiende, I. Orife, D. David, S. Ngom, T. P. Adewumi, P. Rayson, M. Adeyemi, G. Muriuki, E. Anebi, C. Chukwuneke, N. Odu, E. P. Wairagala, S. Oyerinde, C. Siro, T. S. Bateesa, T. Oloyede, Y. Wambui, V. Akinode, D. Nabagereka, M. Katusiime, A. Awokoya, M. Mboup, D. Gebreyohannes, H. Tilaye, K. Nwaike, D. Wolde, A. Faye, B. Sibanda, O. Ahia, B. F. P. Dossou, K. Ogueji, T. I. Diop, A. Diallo, A. Akinfaderin, T. Marengereke, and S. Osei, "Masakhaner: Named entity recognition for african languages," *Trans. Assoc. Comput. Linguistics*, vol. 9, pp. 1116–1131, 2021. [Online]. Available: https://doi.org/10.1162/tacl_a_00416