NLP Project Report:

Advanced Text Classification

Omar Hawash

Date: 14.04.2024

NLP Course - AN-Najah National University

Supervisor: Dr.Hamed Abdelhaq

INTRODUCTION

Applying and building different text classification models to categorize documents into 91 classes using preprocessing, various encoding methods. Models will be evaluated with the macro-averaged F1-score. The report covers methodology, performance comparison, and probably suggests more enhancements for better results.

HYPOTHESIS

Dataset: news documents each having body text and class. (with 91 different classes)

<u>Training data</u>: 11413 samples. <u>Testing data</u>: 4024 samples.

A. Naive Bayes from scratch

B. Scikit-learn Naive bayes:

C. Word embedding models:

Used Libraries

Text preprocessing:

- `**Stanza**` (corenlp previously) for tokenization and lemmatization

- `**Spacy**` more general faster tokenization library

- `**NLTK**` for stopwords

Pretrained Models:

- `**scikit-learn**` for vectorization and models

- `**gensim**` for word embedding models

Data Representation:

- `**pandas**` for data manipulation

MATERIALS

1. Naive bayes theoretical equation
2. Sk-learn pre-trained models
3. Gensim pre-trained models

PROCEDURE

[...image here...]

Models Accuracy

A. Naive Bayes from scratch

Starting off, applying the main form of the naive bayes classification method equation.

1. **Naive Bayes_01:** Remove new lines, chars less than 1 length, and stop words also using spacy for tokenization and lemmatization. (vocab = 31K)
2. **Naive Bayes_02:** Everything in previous one, with lowercase and removing special characters. (vocab = 25K)
3. **Naive Bayes_03:** replacing spacy with stanza nlp for the tokenization step.

Model	Mean average	F1-Score (Macro-Averaged)
Naive Bayes_01	**50.67%**	**3.57%**
Naive Bayes_02	**48.56%**	**3.63%**
Naive Bayes_03	**49.65%**	**3.48%**

B. Scikit-learn Naive bayes:

Using pre-trained models from Scikit-learn, including Multinomial NB.

used parameters: `max_df=0.05, min_df=0.005, max_features=1000` **(for both trials)**

1. **SKLearn Naive Bayes Count-vector:** multinomial naive bayes implementation using count vectorizer.
2. **SKLearn Naive Bayes TF-IDF:** multinomial naive bayes implementation using tf-idf vectorizer.

Model	Mean average	F1-Score (Macro-Averaged)
sk_learn count vec.	**68.29%**	**30.73%**
Sk_learn tf-idf vec.	**66.4%**	**13.73%**

C. Word embedding models:

Applying logistic regression using 3 different embedding pre-trained models.

1. **Glove:** using **glove-wiki-gigaword-300** with 400K vocab, ~ 0.37GB
2. **Word2Vec:** using **word2vec-google-news-300** with 3M vocab, ~1.66GB
3. **FastText:** using **fasttext-wiki-news-subwords-300** with 1M vocab, ~0.96GB
4. **Glove_02:** `max_iter=200, regularization set at: C=1000.0`
5. **Glove_03:** `max_iter=1000,  regularization set at: C=1000.0, class_weight='balanced'`

Model	Mean average	F1-Score (Macro-Averaged)
Glove	**72.79%**	**26.67%**
Word2Vec	**69.63%**	**15.96%**
FastText	**59.05%**	**5.03%**
Glove_02	**72.71%**	**35.7%**
Glove_03	**71.4%**	**37.73%**

Also I've tried changing the solver function for convergence between ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga']. The huge benefit was the default one was way faster than others. And still The results were almost identical or worse.

D. SVM & random Forest

Applying 2 different models using glove word embeddings.

1. **SVM:** `kernel='rbf', C=1000.0, class_weight='balanced'`
2. **Random Forest:** `n_estimators=70, max_depth=100, criterion='entropy'`

Model	Mean average	F1-Score (Macro-Averaged)
SVM	**72.39%**	**36.78%**
Random Forest	**66.2%**	**22.66%**

RESULTS

First up, our Naive Bayes model from scratch, each with different preprocessing, gave accuracies around 50% and F1-Scores near 3.5%. Despite different approaches, like different tokenization tools and text preprocessing methods, their performance stayed almost the same.

With Scikit-learn Naive Bayes models, where the CountVectorizer model gave 68.29% accuracy and 30.73% F1-Score, significantly outperforming its TF-IDF in f-score, and a small change in accuracy.

The best ones yet, Word Embedding Models, with Glove, Word2Vec, and FastText. Glove was the highest one with scores. Peaking at a 72.79% mean average and an F1-Score of 37.73% after fine-tuning iterations and regularization.

Interestingly, changing the solver of logistic regression had a negligible effect, so the choice of algorithm does not change results much.