ANJISHNU MUKHERJEE
G01373990
amukher6@gmu.edu
dataminer username - noobmaster

Homework Assignment 1
Data Mining (CS 584), Fall 2022

2022-09-18

1 Abstract

The assignment consists of 5 distinct parts :

- Loading and Preprocessing the data
- Vectorising the cleaned data
- Reducing the feature dimensionality
- Developing a KNN classifier from scratch
- "Lazy"-learning with the KNN on training data, with cross-validation on a held-out set of validation data

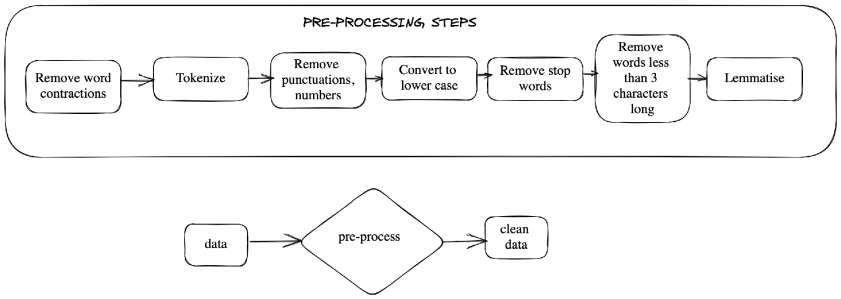
Once all the above steps are complete, we need to use the best set of parameters as obtained from the training set, to predict class labels on our test set. The resulting predictions file is uploaded to the dataminer website.

2 Methods

Since the dataset is too large, I load a small fraction of the dataset when developing my solution, and only load the full dataset when doing ablations.

2.1 Preprocessing

I follow a simple 7 step process for cleaning the text data :



- Remove word contractions (eg. you're → you are)
- Tokenization using NLTK $word\_tokenize$ which uses a combination of Treebank Word tokenizer and Punkt Sentence tokenizer for english language. I also experimented with a BERT based tokenization approach to further improve my accuracy on the test set, but ran into memory optimization issues and thus I am leaving it out of the submission and report.
- Punctuations, numbers and other non-alphabet characters are removed
- The words are converted to lower-case
- English stop words from the NLTK corpus are removed
- Words with less than 3 characters are also removed
- Finally, I lemmatize using the NLTK WordNet lemmatizer. Since lemmatization considers context as well before converting a word to its base form, this approach seemed to work better for me than the default approach of using the Porter Stemmer.

2.2 Vectorisation

I experimented with the Count Vectorizer and TF-IDF Vectorizer, both from scikit-learn. Overall, tfidf was better. This is simply due to the nature of tf-idf, as it considers the overall document weightage of every term instead of counting just the frequency, which allows it to penalize more frequently appearing words.

For tfidf, when building the vocabulary, I strictly ignore words having frequencies less than 3 and more than 95% to remove some outliers in the frequency distribution.I also consider only the top 1000 features ordered by term frequency across the corpus when building the vocabulary. The choice of these three numbers is arbitrary, except for 1000, which was chosen based on the ablation studies I ran on the (SVD+TF-IDF) combination as discussed later.

TF-IDF output has a non-zero percentage of only 3.13%, meaning that it is a very sparse matrix. To remove the irrelevant zero features, we would next do dimension reduction.

2.3 Dimensionality reduction

For getting a low-dimensional "semantic" space from the sparse TF-IDF output, I chose to use Truncated SVD, which is an improvement over regular PCA in the following aspects :

- Truncated SVD of sklearn is optimized for sparse matrices, whereas regular PCA is not.
- Truncated SVD implementes a variant of SVD, that computes only the n largest values. n is a user-provided value which I set to 750 based on the results from my experiments.
- Truncated SVD doesn't need its input matrix to be centered. And turning on sublinear scaling during calculations compensates for some of the wrong assumptions the algorithm makes.

The final SVD output is 750, which is a low enough dimensional space, considering that the input has 18,000 examples. Post-SVD, my embeddings for the dataset have a 99.83% non-zero values, which is a big change from the 3.13% from the TF-IDF output.

2.4 KNN

I built a KNN classifier from scratch. Some important notes about the implementation :

- I choose the k nearest neighbours based on numpy argsort. But argsort uses quicksort by default, which is not a stable sorting method. I modify this to use timsort, which is stable. This ensures that if two neighbouring points have the same distances, the one whose index appears first in the distance matrix would be chosen. This is important because, depending on which points are chosen at this stage, the downstream performance on the test set may be affected.
- For choosing a class label based on the k nearest neighbors, I chose a majority voting scheme. This can be implemented in many ways, but I implemented it by calculating mode using the scipy stats package to take advantage of the sparse representation of my embeddings.
- I compared the performances of a weighted and a non-weighted implementation. The weighted one performed better. For weighted KNN, the distances of the k nearest neighbours are used to weigh the majority voting process for choosing the label assigned to the point.
- For distance metric, I experimented with both "cosine" and "euclidean" distances. But, as we will see next in the results section, "cosine" always performs better in terms of accuracy/f1 score on train data.

2.5 Training/Testing Cross-validation

- I split the 18,000 examples of training data into a 80:20 split, where I will hold out 20 percent of the data for validation.
- With this train-validation separation, I would run experiments with two types of cross fold validation (kfold and stratified kfold). But, for the optimal set of parameters and hyperparameters, both of these methods would result in a similar score for the KNN. I ensured that all folds have an equal number of training samples and experimented across fold sizes 3, 5 and 10 without noticing any significant changes.

- Apart from the distance metrics for KNN, I also experimented with the other parameter for this classifier which is basically the value of k (or the number of nearest neighbours). When $k \approx 450$, both accuracy and f1-score on the validation set reaches an optimal value of 0.84 and stays nearly constant beyond that. I checeked this by running experiments on k from 1 upto 2000. Also, I only chose odd values of k, because this is a two-class problem, so choosing an odd number of neighbors would ensure that my majority voting scheme won't have a tie in any of the local neighborhoods, unless the distances of some of the k neighbors are same (and I have already implemented argsort using timsort to take care of that case).
- I create a grid search loop to iterate over both the parameters of the KNN classifier, i.e., all values of k and the two distance metrics ("euclidean" and "cosine") and ran multiple ablation studies to determine the variations of these parameters and their effects on validation accuracy / f1-score.
- For each optimal grid search completion, I would get a value of k for which

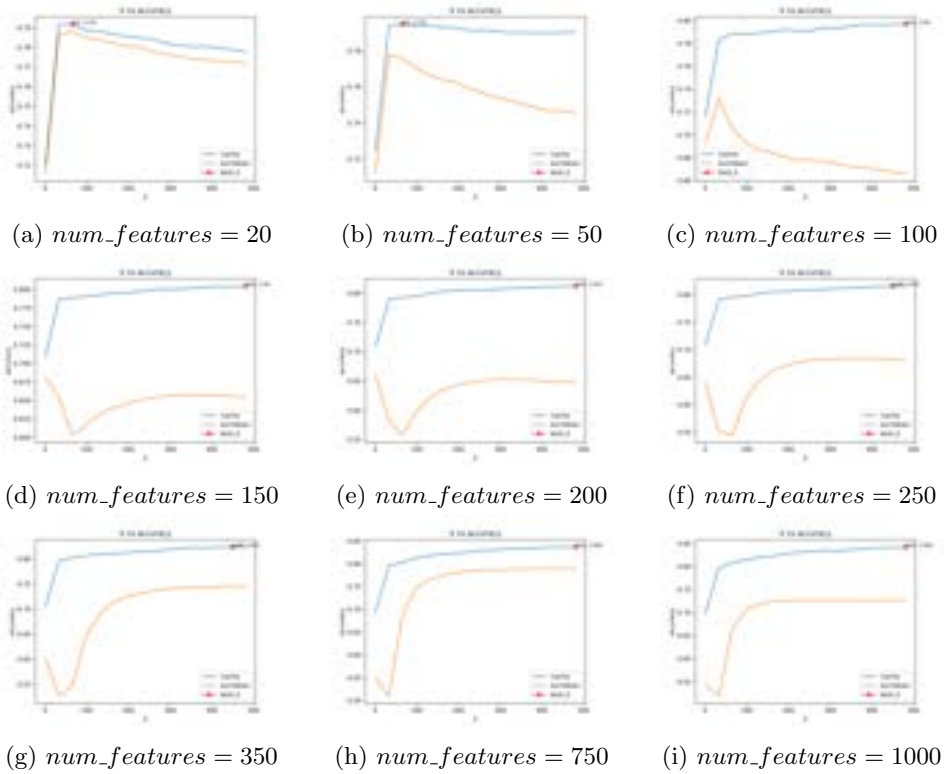
3 Results

3.1 Vectorisation and Dimensionality reduction

- TF-IDF Embeddings shape : $(18000, 1000)$
- Vocab length : 1000
- TF-IDF non zero percentage : 3.13%
- TF-IDF+Truncated SVD Embeddings shape : $(18000, 750)$
- SVD non zero percentage : 99.83%

3.2 Ablation studies

For number of features less than 100, the graphs are not stable, in the sense that accuracy on validation drops after an initial high. But on increasing num features from 100 onwards upto 1000, the graphs kind of look similar and the accuracy remains nearly constant at 0.84.



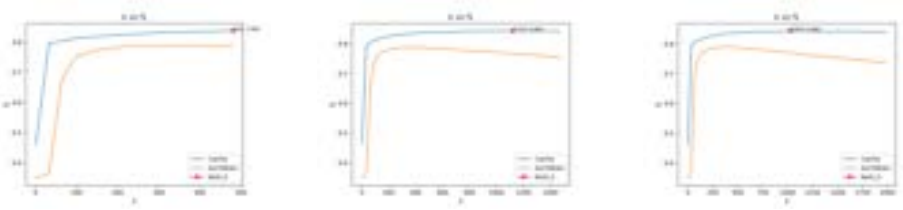
(a) $num\_features = 20$ (b) $num\_features = 50$ (c) $num\_features = 100$

(d) $num\_features = 150$ (e) $num\_features = 200$ (f) $num\_features = 250$

(g) $num\_features = 350$ (h) $num\_features = 750$ (i) $num\_features = 1000$

Figure 1: Ablation studies for number of features

For k, as can be seen from the 3 figures, beyond 450, change in k does not really affect f1 score (or accuracy, which has the same value as f1 for all the three figures) for the KNN classifier.



(a) $k \in (1, 500)$ (b) $k \in (1, 1000)$ (c) $k \in (1, 2000)$

Figure 2: Ablation studies on k

4 Conclusion

After running all these experiments, I got 0.84 highest score on validation set and 0.82 highest score on test data. One particular set of parameters which gives this is $k = 1025$ with cosine distance. Obviously as stated and proved above, there can be multiple such values which will give same result.