Homework 3: Classification and Vector Semantics

Tristan Maidment (TDM51@pitt.edu)

Code Design

This project was designed around the classes provided by the Scikit-Learn (*sklearn*) framework. This allows to use off-the-shelf algorithms built into the framework, in additional to seamlessly integrating custom algorithm and text representations.

run_tests.py

This file contains the cross-validation testing function, *test_cv*, which provides an interface for testing a combination of text representation and algorithm. Testing is done via 5-fold cross-validation, to thoroughly test the algorithms' performances. The cross-validation data is split via the *article_id*, such that all comments for an article are grouped together. This prevents data leakage, ensuring comments from the same article are not separated into different folds.

The scoring metrics from cross-validation are *accuracy* for classification tasks, and *mean squared error* for regression tasks. While other metrics, such as AUC for classification were considered, the negligible class imbalance made accuracy a more interpretable metric. All scores from the cross validation, including the average score, as well as the time required to fit and predict the algorithm are outputted. *run_tests.py* is set up to run the full suite of tests used for this report, which will run automatically if called.

dataset.py

The dataset class is designed to handle the variable types of ground truth data types and preprocess the comment text. The class also requires that a specific type of text representation be specified, a *sklearn* TransformerMixin object, to be specific, allowing for a variety of different text representations to be used.

The decided text preprocessing was simply to cast each sentence to lowercase. Additional preprocessing methods were explored, including removing all special characters, with an exception of punctuation, and to add additional spaces between words and characters to aid the bag of word model. However, these preprocessing types significantly hurt the performance of the neural-network based embeddings and provided very little improvement to the traditional word representations. However, the code for these different preprocessing is left in the class and can be tested by manually setting the input parameter.

majorityvote.py

This class contains the logic for the majority-vote classifier, as required in step 2. This classifier finds the most commonly predicted class during training, and only returns that class during prediction.

latentsemanticanalysis.py

This class contains the code for the dense vector semantic representation of choice, in this case an SVD of the sparse *Bag of Words* (BoW) count matrix, formally known as *Latent Semantic*

Analysis (LDA). LDA was an obvious choice for the dense vector semantic representation, since it directly builds upon the sparse representation being tested.

neural.py
This class provides access to the pre-trained neural embeddings from the popular *spaCy* NLP package.

Experiments

Step 2: To look at initial differences in performance between the majority vote classifier and the logistic regression classifier, we look at the simple BoW representation. Based on the cross-validated results, the logistic regression classifier performs consistently better than the majority vote classifier, demonstrating strong discriminative performance from the BoW feature representation.

BoW Full Results

	Average	Fold 0	Fold 1	Fold 2	Fold 3	Fold 4
Majority-Vote	0.5312	0.5311	0.5311	0.5311	0.5311	0.5311
Logistic Regression	0.8533	0.8086	0.8756	0.866	0.8469	0.8696

Step 3: To improve the logistic regression baseline classifier, we consider an additional three vector semantic representations. For the purpose of this report, I have only included the average accuracies. To see the full results, please run the testing code, or reference the raw console output section at the bottom.

1. **Sparse** – term frequence-inverse document frequence (document normalized BoW)
2. **Dense** – latent semantic analysis (SVD on BoW count matrix)
3. **Neural*** - *spaCy* neural embeddings

** Both "Dense" and "Neural" are dense vector representations, but I have labeled them separately for the readers convenience.*

Average Classification Accuracies

	BoW	Sparse	Dense	Neural
Majority-Vote	0.5312	0.5312	0.5312	0.5312
Logistic Regression	0.8533	0.8332	0.8389	0.7258

Surprisingly, we find that the BoW outperforms all other methods in our testing environment. However, the sparse and dense representations are not far behind the BoW representation for logistic regression. The neural embedding provided by spa*C*y does provide great discriminative performance in this task.

Step 4: The expert annotations in the SFU corpus contain not only classification labels but contains a column for their confidence level of whether or not the comment is constructive. However, the distribution of this label is continuous, between the range 0 – 1. This provides an interesting opportunity to perform regression, instead of classification, to predict this confidence level. I hypothesize that the BoW embeddings will perform the best, as was found in our initial findings.

Since the design of our experimental framework supports this task without modification, we can simply drop in our regression algorithms of choice. For the purpose of this experiment, we are testing a simple Linear Regression method, to Support Vector Regression (a regression extension of a Support Vector Machine).

Lastly, since we are able to predict any continuous set of labels, we can also predict the toxicity level confidence column, where I expect to similar results to the constructive confidence test.

Average Constructive Confidence MSE

	BoW	Sparse	Dense	Neural
Linear Regression	1.9997	0.2906	3.7430	0.8176
Support Vector Regression	0.0579	0.0233	0.0406	0.0762

Average Toxicity Confidence MSE

	BoW	Sparse	Dense	Neural
Linear Regression	3.4717	0.4415	3.9247	0.8836
Support Vector Regression	0.0544	0.0229	0.0792	0.0825

As we can see, the BoW representation did not perform the best, as I had hypothesized before running these experiments. Across the board the *Sparse* representation performed the best at this task, followed by the previously poor neural *spaCy* neural embedding and dense embeddings.

To get a fuller understanding of the different representations, it is appropriate to compare the amount of data required to an entry of each representation, and the effect it has on computational time. In a production environment, this is a particularly important set of traits that is sometimes overlooked in academic works.

Components per Entry

	BoW	Sparse	Dense	Neural
Number of Components	7982	7982	300	300

Average Constructive Confidence Compute Time (seconds)

	BoW	Sparse	Dense	Neural
Linear Regression	0.1499	0.0221	0.0143	0.0204
Support Vector Regression	0.2395	0.2659	0.2246	0.2493

Average Toxicity Confidence Compute Time (seconds)

	BoW	Sparse	Dense	Neural
Linear Regression	0.1509	0.0220	0.0132	0.0111
Support Vector Regression	0.2328	0.2447	0.2347	0.2308

For the linear regression implementation, the Dense and Neural representations, with 300 components per entry, take a significantly smaller amount of time to fit and predict than the BoW and Sparse representations. Furthermore, the 7982 components required to maintain the BoW and Sparse representations takes 26x more memory than the 300 components required in Dense and Neural representations.

Conclusion

As a conclusion, each of the different representations have benefits in different use cases, and for different tasks. While the simple BoW representation works well in the simple classification task, it is quickly outshined by the more advanced representations in the more complex regression tasks. Furthermore, while the denser representations do not provide as much discriminative performance, they perform quite similarly with a significant decrease in resources.

Testing Raw Console Output

```
=== TEXT CLASSIFICATION ===

--- BoW Representation Performance ---
Average Score: 0.5312 | Fold Scores: [0.5311, 0.5311, 0.5311, 0.5311, 0.5314] | Average Calculation Time 0.0186
Average Score: 0.8533 | Fold Scores: [0.8086, 0.8756, 0.866, 0.8469, 0.8696] | Average Calculation Time 0.0744

--- Sparse Representation Performance ---
Average Score: 0.5312 | Fold Scores: [0.5311, 0.5311, 0.5311, 0.5311, 0.5314] | Average Calculation Time 0.0167
Average Score: 0.8332 | Fold Scores: [0.823, 0.8469, 0.8086, 0.8469, 0.8406] | Average Calculation Time 0.0234

--- Dense Representation Performance ---
Average Score: 0.5312 | Fold Scores: [0.5311, 0.5311, 0.5311, 0.5311, 0.5314] | Average Calculation Time 0.0007
Average Score: 0.8389 | Fold Scores: [0.8038, 0.8565, 0.8565, 0.8325, 0.8454] | Average Calculation Time 0.0222

--- Neural Representation Performance ---
Average Score: 0.5312 | Fold Scores: [0.5311, 0.5311, 0.5311, 0.5311, 0.5314] | Average Calculation Time 0.0007
Average Score: 0.7258 | Fold Scores: [0.7368, 0.7512, 0.6938, 0.7129, 0.7343] | Average Calculation Time 0.0180
=== TEXT REGRESSION ===
Using the 'is_constructive:confidence' column for regression values

--- BoW Representation Performance ---
Average Score: 1.9997 | Fold Scores: [1.5109, 1.3955, 1.6458, 2.4161, 3.0303] | Average Calculation Time 0.1499
Average Score: 0.0579 | Fold Scores: [0.1008, 0.0502, 0.0417, 0.0462, 0.0504] | Average Calculation Time 0.2395

--- Sparse Representation Performance ---
Average Score: 0.2906 | Fold Scores: [0.3763, 0.3019, 0.1912, 0.2471, 0.3363] | Average Calculation Time 0.0221
Average Score: 0.0233 | Fold Scores: [0.0016, 0.0342, 0.0309, 0.0228, 0.0301] | Average Calculation Time 0.2659

--- Dense Representation Performance ---
Average Score: 3.7430 | Fold Scores: [6.5015, 0.7561, 2.8301, 4.4693, 4.1577] | Average Calculation Time 0.0143
Average Score: 0.0406 | Fold Scores: [0.0903, 0.0743, 0.0089, 0.005, 0.0346] | Average Calculation Time 0.2246

--- Neural Representation Performance ---
Average Score: 0.8176 | Fold Scores: [0.7632, 0.8813, 1.2462, 0.7038, 0.4932] | Average Calculation Time 0.0204
Average Score: 0.0762 | Fold Scores: [0.0369, 0.1097, 0.0937, 0.0547, 0.0861] | Average Calculation Time 0.2493
=== TEXT REGRESSION ===
Using the 'toxicity_level:confidence' column for regression values

--- BoW Representation Performance ---
Average Score: 3.4717 | Fold Scores: [4.7609, 1.6263, 1.8782, 2.9838, 6.1092] | Average Calculation Time 0.1509
Average Score: 0.0544 | Fold Scores: [0.0692, 0.074, 0.0347, 0.0439, 0.0504] | Average Calculation Time 0.2328

--- Sparse Representation Performance ---
Average Score: 0.4415 | Fold Scores: [0.7189, 0.3391, 0.3145, 0.3778, 0.4569] | Average Calculation Time 0.0220
Average Score: 0.0229 | Fold Scores: [0.0249, 0.0299, 0.0176, 0.0149, 0.0272] | Average Calculation Time 0.2447

--- Dense Representation Performance ---
Average Score: 3.9247 | Fold Scores: [6.453, 2.4713, 2.854, 2.6707, 5.1745] | Average Calculation Time 0.0132
Average Score: 0.0792 | Fold Scores: [0.0728, 0.087, 0.0845, 0.0349, 0.1169] | Average Calculation Time 0.2347

--- Neural Representation Performance ---
```

Average Score: 0.8836 | Fold Scores: [1.3374, 0.6531, 0.8878, 0.662, 0.8776] | Average Calculation Time 0.0111
Average Score: 0.0825 | Fold Scores: [0.1012, 0.1088, 0.0003, 0.0356, 0.1668] | Average Calculation Time 0.2308
Done.