**Project 3---An exploration of the impact of vectorization on classification**

**Experiment Dataset**: Amazon Customer Reviews dataset

**Target**: binary outcome positive=1 vs negative=0 (sentiment)

**Experiment Goals**:

* does the type of vectorization selected have a significant impact on the performance of the classification models?
* find the best balance between preprocessing, vectorization and algorithms…

In fact, not only the vectorization matters, but also the model is crucial.

**Experiment Method**: Control variable Method.

Keep the same vectorization method, change models, keep the same model, compare different vectorization methods.

**Bag of words (BoW) family**:

**Method1---Bag of Words Vectorization with Binary Weighting (CountVectorizer):**

This method only counts whether a word occurs or not, which is simple to compute and reduces the effect of word frequency, but it has many drawbacks,

* ignores the word frequency information,
* don’t compute word importance (couldn’t distinguish important words),
* generates high-dimensional sparse matrices,
* loses the order of words,
* lacks context.

**Method2---Bag of words vectorization with terms frequency (TF) weighing:**

This method counts word frequencies, considers word frequency information, and is simple to calculate, but again has the following drawbacks:

* couldn’t distinguish important words,
* generates high-dimensional sparse matrices,
* loses the order of words,
* lacks context.

**Method3--- Bag of words with TFIDF weighing (Our starting point):**

This method computes both word frequency and inverse document frequency, these 2 could help distinguish important words from common words. Since this method is obviously better than those 2 above, we pick this one as our starting point. Drawbacks as below:

* generates high-dimensional sparse matrices,
* loses the order of words,
* lacks context.

**Method3---Performance on Random Forest (by default without hyperparameter tuning):**

A screenshot of a computer

AI-generated content may be incorrect. A graph with a line

AI-generated content may be incorrect.

The accuracy is 0.86, good, and the ROC-AUC score is 0.93, outstanding.

A screenshot of a computer program

AI-generated content may be incorrect.

Check with baseline model---Bing Liu lexicon, our result with RF is good.

**Method3---Performance on XGBoost (By default without hyperparameter tuning):**

A screenshot of a computer

AI-generated content may be incorrect. A graph with a line

AI-generated content may be incorrect.

The results are good, but worse than those of Random Forest.

**Method3---Performance on LightGBM (By default without hyperparameter tuning):**

A screenshot of a computer

AI-generated content may be incorrect. A graph with a line

AI-generated content may be incorrect.

The results are better than those of XGBoost, but still worse than those of Random Forest.

**Method3---Performance on Logistic Regression (By default without hyperparameter tuning):**

A screenshot of a computer

AI-generated content may be incorrect. A graph with a line

AI-generated content may be incorrect.

The results of Logistic Regression are better than those of Random Forest, XGBoost, LightGBM.

**Method3---Performance on Naïve Bayes (By default without hyperparameter tuning):**

A screenshot of a computer code

AI-generated content may be incorrect. A graph with a line

AI-generated content may be incorrect.

**Method3---Performance on LinearSVC (By default without hyperparameter tuning):**

A screenshot of a computer

AI-generated content may be incorrect.

Overall, **with TFIDF vectorization**, **without hyperparameter tuning**, we could achieve **the best results with Logistic Regression, accuracy is 0.87, ROC-AUC is 0.94**.

TFIDF can generate **high-dimensional sparse matrices**, the text becomes **linearly separable** in high-dimensional sparse space, so the logistic regression has the best performance, and comes with L2 regularization to prevent overfitting.

——————————————————————————————————

**Method4--- Vectorization using Topic Modeling with Non-Negative Matrix Factorization (NMF):**

This method is mainly used for dimensionality reduction and topic modeling, which could also be used for vectorization. It needs non-negative data, so it’s normally combined with TFIDF. Advantages:

* Reduced dimensionality, avoid high-dimensional sparse matrices, use low-dimensional topic vector matrices,
* High explainability.

Disadvantages:

* Cannot interpret relationships between words,
* Still can’t deal with word order and context,
* Sensitive to number of topics.

**Method4---Performance on 6 models (By default without hyperparameter tuning):**

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A graph of a positive rate

AI-generated content may be incorrect.

Checking the results, we could find **that with NMF**, **the performances are similar on Random Forest, XGBoost, and LightGBM**. **XGBoost and LightGBM are slightly better than Random Forest. The rest are worse. Best one: LGBM, accuracy is 0.85,ROC-AUC is 0.93.**

The results make sense since **NMF reduces the dimensionality**, text isn’t that linearly separable, some models work worse such as Logistic regression, but they are still ok to use, the performances are good.

——————————————————————————————————

**Method5---Vectorization using Topic Modeling with Latent Semantic Analysis (LSA):**

This method is called TruncatedSVD which is also mainly used for dimension reduction and topic modeling. It’s similar to NMF but could work for negative values.

Advantages:

* Reduced dimensionality, avoid high-dimensional sparse matrices, use low-dimensional topic vector matrices,

Disadvantages:

* Cannot interpret relationships between words,
* Still can’t deal with word order and context,
* Sensitive to number of topics.

**Method5---Performance on 5 models (By default without hyperparameter tuning):**

Here because we have negative values, we don’t consider Naïve Bayes model.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A graph with a line and numbers

AI-generated content may be incorrect.

**With LSA, Best models: XGBoost and Random Forest.**

**Best score: accuracy is 0.84, ROC-AUC is 0.92.**

——————————————————————————————————

**Method6---** **Vectorization using Topic Modeling with Latent Dirichlet Allocation (LDA):**

LDA is one method based on Bayes inference used for dimension reduction and topic modeling. It’s combined with CountVectorizer.

Advantages:

* Reduced dimensionality, avoid high-dimensional sparse matrices, use low-dimensional topic vector matrices,
* Suitable for long text,
* Good explainability.

Disadvantages:

* Cannot interpret relationships between words,
* Still can’t deal with word order and context,
* Sensitive to number of topics,
* Not stable on short text.

**Method6---Performance on 6 models (By default without hyperparameter tuning):**

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A graph with a line and numbers

AI-generated content may be incorrect.

**With LDA, overall performances drop.**

**The best model: LGBM, accuracy is 0.78, ROC-AUC is 0.86.**

Reason: The output of LDA is **Topic probability distribution**, which is more difficult for models to distinguish.

——————————————————————————————————

**Method7---** **Vectorization with Word2Vec embeddings:**

Word2Vec could convert words into word vectors, preserving semantic information. Similar words are close in high dimensional space. 2 main algorithms: CBOW - use context to predict target, Skip-gram – use target to predict context.

Advantages:

* Keep semantic information,
* Suitable for deep learning,
* Could generate text, for example, could be used as chatbot.

Disadvantages:

* Couldn’t deal with unknown words,
* Need large data to train.

To use word2vec, we need tokenization first, while we don’t need this when using all the above vectorization methods.

**Method7---Performance on 5 models (By default without hyperparameter tuning):**

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A graph of a function

AI-generated content may be incorrect.

**With Word2Vec, overall performances are all good.**

**The best model: Random Forest, accuracy is 0.84, ROC-AUC is 0.92.**

Reason: Word2Vec outputs **nonlinearly separable dense vectors**. Random Forest could process well and is more robust to noises (zero vectors).

——————————————————————————————————

**Method8---** **Vectorization with GloVE embeddings:**

GloVE is a word embedding method based on global co-occurrence matrices, which is semantically more advantageous.

Advantages:

* Rich semantic information using the co-occurrence matrix of the entire corpus,
* Load pre-trained models,
* Stable word vectors,
* Fast computation.

Disadvantages:

* Couldn’t deal with unknown words, same as Word2Vec,
* Couldn’t add new data, otherwise train the whole GloVE,
* Need large data to train,
* Compared with Word2Vec, Glove couldn’t generate text, only for representing.

**Method8---Performance on 6 models (By default without hyperparameter tuning):**

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A graph of a function

AI-generated content may be incorrect.

**With GloVE, overall performances are all good, but drop a little bit.**

**The best model: XGBoost, accuracy is 0.82, ROC-AUC is 0.91.**

Reason: GloVE is based on pre-trained word vectors, may not match well our data in this case, we could try 300d later.

——————————————————————————————————

**Method9---** **Vectorization with FastText embeddings:**

FastText is an advanced version of Word2Vec, by using subwords modeling it could solve the problem of OOV (out of vocabulary) words.

Advantages:

* Solve the problem of OOV words (better than Word2Vec, GloVE),
* Suitable for short text, more robust,
* Could be used in special fields (better than Word2Vec, GloVE),
* Support incremental training (better than GloVE),

Disadvantages:

* Slower than Word2Vec and GloVE,
* Nonsensible to stop words.

**Method9---Performance on 5 models (By default without hyperparameter tuning):**

A screenshot of a graph

AI-generated content may be incorrect. A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A graph with a line and numbers

AI-generated content may be incorrect.

**With FastText, overall performances are all good.**

**The best model: Random Forest, accuracy is 0.84, ROC-AUC is 0.92.**

FastText also outputs dense vectors.

**Method10---** **Vectorization with (BERT) Transformers embeddings:**

BERT is a deep learning model based on bidirectional transformers. It’s very powerful: it could process the context, OOV words, disambiguation, short & long text, and so on.

Here we use BERT transformers vectorization and combine with machine learning models.

**Method10---Performance on 5 models (By default without hyperparameter tuning):**

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A graph with a line and numbers

AI-generated content may be incorrect.

**With BERT**, we could see **all the performances are excellent**!

The best model: **Linear SVC, accuracy is 0.92, ROC-AUC is 0.97**. **Logistic regression** is almost the same, accuracy is also 0.92, ROC-AUC is 0.97. Next is **XGBoost,** also like these!

We could find that **data in BERT vector space is linearly separable.**

Until now, we haven’t conducted hyperparameter tuning, and we also haven’t used deep learning models. Next, we would use BERT vectorization and BERT classification.

**Deep Learning---BERT Transformers:**

We use a small pre-trained model: **bert-base-uncased.**

Next come common steps: tokenization, train test split, turn to tensors and dataset, dataloader, train model and evaluation.

Optimizer: the best one for transformers, **AdamW**.

Learning rate Scheduler: **get\_linear\_schedule\_with\_warmup.**

**Final performance:**

**Accuracy is 0.96, ROC-AUC is 0.99. Better than BERT Vectorization + Machine learning models, Perfect!**

**Performance Rank:**

1. **BERT Vectorization + BERT Classifier, ACC-0.96, ROCAUC-0.99.**
2. **BERT Vectorization + LinearSVC/Logistic Regression, ACC-0.92, ROCAUC-0.97.**
3. **TFIDF + Logistic Regression, ACC-0.87, ROCAUC-0.94.**
4. **NMF + LGBM, ACC-0.85, ROCAUC-0.93.**
5. **LSA + XGBoost/Random Forest, ACC-0.841, ROCAUC-0.921.**
6. **Word2Vec + Random Forest, ACC-0.841, ROCAUC-0.921.**
7. **FastText + Random Forest, ACC-0.836, ROCAUC-0.918.**
8. **GloVE + XGBoost, ACC-0.82, ROCAUC-0.91.**
9. **LDA + LGBM, ACC-0.78, ROCAUC-0.86.**

**Conclusion:**

Vectorization methods and models both matter a lot for final performance. Other steps like text normalization could also affect. Different vectorization methods could output different vector matrix forms, and some models could match well.

For example, if one vectorization method yields linearly separable data, Logistic regression and LinearSVC could work very well. Some advantages: fast and transparent! To pursue perfect performance, there is no doubt to use deep learning models. But we would lose explainability.

Nowadays, MIT researchers are exploring the explainability of neural networks and find some breaking news---**COAR method (Component Attribution via Regression)**

(Meng et al., 2024). The core logic is to estimate **each component’s effect** on the prediction. In neural networks, different parts play different roles in different tasks. **COAR decomposes the predictions** of the neural network onto these components to **find out which parts have the greatest impact on the model decisions**. It also **maps the output of each component to the final prediction** and then **uses a regression model to estimate their contribution**. These models aren’t black boxes anymore, we could use them as we want to pursue performance, especially in some specific domains where explainability is well preferred.A diagram of a cat

AI-generated content may be incorrect.

**Reference:**

1. Meng, Q., Geiping, J., Goldblum, M., Moosavi-Dezfooli, S., Goldstein, T., & Fowl, L. (2024). *Decomposing and Editing Predictions by Modeling Model Computation*. arXiv. <https://arxiv.org/abs/2404.11534>