Supervised methos - A binary classifier for sarcasm

Machine Learning - Master in Data Science - Universitat de Girona

Wilber E. Bermeo Quito

January 16, 2023

1 Introducción

The goal of this project is to develop supervised models that can answer the following questions: Can you identify sarcastic sentences? Can you distinguish between fake news and legitimate news? Finally, we will discuss which model is the best in this case and the reasons behind it.

The job assignment requires to build the following models:

- NB
- KNN
- SVM

This report is accompanied by other important files such as the notebook.ipynb where I implemented the models and the notebook.html, which is the exported version of notebook.ipynb that can be read using a browser. **I chose not to generate a pdf as there can be issues with exporting to pdf**.

2 The data

The data is from Kaggle[3] but you can find the original authors here [4, 5].

The dataset contains three columns.

- is_sarcastic: 1 if the record is sarcastic otherwise 0
- headline: the headline of the news article. It's in English.
- article_link: link to the original news article. Useful in collecting supplementary data

During the exploratory data analysis, I noticed that there were no missing values, but there were repeated sarcastic sentences that I removed. In the explanatory data analysis, I noticed that I could extract the domain from the article link and create a new feature, but this feature was highly correlated with the dependent variable. Therefore, I decided not to add it as a feature in the model, as it would make the models more accurate but also dependent on the source, which would make the models less generic and reusable. Is also important to say that the hole data-set has almost 30K of samples, which is not that bad but here we are not talking about big data. Finally the dependent variable is unbalanced, there are more samples of non-sarcastic headline that sarcastic ones.

3 Transform the text

English, like other natural languages, has its own syntactic rules such as hyphens, commas, and semicolons. These rules provide context for humans who understand the language. However, in many cases, simplifying the language by removing unnecessary ”noise” can still allow for understanding of the message. This is similar to how reducing words to their root form or using synonyms can also simplify language.

I used the Natural Language Toolkit [6] library to remove stop words and simplify sentences by transforming words to their root form.

3.1 Text to vector

Once the text is cleaned, I needed to transform the string into something that a mathematician or algorithm M.L model could understand. Searching around the Internet I found that it's possible to extract features from the text, using CountVectorizer[1] you can transform text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors.

The CountVectorizer class supports the parameters ngram_range and min_df. The min_df parameter allows you to ignore words that are less frequent than the value you configured. The ngram_range parameter allows you to configure the number of contiguous items in a sequence (n) from a given sample of text or speech that you want to encode. However, it is important to note that as the value of n increases, more context is provided to the machine, but the number of features increases exponentially.

For example, in the code below, you can see the configuration of min_df and ngram_range. The min_df value of 1 is constant throughout the project, while the ngrams_range is a hyper-parameter that is used in the feature selection approach.

```python
from sklearn.feature_extraction.text import CountVectorizer
corpus = df['clean_headline']
vectorizer = CountVectorizer(
    dtype='int32',
    analyzer='word',
    ngram_range=(1,1),
    min_df=1)
X = vectorizer.fit_transform(corpus)
```

All the models use CountVectorizer, but the KNN model also uses TfidfTransformer because when using the representation of CountVectorizer alone, the model produced poor results. Tf-Idf is a technique that weights the importance of words in a text, and it can be used as a feature representation for text data. By using Tf-Idf in addition to CountVectorizer, the KNN classifier receives a more informative representation of the text samples, which improved its performance.

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_extraction.text import TfidfTransformer

pipe = Pipeline ([
    ('preprocessor', preprocessor),
    ('tfidf', TfidfTransformer()),
    ('knn', KNeighborsClassifier(algorithm='brute', weights='uniform', metric='minkowski')) # note:
    ↪  kd_tree is not supported in sparse matrix
])
```

4 End to End Modeling

To ensure that this project is easily transferable, I will employ an end-to-end design pattern. In brief, this approach involves incorporating data transformations and modeling into a single workflow. Since I am utilizing scikit-learn, I will utilize the Pipelines and Transformers from this library.

To be able to achieve this, I created a custom TextTransformer responsible for cleaning the headlines and transforming them into vectors using the CountVectorizer transformer. I also created a main transformer pipeline that can be reused throughout all the models.

```
text_processor_pipeline = Pipeline(steps=[
            ('denoiser', TextDenoiser()),
            ('ct', ColumnTransformer(transformers=[
                ('v', CountVectorizer(
                        dtype='int32',
                        analyzer='word',
                        min_df=1), 0)
                ])
            )
  ])

# Column transformer get's ride of no transformed columns
preprocessor = ColumnTransformer(
    transformers=[('tf1', text_processor_pipeline, ['headline'])],
)
```

5 The modeling

As I observed in the Exploratory Data Analysis section, the data is unbalanced. To address this issue, as the majority of sentences are non-sarcastic, I did:

- Implement stratified sampling based on the dependent variable, to ensure that the test dataset has the same distribution as the original set.
- Implement under-sampling to decrease the number of samples in the majority class.
- Reserve 15% of the original data-set as a test set. The test set resulting from the under-sampling process will have around 25K samples, therefore the validation test set will have around 3K-4K samples.
- Utilize GridSearchCV to tune the hyper-parameters of the transformations and models simultaneously, by applying 5-fold cross-validation to the training set.
- For each trained model, I will display its corresponding confusion matrix, ROC curve, and other statistics such as training time, the score of the best model for the training test and test set.

5.1 Performance metric

I made use of F1-score as the main performance metric in all models because F1-score is a good metric to evaluate binary classification models, especially when the classes are imbalanced, it balances the precision and recall, it gives a good overview of how well the model is performing overall.

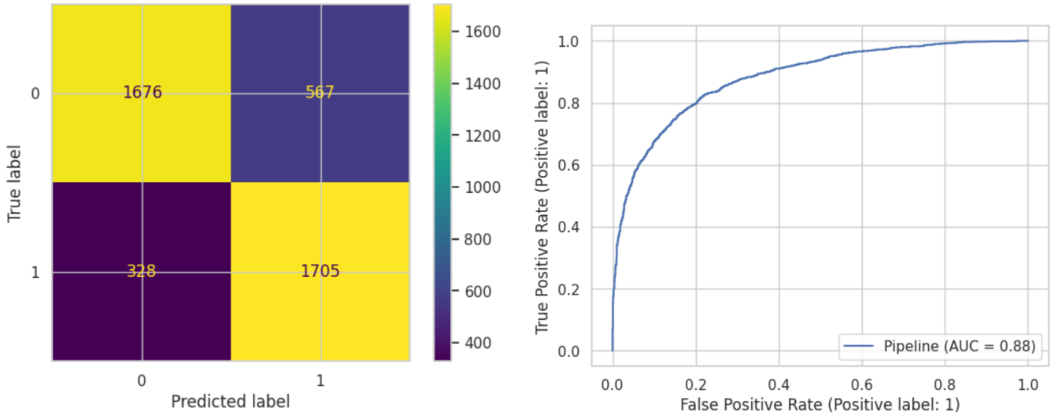
Yet I also use it along with other metrics such as accuracy, precision, recall, and a ROC curve as you can see in the notebook.

5.2 Multinomial NB

Multinomial Naive Bayes is a classification algorithm for text data. It uses Bayes theorem to calculate the probability of a text sample belonging to a certain class, given a set of features (word counts). It makes the assumption that the features (words) are independent given the class and it estimates the probability of a word belonging to a class based on its frequency in the training data. It is efficient, easy to implement and works well with sparse data, commonly used in text classification and NLP tasks.

The tuned hyparameters and the GridSearchCV configuration.

```
alphas = [0.05,0.1,0.5,1.0,2.0,4.0]
ngrams = [(1, 1), (1, 2)]
grid_params = {
  'mnb__alpha': alphas,
  'mnb__fit_prior': [True],
  'preprocessor__tf1__ct__v__ngram_range': ngrams
}
gs_mnb = GridSearchCV(
    pipe,
    grid_params,
    cv = 5,
    scoring = 'f1',
    n_jobs=-1
)
```



(a) Confusion matrix.

(b) ROC curve.

Figure 1: Best Multinomial NB model metrics.

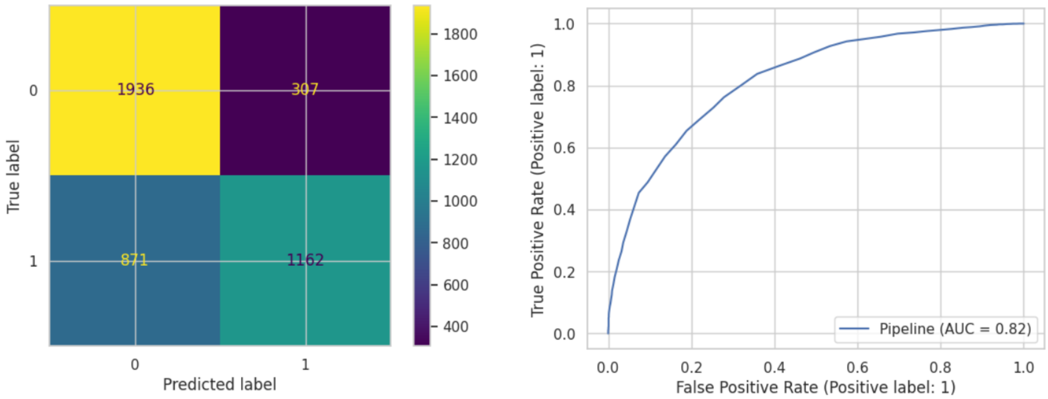
5.3 K-NN

A k-nearest neighbor (k-NN) classifier can be used to classify text data, just like it can be used to classify data in other domains. The basic idea behind k-NN is to find the k data points in the training set that are closest to a given test point, and then classify the test point based on the majority class of those k nearest neighbors.

For text classification, the process would involve converting the text into a numerical representation, such as a bag-of-words or a word embedding, and then using the k-NN algorithm to classify the text based on the similarity of its representation to the training data. This can be effective for text classification tasks with a small number of classes and a relatively large amount of labeled training data.

The tuned hyparameters and the GridSearchCV configuration.

```
ngrams = [(1, 1), (1, 2)]
grid_params = {
  'knn__n_neighbors': [15,30,45,60,75,90],
  'preprocessor__tf1__ct__v__ngram_range': ngrams,
  'knn__p': [1, 2],
  'knn__leaf_size': [10, 20]
}
gs_knn = GridSearchCV(
    pipe,
    grid_params,
    cv = 5,
    scoring = 'f1',
    n_jobs=-1
)
```



(a) Confusion matrix.

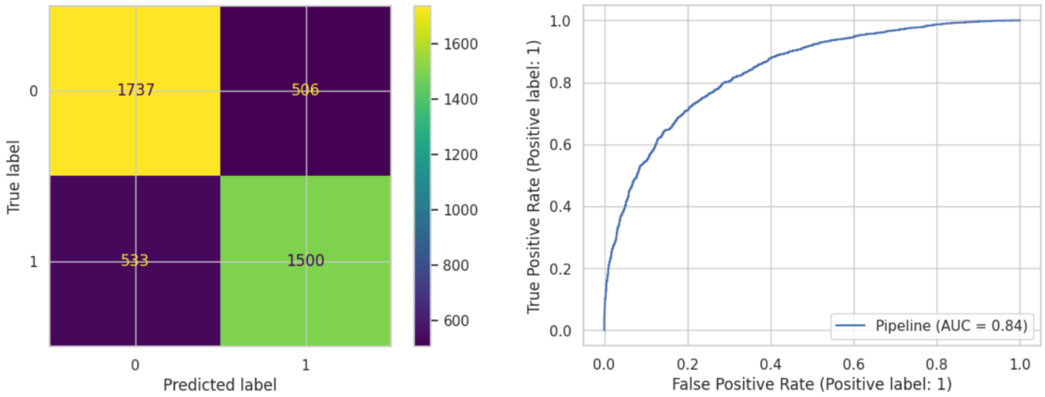
(b) ROC curve.

Figure 2: Best K-NN model metrics.

5.4 SVM

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for text classification. The basic idea is to convert the text into a numerical representation, such as a bag-of-words or a word embedding, and then use the SVM algorithm to find a decision boundary that separates the different classes of text in the training set. SVM is particularly useful when there is a clear margin of separation between the classes in the feature space, which can be the case in text classification. It tries to find the best boundary between the classes by maximizing the margin, which is the distance between the boundary and the closest training data points from each class, also known as support vectors. This makes it less sensitive to the overfitting and better generalization to unseen data. SVM can be effective for text classification tasks with a small number of classes and a relatively large amount of labeled training data, especially if the linear kernel is used. But if the data is not linearly separable, it may be necessary to use a non-linear kernel function, such as polynomial or radial basis function (RBF) kernel. In my case, RBF kernel has a better performance.

```
ngrams = [(1, 1), (1, 2)]
grid_params = {
    'svc__C': [0.1, 0.9, 9],
    'svc__kernel': ['linear', 'rbf'],
    'svc__cache_size': [ 200, 400 ],
    'preprocessor__tf1__ct__v__ngram_range': ngrams,
}
gs_svm = GridSearchCV(
    pipe,
    grid_params,
    cv = 5,
    scoring = 'f1',
    n_jobs=-1
)
```



(a) Confusion matrix.

(b) ROC curve.

Figure 3: Best SVM model metrics.

5.5 Ensemble[2]

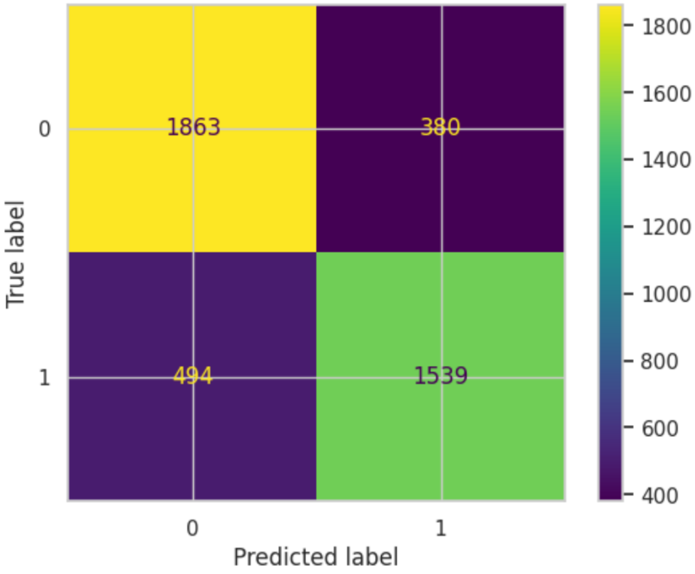
I observed that while the Multinomial Naive Bayes and SVM models I trained had high performance on the training set, they were overfitting as their accuracy decreased significantly on the test set. On the other hand, the KNN model I used had poor performance on both the training and validation sets. To solve this issue, I thought of using an ensemble approach to reduce the overfitting of the Multinomial NB and SVM models and increase the variance of the final model. Additionally, to address the high bias of the KNN model, I gave it less importance in the ensemble process.

This approach let me has a more accurate model as you will see in the last section, but of course it needs the 3 trained models before it can be used.

```
from sklearn.ensemble import VotingClassifier

estimators = []
for estimator_name, data in M.items():
    if estimator_name == 'ensemble':
        continue
    estimator = data['gsearch'].best_estimator_
    estimators.append((estimator_name, estimator))

ensemble = VotingClassifier(estimators, voting='hard', n_jobs=-1, weights=[1.25, 1, 1.25])
ensemble.fit(X_res, y_res)
```



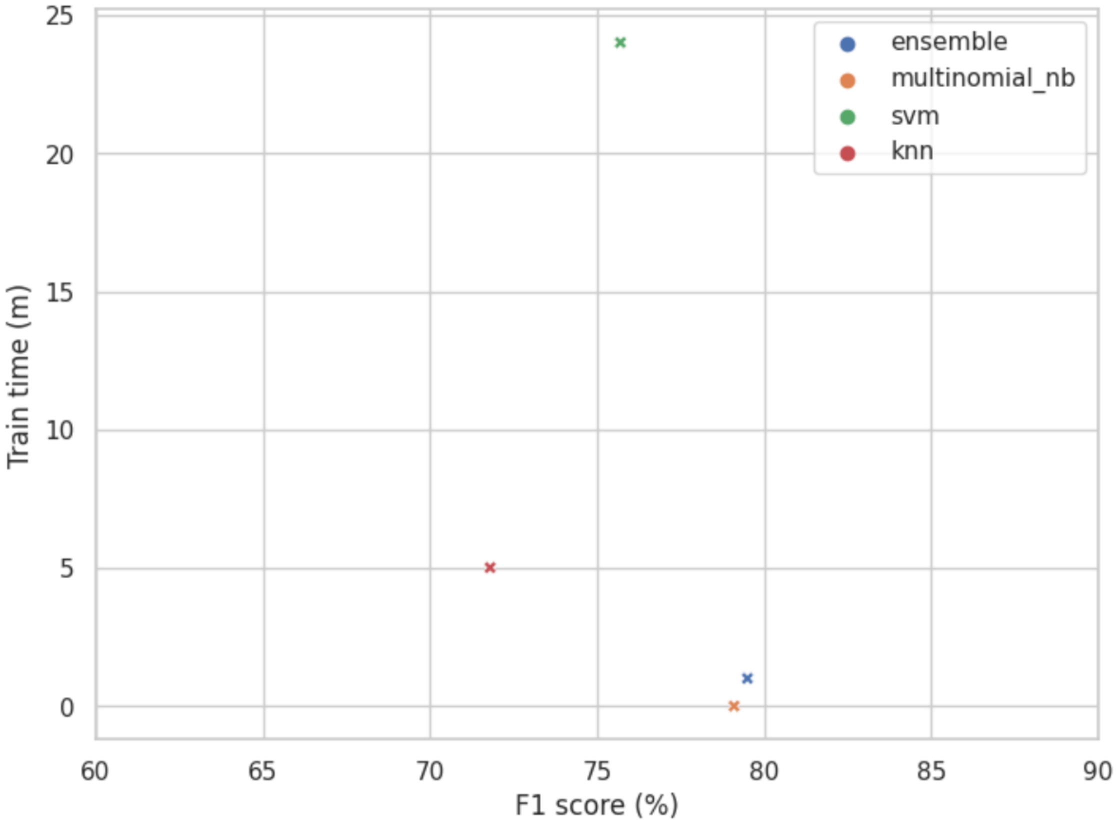
(a) Confusion matrix.

6 Metrics and conclusion

Bellow you can see the different metrics to compare the different models sorted by F1-score. Multinomial Naive Bayes not only performs better when compared to other models during testing, but also has a significantly faster training time than the other models. While other models such as KNN and SVM take almost 5 minutes and a half hour respectively, Naive Bayes' training time is less than a minute. Although SVM has better performance compared to KNN, its training time is 5 times longer. To avoid this issue, I would recommend using KNN instead of SVM despite its slightly poorer performance of 5 percentiles. Therefore, the best model for this sarcasm classification task is Multinomial Naive Bayes.

By using ensemble to join multiple tree models, I was able to improve the performance of the overall model, despite a slight decrease in accuracy in the training set. This is normal due to the bias-variance trade-off. The main metrics used to measure performance of these models were also better than those of the individual models. However, it is important to consider the additional training time required for the ensemble model (because you need the other models and itself). Despite this, it is not a significant issue in situations where time is not of the essence.

	precision_score	recall_score	f1_score	accuracy_score	train_accuracy_score	train_time
ensemble	0.796	0.796	0.795	0.796	0.946	73.32
multinomial_nb	0.795	0.791	0.791	0.791	0.97	48.89
svm	0.757	0.757	0.757	0.757	0.948	1472.83
knn	0.738	0.725	0.718	0.725	0.74	346.33



(a) F1- Score - models performance over time.

References

References

[1] CountVectorizer. Available at https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

[2] Ensemble. Available at https://towardsdatascience.com/ensemble-learning-in-sklearn-587f21246e8d.

[3] Kaggle. Available at https://www.kaggle.com/.

[4] Rishabh Misra and Prahal Arora. Sarcasm detection using hybrid neural network. *arXiv preprint arXiv:1908.07414*, 2019.

[5] Rishabh Misra and Jigyasa Grover. *Sculpting Data for ML: The first act of Machine Learning*. 01 2021.

[6] nltk. Available at https://www.nltk.org/.