Nitro NLP

Clinciu Diana
`grupa 344`

State Antonia
`grupa 344`

May 14, 2024

Abstract

This article presents our different approaches to solving a text classification challenge in the Romanian language. In this task, we must classify the text between satirical and non-satirical news. We experimented with several different models, including Logistic Regression, Random Forest, Stochastic Gradient Descent, and a Fully Connected Neural Network.

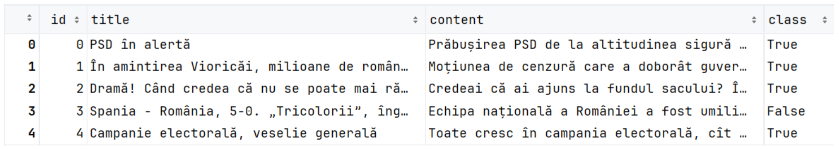
A key aspect we addressed is data processing, which involved text preprocessing tasks such as removing special characters and lemmatizing words.

Another challenge encountered was the missing data in the given data set. Having known that either the content or the title might be missing, we simply completed the missing content sections with the title sections of those entries and the text classification was made by the content section.

Results show that the models achieved accuracies between 0.8 and 0.85 with Logistic Regression, Random Forest, and Stochastic Gradient Descent, while the Fully Connected Neural Network performed the best with an accuracy of 0.88. Based on these findings, the neural network model proved to be the most effective model for this classification task.

1 Introduction

In the vast domain of natural language processing known as NLP, text classification serves as a fundamental task with applications ranging from sentiment



	id	title	content	class
0	0	PSD în alertă	Prăbușirea PSD de la altitudinea sigură …	True
1	1	În amintirea Vioricăi, milioane de român…	Moțiunea de cenzură care a doborât guver…	True
2	2	Dramă! Când credea că nu se poate mai ră…	Credeai că ai ajuns la fundul sacului? Î…	True
3	3	Spania - România, 5-0. „Tricolorii”, îng…	Echipa națională a României a fost umili…	False
4	4	Campanie electorală, veselie generală	Toate cresc în campania electorală, cît …	True

Figure 1: Training data

analysis to topic labeling. Our work is focused on the specific challenge of classifying Romanian text into satirical and non-satirical news categories, a task of increasing importance given the increased spreading of news in the digital media and online content.

Discerning between satirical and non-satirical content represents a delicate challenge, as satire genre usually employs distinct techniques to critique societal flaws and failures by amplifying them to absurd proportions. Satirical writings are characterized by its clever use of exaggeration, humor, and irony. Understanding and accurately categorizing satirical content in Romanian news articles requires an approach that appreciates the subtleties of language and cultural context.

2 Method

Initially, we addressed this challenge using common machine learning classification algorithms such as Logistic Regression and Random forest.

2.1 Dataset

The data set contains *train.csv* and *test.csv* files which consist of the training and test set. We further split the training set into 70% the training set (used to train the models) and 30% the validation set (used to check the accuracy of the models).

The distribution of the training dataset is illustrated in *Figure 2*, highlighting that approximately 60% of the data contains satirical news, with the remaining 40% representing non-satirical content.

Therefore, we examined the training dataset and have provided a screenshot of the data found in the first lines in the training set (*Figure 1*).

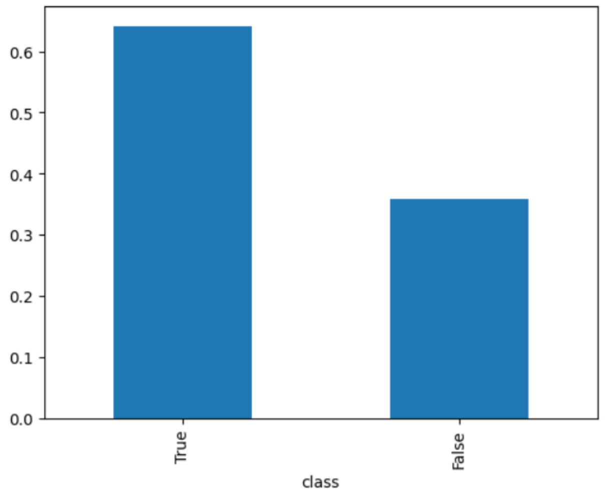


Figure 2: Distribution of the training data

2.2 Preprocessing

What preprocessing methods did you try? Why did you chose them? How did they affect the model?

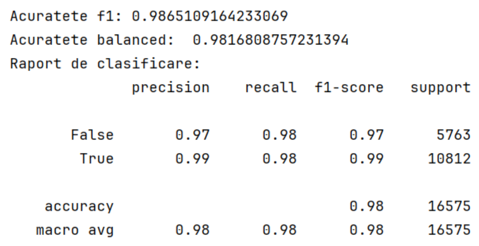
We approached the task by translating all the text to lower case, eliminating all the symbols and lemmatizing the words.

```
def process_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)

    words = word_tokenize(text)
    new_text = ' '.join(lemmatize_text(words))
    return new_text

def lemmatize_text(l_words):
    words = [lemmatizer.lemmatize(word) for word in l_words]
    return words
```

In order to preprocess text data for a machine learning task, we used *CountVectorizer()* from sklearn which converts a collection of text documents to a matrix of token counts.



```
Acuratete f1: 0.9865109164233069
Acuratete balanced:  0.9816808757231394
Raport de clasificare:
              precision    recall  f1-score   support

       False       0.97      0.98      0.97      5763
        True       0.99      0.98      0.99     10812

    accuracy                           0.98     16575
   macro avg       0.98      0.98      0.98     16575
```

Figure 3: Accuracy report for lr model

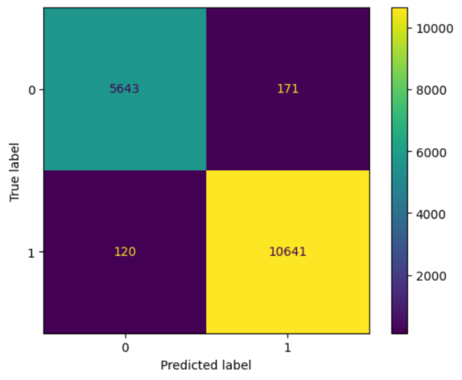


Figure 4: Confussion matrix for lr model

2.3 Models

2.3.1 Logistic Regression

The first implementation was a logistic regression (LR) model for text classification using Python's scikit-learn library.

Logistic regression is a popular classification algorithm that is particularly effective for binary classification tasks like distinguishing between satirical and non-satirical news articles.

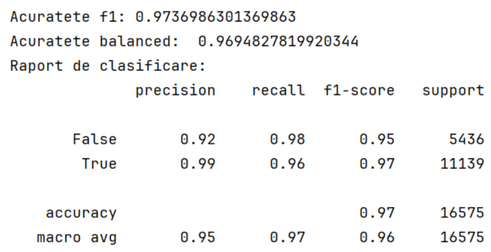
```
lr = LogisticRegression(max_iter=2000)
lr.fit(train_content, train_label)
predictions_lr = lr.predict(test_content)
```

The *max_iter* parameter specifies the maximum number of iterations for the solver algorithm to converge and was set it to 2000.

During training, the logistic regression model learns to map the textual features to the correct class labels based on the provided training examples.

After training the logistic regression model, we use it to make predictions on the test and validation data. The predict() method applies the trained logistic regression model to the test data and generates predicted labels based on the learned patterns from the training phase.

The accuracy report on the validation data we have obtained is visible in *Figure 3* alongside the confussion matrix *Figure 4*, while on the test data we obtained 0.855 balanced accuracy on Kaggle (above the baseline).



```
Acuratete f1: 0.9736986301369863
Acuratete balanced:  0.9694827819920344
Raport de clasificare:
              precision    recall  f1-score   support

       False       0.92      0.98      0.95      5436
        True       0.99      0.96      0.97     11139

    accuracy                           0.97     16575
   macro avg       0.95      0.97      0.96     16575
```

Figure 5: Accuracy report for rf model

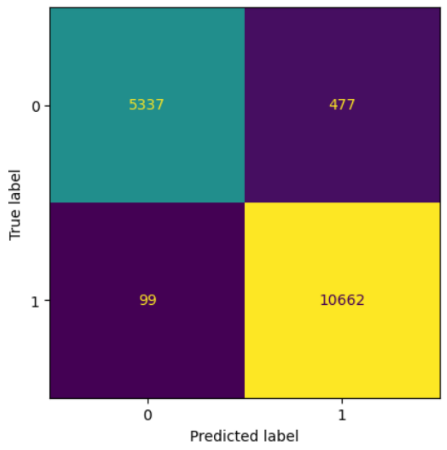


Figure 6: Confussion matrix for rf model

2.3.2 Random forest

Our following attempt to solve this classification task was implementing a random forest model using scikit-learn.

It is created a Random Forest classifier model with the *n_estimators* parameter set to 100, which is the number of decision trees (estimators) in the ensemble. In this case,it means that the Random Forest will consist of 100 decision trees.

Figure 5 and *Figure 6* highlight the results obtained with this model on the validation set. As for the test set, the score on Kaggle is 0.805

```
model_rf = RandomForestClassifier(n_estimators=100)
model_rf.fit(train_content, train_label)
predict_rf_test = model_rf.predict(test_content)
```

2.3.3 Stochastic Gradient Descent

SGDClassifier implements a linear classifier (SVM, logistic regression, or perceptron) using stochastic gradient descent (SGD) learning. It is well-suited for large-scale text classification tasks. *max_iter* parameter specifies the maximum number of iterations (epochs) to train the classifier. Training stops when this maximum number of iterations is reached or when convergence is achieved.

This model displayed similar results to the logistic regression model.

```
sgd = SGDClassifier(max_iter=2000)
sgd.fit(train_content, train_label)
predictions = sgd.predict(test_content)
```

2.3.4 Fully Connected Neural Networks

We have constructed a neural network model for binary text classification using TensorFlow and Keras in Python. The sequential model has three layers:

- `tf.keras.layers.Dense(128, activation='relu', input_shape= (train_content.shape[1],))`: dense layer which adds a fully connected layer with 128 units and ReLU activation function
- `tf.keras.layers.Dense(64, activation='relu')`: adds another dense layer with 64 units and ReLU activation function
- `tf.keras.layers.Dense(1, activation='sigmoid')`: adds the output layer with a single unit (for binary classification) and sigmoid activation function, which outputs a probability score between 0 and 1 representing the likelihood of the positive class.

The model is compiled using *adam* optimizer, an efficient and popular optimization algorithm commonly used in training neural networks, which adapts the learning rate for each parameter during training. And *binary_crossentropy* for the loss function, which measures the difference between the true labels (0 or 1) and the predicted probabilities output by the model.

On the test data this model has shown the best accuracy of 0.88 on the test set on Kaggle

```
    model_retea = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=
                            (train_content.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

    model_retea.compile(optimizer='adam',
                        loss='binary_crossentropy',
                        metrics=['accuracy'])

    model_retea.fit(train_content, train_label, epochs=10, batch_size=
    32, validation_data=(validation_content, validation_label))

    predict_retea_test = (model_retea.predict(test_content) > 0.5)
                            .astype(int)
```

3 Conclusion

In summary, for the give text classification task, we explored various machine learning and deep learning models, each offering unique advantages and suitable applications, this project being a rewarding exploration into the world of text classification, equipping us with practical skills in machine learning, deep learning, and natural language processing.