MOROCO sub-dialect identification Report

Brâncoveanu Anca-Maria, 242

Description of the Machine learning approach:

Along the way I have tried various approaches to solve the competition. Among these approaches I used models such as Bag of Words, Linear SVC, Multilayer Perceptron, Logistic Regression, Keras, FastAI, Kernel Ridge Regression and Complement Naive Bayes. The one that had the best score was Complement Naive Bayes, solution that was kept for the final evaluation.

“ComplementNB implements the complement naive Bayes (CNB) algorithm. CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the complement of each class to compute the model’s weights. The inventors of CNB show empirically that the parameter estimates for CNB are more stable than those for MNB. Further, CNB regularly outperforms MNB (often by a considerable margin) on text classification tasks. The procedure for calculating the weights is as follows:

$$\hat{\theta}_{ci} = \frac{\alpha_i + \sum_{j:y_j \neq c} d_{ij}}{\alpha + \sum_{j:y_j \neq c} \sum_k d_{kj}}$$

$$w_{ci} = \log \hat{\theta}_{ci}$$

$$w_{ci} = \frac{w_{ci}}{\sum_j |w_{cj}|}$$

where the summations are over all documents j not in class c, d_{ij} is either the count or tf-idf value of term i in document j, α_i is a smoothing hyperparameter like that found in MNB, and $\alpha = \Sigma_i \alpha_i$. The second normalization addresses the tendency for longer documents to dominate parameter estimates in MNB. The classification rule is:

$$\hat{c} = \arg\min_c \sum_i t_i w_{ci}$$

i.e., a document is assigned to the class that is the poorest complement match.”[1]

“An n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The n-grams typically are collected from a text or speech corpus. When the items are words, n-grams may also be called shingles.”[2]

Steps for my implementation of the Complement Naive Bayes solution:

1. I started with importing the data from the given files and then I extracted the features (the words) into a vector for easier manipulation. For vectorising I used *CountVectorizer* (from sklearn.feature_extraction.text) with the following parameters:

- lowercase=False: used because there are both lowercase and uppercase letters
- ngram_range=(4, 5): used to get n-grams; 4 and 5 represents the boundaries
- analyzer='char_wb': analyser to tell if the feature should be a n-gram word or character; 'char_wb' creates character n-grams only from text inside word boundaries

2. Afterwards, for better training data, I normalised the training, validation and testing vectors using the *TfidfTransformer* (term-frequency times inverse document-frequency transformer from sklearn.feature_extraction.text) scaler which transformed a count matrix to a normalized tf or tf-idf representation.
3. In order to represent the whole process I used the *Complement Naive Bayes* model with parameter alpha = 0.005 as a smoothing parameter. I trained the model and then I predicted the testing and validation vectors.
4. Finally, I printed the classification report with precision, recall, f1-score and support columns for validation labels and validation prediction with an accuracy of 7 digits. The .csv file was created using np.vstack with the testing_samples['id'] and test_prediction parameters. Because the file would contain 2 lines of data and I needed 2 columns of data, I transposed the results using .T .
5. For the confusion matrices I used the np.set_printoptions(precision=2) function, where the precision parameter represents the number of digits of precision for floating point output

Macro F1 scores for each provided validation set and confusion matrices for the final model:

I will present differences between 2 of my best approaches: Complement Naive-Bayes and MLPClassifier

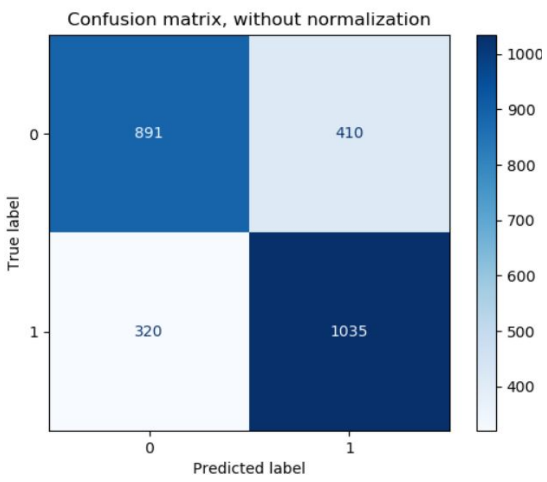
Complement Naive Bayes:

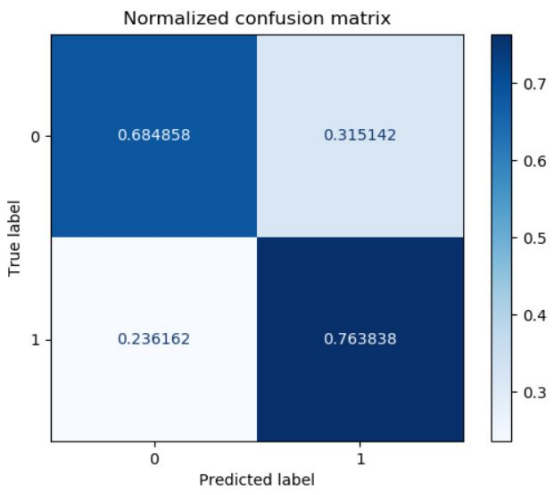
table/matrix	F1 score	n-grams	alpha
1	0.62294	1, 2	0.001
2	0.62257	1, 2	0.005
3	0.78688	1, 3	0.005
4	0.92935	4, 5	0.005

MLPClassifier

table/matrix	F1 score	n-grams	hidden layer sizes
5	0.33782	1, 2	2, 8
6	0.33782	2, 5	2, 8
7	0.33782	3, 5	4, 8
8	0.63394	1, 2	16, 32

Confusion matrices for the best ComplementNB model:





The Python code of my approach:

```
1. import numpy as np
2. from sklearn.metrics import classification_report
3. from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
4. from sklearn.naive_bayes import ComplementNB
5. import matplotlib.pyplot as plt
6. from sklearn.metrics import plot_confusion_matrix
7.
8. # Reading data from folder
9. training_samples = np.genfromtxt('ml-2020-unibuc-3/train_samples.txt',
   dtype=None, comments=None, encoding="utf-8", delimiter='\t', names=('id',
   'text'))
10. training_labels_samples = np.genfromtxt('ml-2020-unibuc-3/train_labels.txt',
   dtype=None, comments=None, names=('id', 'label'))
11. validation_samples = np.genfromtxt('ml-2020-unibuc-3/validation_samples.txt',
   dtype=None, comments=None, encoding="utf-8", delimiter='\t', names=('id',
   'text'))
12. validation_labels_samples =
   np.genfromtxt('ml-2020-unibuc-3/validation_labels.txt', dtype=None,
   comments=None, names=('id', 'label'))
13. testing_samples = np.genfromtxt('ml-2020-unibuc-3/test_samples.txt', dtype=None,
   comments=None, encoding="utf-8", delimiter='\t', names=('id', 'text'))
14.
15. # What kind of data was extracted from files
16. training_words = training_samples['text']
17. training_labels = training_labels_samples['label']
18. validation_words = validation_samples['text']
19. validation_labels = validation_labels_samples['label']
20. testing_data = testing_samples['text']
21.
22. # After the model is tested on the normal data, it is trained on validation
23. #training_words = np.append(training_words, validation_words)
24. #training_labels = np.append(training_labels, validation_labels)
25.
26. # Transform corpus to a vector of term/token counts
27. vector = CountVectorizer(lowercase=False, ngram_range=(4, 5),
   analyzer='char_wb')
28. training_vector = vector.fit_transform(training_words, training_labels)
29. validation_vector = vector.transform(validation_words)
30. testing_vector = vector.transform(testing_data)
31.
32. # Training tool that allows usage of better training data
33. scaler = TfidfTransformer()
34. normalized_training_vector = scaler.fit_transform(training_vector)
35. normalized_validation_vector = scaler.transform(validation_vector)
36. normalized_testing_vector = scaler.transform(testing_vector)
37.
38. # Mathematical representation of a real-world process
39. model = ComplementNB(alpha=0.005)
```

```
model.fit(normalized_training_vector, training_labels)
# Getting predictions
test_prediction = model.predict(normalized_testing_vector)
validation_prediction = model.predict(normalized_validation_vector)

# Report with precision, recall, f1-score and support
print(classification_report(validation_labels, validation_prediction, digits=7))

# Print results in a .csv file
submission = np.vstack((testing_samples['id'], test_prediction)).T
np.savetxt('MOROCOdialect.csv', submission, fmt='%s', delimiter=',',
header='id,label', comments='')

# Plot confusion matrix
titles_options = [('Confusion matrix, without normalization', None, 'd', ''),
('Normalized confusion matrix', 'true', 'f', '_norm')]
np.set_printoptions(precision=2)
for title, norm, fmt, extension in titles_options:
    display = plot_confusion_matrix(model, normalized_validation_vector,
validation_labels, normalize=norm, cmap=plt.cm.Blues, values_format=fmt)
    display.ax_.set_title(title)
    print(title)
    print(display.confusion_matrix)
    plt.savefig('confusion_matrix' + extension)
    plt.close()
```

Bibliography:

1. https://scikit-learn.org/stable/modules/naive_bayes.html; Rennie, J. D., Shih, L., Teevan, J., & Karger, D. R. (2003). Tackling the poor assumptions of naive bayes text classifiers. In ICML (Vol. 3, pp. 616-623).
2. https://en.wikipedia.org/wiki/N-gram#n-gram_models, last edited on 18 March 2020, at 17:43 (UTC).
3. Understanding and writing of the code: https://scikit-learn.org/stable/