Machine Learning for Natural Language Processing 1
Exercise 1

Part 1

Other features that can be used to determine the language:

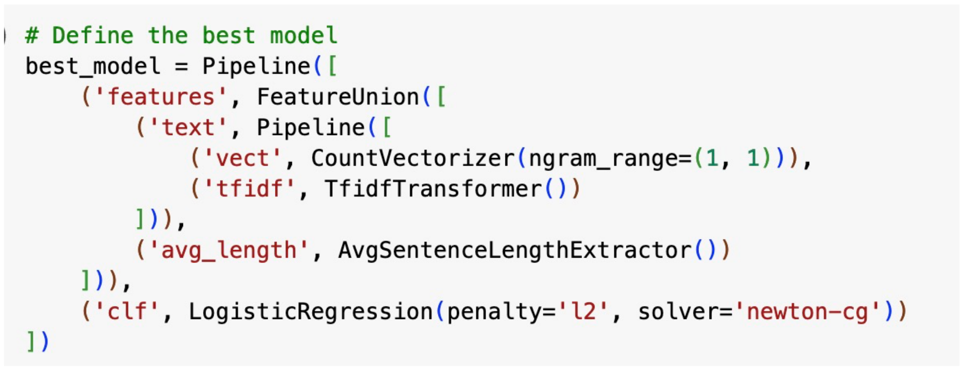
- Average word length / sentence length
- Part-of-Speech (POS) Tags
- Dependencies
- N-grams

Here we employed the average sentence length as an additional linguistics feature with the help of NLTK, and added it into the pipeline.

Hyperparameter Optimization:

- **Penalty (Regularization):** 'l1', 'l2'
- **Solver:** 'liblinear', 'saga', 'newton-cg'
- **Vectorizer Parameters:**
  - ngram_range: [(1, 1), (1, 2), (2, 2)]

Here is our best hyperparameter combination:

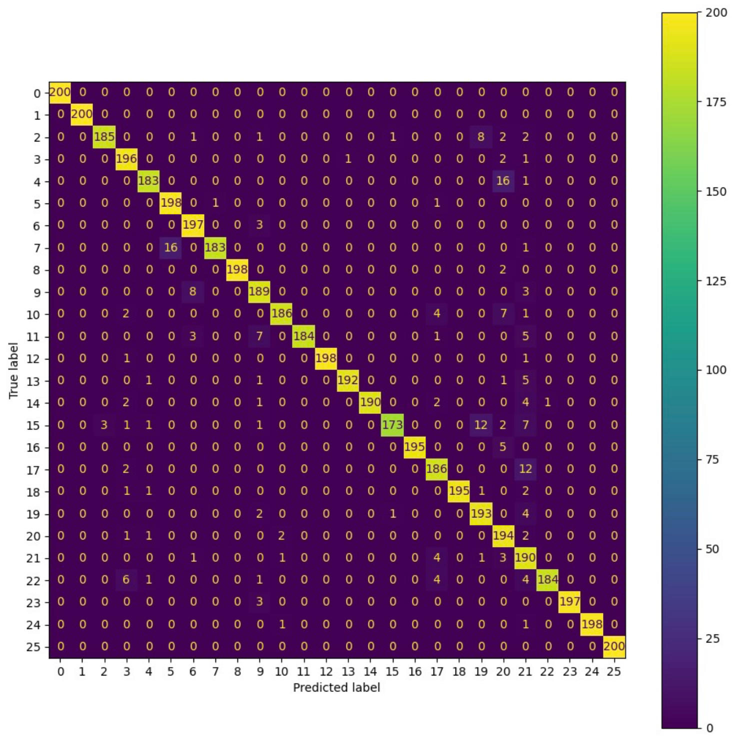


```
# Define the best model
best_model = Pipeline([
    ('features', FeatureUnion([
        ('text', Pipeline([
            ('vect', CountVectorizer(ngram_range=(1, 1))),
            ('tfidf', TfidfTransformer())
        ])),
        ('avg_length', AvgSentenceLengthExtractor())
    ])),
    ('clf', LogisticRegression(penalty='l2', solver='newton-cg'))
])
```

Advantages of Grid Search Cross-Validation:

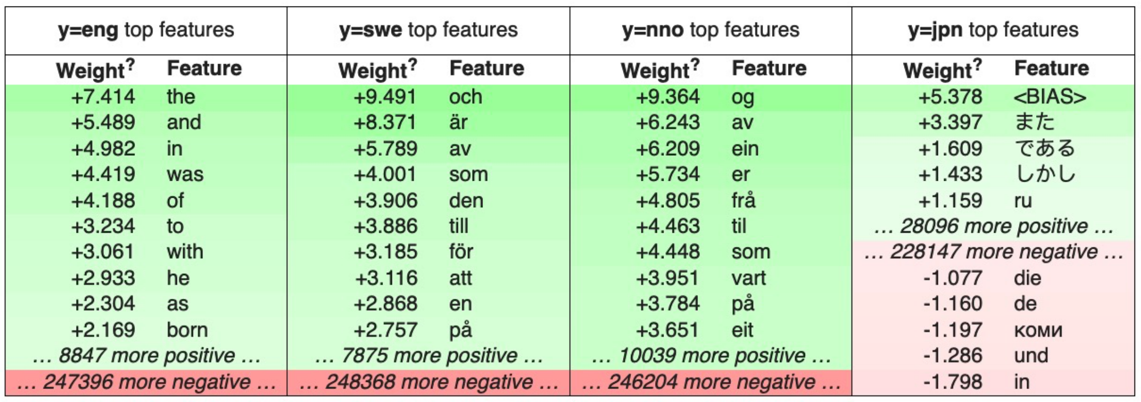
- Grid Search systematically explores a predefined set of hyperparameter combinations.
- It efficiently tunes multiple hyperparameters simultaneously, reducing the need for manual trial-and-error tuning.
- Grid Search incorporates cross-validation, which ensures a more robust evaluation of the model's performance.

The confusion matrix is shown as below:



As we can see, The confusion matrix provides a detailed understanding of the model's performance and offers insights into specific types of errors made by the model.

The Feature Importance Table:



y=eng top features		**y=swe** top features		**y=nno** top features		**y=jpn** top features	
Weight?	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**
+7.414	the	+9.491	och	+9.364	og	+5.378	<BIAS>
+5.489	and	+8.371	är	+6.243	av	+3.397	また
+4.982	in	+5.789	av	+6.209	ein	+1.609	である
+4.419	was	+4.001	som	+5.734	er	+1.433	しかし
+4.188	of	+3.906	den	+4.805	frå	+1.159	ru
+3.234	to	+3.886	till	+4.463	til	*… 28096 more positive …*	
+3.061	with	+3.185	för	+4.448	som	*… 228147 more negative …*	
+2.933	he	+3.116	att	+3.951	vart	-1.077	die
+2.304	as	+2.868	en	+3.784	på	-1.160	de
+2.169	born	+2.757	på	+3.651	eit	-1.197	коми
… 8847 more positive …		*… 7875 more positive …*		*… 10039 more positive …*		-1.286	und
… 247396 more negative …		*… 248368 more negative …*		*… 246204 more negative …*		-1.798	in

The outputs of the vectorizer is much more important than our extra linguistic feature.

Ablation:

The two languages we selected are: (the first and last labels in the matrix)

```
selected_languages = ['ace', 'swe']
```

Results:

1. Character Limit: None (All characters)
   - Accuracy: 1.0
2. Character Limit: 500
   - Accuracy: 1.0
3. Character Limit: 100
   - Accuracy: 0.9975
4. Character Limit: 20
   - Accuracy: 0.715

Observations:

In the ablation study, the classifier achieved the accuracy (1.0) when exposed to the entire text (No Character Limit). Even with a substantial reduction to 500 characters, high accuracy (1.0) was maintained. Further reducing to 100 characters resulted in a slight decrease in accuracy (0.9975), showing a slight reduction in terms of accuracy.

Therefore, we tried to reduce the characters to 20 as an extreme case. It has been proven that a severe reduction to 20 characters led to a significant drop in accuracy (0.715), suggesting the model's struggle to capture meaningful features with such limited information.

Interpretation:

- The limitation of the number of training characters will reduce the accuracy of the model.
- The observed drop in accuracy at 20 characters suggests that the model's performance is sensitive to extremely limited text, indicating a threshold below which it struggles to make accurate predictions.

Part 2

Hyperparameter Exploration Process:

- **Layer Sizes:**
  - Experimented with varying the number of units in both hidden layers.
- **Activation Functions:**
  - Tried to use “F.relu” and “F.tanh” in both layers.
- **Solvers:**
  - Explored different solver approahes, including torch.optim.RMSprop, torch.optim.Adam, torch.optim.SGD, etc.
  - Adjusted the learning rate including 0.1, 0.01, 0.05, 0.00, etc.
- **Early Stopping:**
  - Changed different Patience and Threshold.

- Vectorizer Parameters:
  - Tried both character-level and word-level analysis with ngram_range=(1, 1), (1, 2), (2, 2).
  - Experimented different max features limilation.
  - binary=False or True

Here is our best hyperparameter combination:

- Layer Sizes:
  - “num_units” in Module1: 2500
  - “nn.Linear” layers: 3773 -> 2500, 2500 -> 256, 256 -> 26
- Activation Functions:
  - “F.relu” for the first layer
  - “F.tanh” for the second layer
- Solvers:
  - “torch.optim.RMSprop” with learning rate 0.001
- Early Stopping:
  - Patience: 10 epochs
  - Threshold: 0.0001, relative improvement
- Vectorizer Parameters:
  - CountVectorizer with character-level analysis, considering unigrams (ngram_range=(1, 1)), without max features limilation, binary=False

Performance:
- Accuracy: 0.9711 (97.11%)

Analysis:

Our neural network model has demonstrated better performance compared to the linear model employed in Part 1. It can be attributed to the architectural complexity and non-linear activation functions inherent in the neural network, which prove effective in capturing intricate patterns within the data.
Additionally, it is noteworthy that the linguistic feature introduced in Part 1, unfortunately, contributed to a decline in model performance, as evidenced by lower accuracy scores.