Assignment 1 Report

Part 1 - Language identification with linear classification

Tasks

1. Have a quick peek at the training data, looking at a couple of texts from different languages. Do you notice anything that might be challenging for the classification

   Answer: Several languages in the Latin family use the same alphabet, and the words they form are extremely similar.

2. How many instances per label are there in the training and test set?

   Answer: 500 instances per label in the training set and 500 instances per label in the test set.

3. Do you think this is a balanced dataset?

   Answer: To some extent, yes, each language has the same number of instances. However, the training set should contain more instances than the test set.

4. Do you think the train/test split is appropriate?

   Answer: No. Training set should have more instances in each language than the test set. Like 4:1. Here it's 1:1.

5. If not, please rearrange the data in a more appropriate way.

   Answer: (Have done in notebook) Here is the approach we use: Firstly, we concat train_df with test_df to get a whole data frame. Each label has 1000 instances now. Secondly, for each label, we select 800 samples as new training set, and remaining 200 samples as new test set. Each language has 800 instances in the new training set and 200 instances in the new test set, where the 'training size: test size = 8:2'.

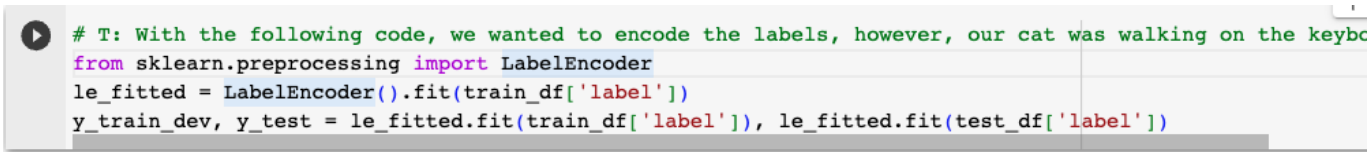
6. Get a subset of the train/test data that includes English, German, Dutch, Danish, Swedish and Norwegian, plus 20 additional languages of your choice (the labels can be found in the file labels.csv)

   Answer: (Have done in notebook) English: eng, German: deu, Dutch: nld, Danish: dan, Swedish: swe, Norwegian: nob, Japanese: jpn, and we randomly select 20 other languages in the remaining except for those ones above.

7. With the following code, we wanted to encode the labels, however, our cat was walking on the keyboard and some of it got changed. Can you fix it?

   Answer:

   original code:



```
# T: With the following code, we wanted to encode the labels, however, our cat was walking on the keybo
from sklearn.preprocessing import LabelEncoder
le_fitted = LabelEncoder().fit(train_df['label'])
y_train_dev, y_test = le_fitted.fit(train_df['label']), le_fitted.fit(test_df['label'])
```

   modified code:

```
# T: With the following code, we wanted to encode the labels,
# however, our cat was walking on the keyboard and some of it got changed. Can you fix it?
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
le_fitted = label_encoder.fit(sub_train_set['label'])
y_train_dev, y_test = le_fitted.transform(sub_train_set['label']), le_fitted.transform(sub_test_set['label'])
```

Reason: We can only fit the training data but not the test data. We can first fit the training data and then transform the training data and test data. So the code "le_fitted.fit(test_df['label'])" is fixed.

Pipeline

We use Pipeline from sklearn for more conveniently running the experiments. We use TfidfVectorizer to replace CountVectorizer and TfidfTransformer. There are two reasons:

1. Due to Eli5, if we use TfidfTransformer there would be errors.
2. This is also more clear and intuitive

Our pipeline is like:

```
text_clf = Pipeline([
    ('TfidfVect', TfidfVectorizer(ngram_range=(1, 2),)),
    ('LR', LogisticRegression())
])
```

Q: **What other features could you use to determine the language? Please include additional linguistic features to your machine learning model for this task.**

A: We apply ngram_range: (1, 2) to replace the original (1, 1), which means unigrams and bigrams are both used. We extend the feature space than the initial param setting.

After vectorizing, we use LogisticRegression() as our model here. The cross validation score after training is [0.96125 , 0.96013889, 0.96333333].

Hyperparameters

We use sklearn's GridSearchCV to do cross validation to find the optimal hyperparameters.

```
param_grid = {'LR__penalty': ['l1', 'l2'],
              'LR__solver': ['liblinear', 'saga'],
              'TfidfVect__ngram_range': [(1, 1), (1, 2)],
              'TfidfVect__norm': ['l1', 'l2']}
```

The hyperparameters we want to search for are:

1. regularization: we try 'L1' and 'L2'
2. Solver: we try 'liblinear' and 'saga'
3. Vectorizer: we try 'unigram' or 'unigrams and bigrams', plus adding 'L1' and 'L2' normalizations

The best-performing model's hyperparameter combination:

> 'LR__penalty': 'l2',
>
> 'LR__solver': 'saga',
>
> 'TfidfVect__ngram_range': (1, 1),
>
> 'TfidfVect__norm': 'l2'

Q: **What is the advantage of grid search cross-validation?**

A: GridSearchCV tries all the combinations of the values passed in the dictionary and evaluates the model for each combination using the Cross-Validation method. Hence after using this function we can get accuracy/loss for every combination of hyperparameters and we can choose the one with the best performance.

Best Model and Error Analysis

Then we train our best-performing model on the training data and get confusion matrix based on the predictions and test data. Here is our discovery:

Diagonal Elements: In the confusion matrix, from the top-left to the bottom-right elements, all of these diagonal elements are close to 200, which is the total number of samples of each label in

the test set. This means the true positive values for each label is relatively high, indicating that the classifier performs well on most classes.

Non-Diagonal Elements: However, there are also non-diagonal elements, which are errors (wrong predictions). The maximum number a class is misclassified is 15, which is the number 'ang' texts are misclassified as 'eng' texts. Another big number is 14, which shows the count of 'pcd' samples that are misclassified as 'eng' texts. To be noticed, we can find that there are also some misclassifications from 'eng' to 'ang' or 'pcd' languages, though a bit fewer. These results indicate that there are some confusion between these pairs of languages.

According to the confusion matrix, we can calculate the precision, recall and f1 score for each language as shown below.

	precision	recall	f1-score	support
ang	0.98	0.93	0.95	200
bar	0.95	0.93	0.94	200
bod	1.00	1.00	1.00	200
dan	0.99	0.97	0.98	200
deu	0.92	0.94	0.93	200
dsb	1.00	0.96	0.98	200
eng	0.80	0.95	0.87	200
hat	1.00	0.98	0.99	200
hrv	0.99	0.98	0.99	200
hun	1.00	0.97	0.98	200
jav	1.00	0.97	0.98	200
jpn	0.80	0.97	0.88	200
kab	1.00	0.94	0.97	200
kbd	1.00	0.99	1.00	200
kor	1.00	0.99	0.99	200
ksh	0.99	0.97	0.98	200
lad	0.99	0.98	0.99	200
mar	1.00	0.97	0.98	200
mkd	1.00	0.98	0.99	200
mwl	0.98	0.99	0.99	200
nav	1.00	1.00	1.00	200
nds-nl	0.96	0.97	0.96	200
nld	0.98	0.95	0.97	200
nob	0.99	0.99	0.99	200
pcd	0.91	0.89	0.90	200
swe	1.00	1.00	1.00	200
urd	1.00	0.99	0.99	200
accuracy			0.97	5400
macro avg	0.97	0.97	0.97	5400
weighted avg	0.97	0.97	0.97	5400

Feature Importance Table for the Top Ten Features

y=eng top features		y=swe top features		y=nob top features		y=jpn top features	
Weight[?]	Feature	Weight[?]	Feature	Weight[?]	Feature	Weight[?]	Feature
+7.963	the	+9.226	och	+9.693	og	+6.524	また
+6.145	and	+8.776	är	+7.339	av	+3.369	しかし
+5.407	in	+6.029	av	+5.647	ble	+2.940	なお
+4.143	was	+3.999	till	+5.495	er	+2.303	その後
+4.005	of	+3.993	som	+4.910	til	*... 12620 more positive ...*	
+3.285	to	+3.927	den	+4.903	som	*... 247999 more negative ...*	
+2.929	with	+3.190	att	+4.171	på	-2.009	and
+2.780	by	+3.103	för	+3.864	det	-2.374	se
+2.777	he	+2.945	på	+3.236	med	-2.548	is
+2.762	as	+2.867	finns	+2.829	for	-2.850	en
... 8879 more positive ...		*... 7460 more positive ...*		*... 11731 more positive ...*		-3.683	in
... 251740 more negative ...		*... 253159 more negative ...*		*... 248888 more negative ...*		-4.962	de

What is more important, extra features or the outputs of the vectorizer?

For vectorizer, outputs from vectorizer are the primary source of information for language classification, they capture the linguistic characteristics and patterns of each language, so they play a crucial role in achieving good accuracy.

For extra features, they can vary based on their relevance to the classification, and the source they come from. Based on the table, we can see there are more than 10k positive features and over 240k negative features used to predict the correct class. Obviously, extra features are also important.

We think they are both important in this task. To say which is better in other tasks, it depends on the specific language classification task and the relevance of additional data.

Ablation Study

Q: **How does the ablation affect the performance of the classifier?**

A: We set the number of characters per instance in the training set (1. All characters, 2. 500 characters, 3. 100 characters, 4. 10 characters (added).

Case 1:

The two languages for which the classifier worked best are: 'bod', 'nav'.

> Accuracy when length is original: 1.0
>
> Accuracy when length is 500: 1.0
>
> Accuracy when length is 100: 1.0
>
> Accuracy when length is 10: 1.0

As a result, the ablation has no influence on the accuracy. This is probably because these two languages ('nav' and 'bod') use completely different vocabulary, they are very easy to distinguish by the best model. In this case, length of the text is slightly influential.

Case 2:

To see if the accuracy will decrease in other conditions, we choose two languages like 'nob' and 'swe'.

> Accuracy when length is original: 1.0
>
> Accuracy when length is 500: 1.0
>
> Accuracy when length is 100: 1.0
>
> Accuracy when length is 10: 0.9225

There is still little influence on accuracy when length is not very small. When the length reduces to 10, there is an obvious drop in accuracy.

To summarize it, the ablation barely affect the performance of the classifier.

Part 2 - Your first Neural Network

Tasks

> 1. Please use again the train/test data that includes English, German, Dutch, Danish, Swedish and Norwegian, plus 20 additional languages of your choice (the labels can be found in the file labels.csv) and adjust the train/test split if needed
> 2. Use your adjusted code to encode the labels here
> 3. In the following, you can find a small (almost) working example of a neural network. Unfortunately, again, the cat messed up some of the code. Please fix the code such that it is executable.
>
>   (All have been done in notebook)

Improve the Neural Network

First, we use the default hyperparameters. After 500 epochs, the validation accuracy is still under 80%. The result is as followed:

Training accuracy: 0.8454807692307692

Testing accuracy: 0.24826923076923077

The difference between training accuracy and testing accuracy is quite large, which means that there may be an overfitting in the training data.

To improve the model, we use the GridSearchCV to find the best parameters.

We make a pipeline as our model and use 'to_float32' to transform the output of the vectorizer to float 32, else there will be an error.

```
text_NN = Pipeline([
    ('CV', CountVectorizer(max_features=100, binary=True)),
    ('to_float32', FunctionTransformer(to_float32)),
    ('NN', NeuralNetClassifier(
    ClassifierModule,
    callbacks = [EarlyStopping()],
    max_epochs=1000,
    criterion=nn.CrossEntropyLoss(),
    lr=0.1,
    verbose = 1,
))
])
```

Except the parameters we fixed (max_epochs and CV max_features), the parameters we want to find best include 5 requirements in Exercise 1 document.

```
param_grid = {  # layer sizes
                'NN__module__num_units': [200, 300],

                # activation function
                'NN__module__nonlin': [nn.ReLU(), nn.LeakyReLU()],

                # solvers
                'NN__optimizer__lr': [0.1, 0.01],

                # early stopping
                'NN__callbacks__EarlyStopping__patience': [10, 20],

                # vect params
                'CV__ngram_range': [(1, 1), (2, 2)],
    }
```

We use 3 cross-validation sets. For saving GridSearchCV time and improving the efficiency, we sample 600 instances from the sub training set, and make it a new dataset.

The best parameters are:

> 'CV__ngram_range': (1, 1),
>
> 'NN__callbacks__EarlyStopping__patience': 20,
>
> 'NN__module__nonlin': LeakyReLU(negative_slope=0.01),
>
> 'NN__module__num_units': 300,
>
> 'NN__optimizer__lr': 0.1

Using the params we got from GridSearchCV, we explore if increasing the number of features will make the accuracy higher. The parameter is both max_features for CV and in_features for NN (they should be the same).

We test features in [100, 200, 500, 5000].

As we increase the number, the model performs better.

Result:

> Model with features = 100 has test accuracy around 0.76 < 0.8
>
> Model with features = 200, 500, 5000 has test accuracy around 0.83, 0.89, 0.96, all higher than 0.8.

So we have obtained higher performance, because we:

1. Use GridSearchCV to find the best performing hyperparameters. For example, setting the earlystopping will reduce the overfitting risk.
2. We increase the number of max features the vectorizer can extract and this provides the model with more feature information for classification.