Machine Learning for Natural Language Processing I

Exercise 1 Lab Report

Language Classification

Content

1. Dataset

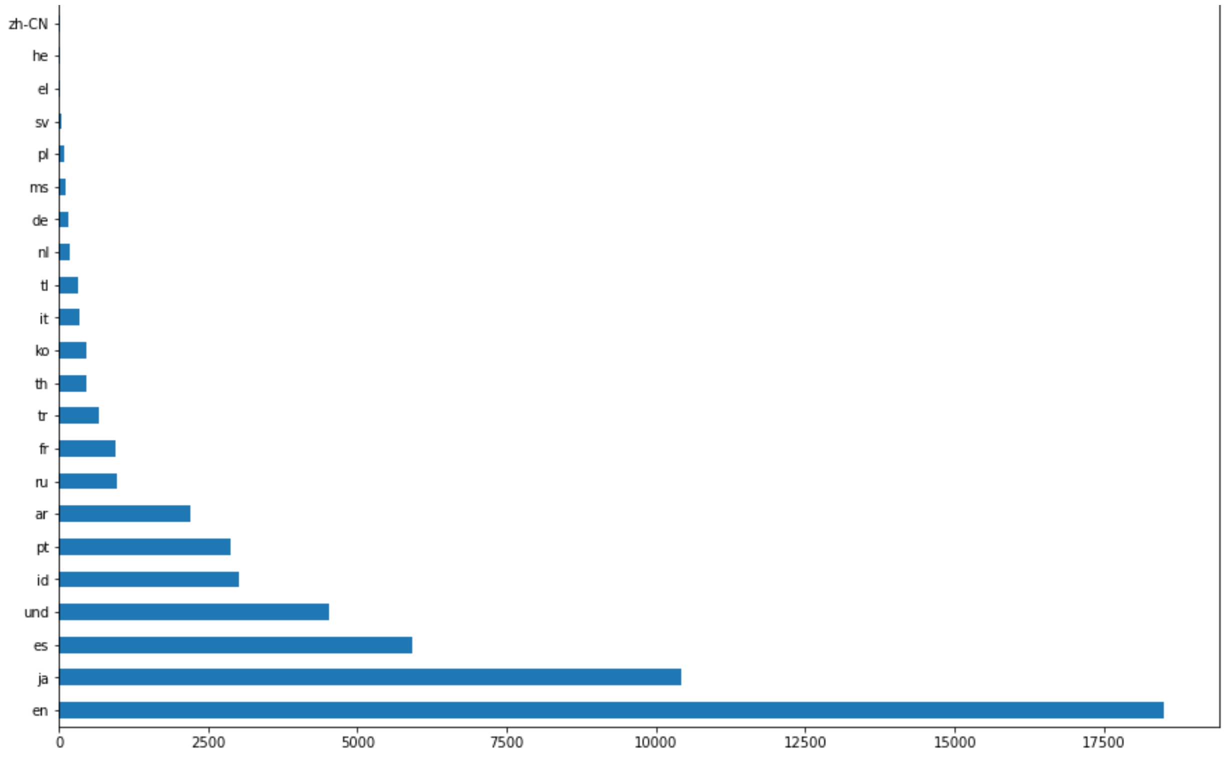
The dataset has two columns, text and language. The text column is a context from a particular language, which is labelled in the language column. They are separated into train dataset (`X_train` and `y_train`), validation dataset[1] (`X_val` and `y_val`) and test dataset (`X_test` and `y_test`) with size of 47408, 5268 and 13280 respectively with no null values. The train dataset is used for training and fine-tuning. After each training, the trained model will be put into the validation dataset for evaluation. For a task, there will be several fine-tuned models, and the one with the best accuracy will be put into the test dataset for the final performance evaluation. Except for the final model, models shouldn't be trained or evaluated by the test dataset.

There are a total of 85 languages in the datasets. The languages with the most contexts are English, Japanese and Spanish for all datasets. However, there are some languages that appear

[1] In the report, the development dataset will also be referred to as the validation dataset.

in the validation or test dataset, but not in the train dataset. There are four additional languages in the validation dataset and twelve additional ones in the test dataset. Although the number of the missing languages are not considerably small, the total rows responsible for those languages are very few. Thus, this effect can be neglected and should not be worried.

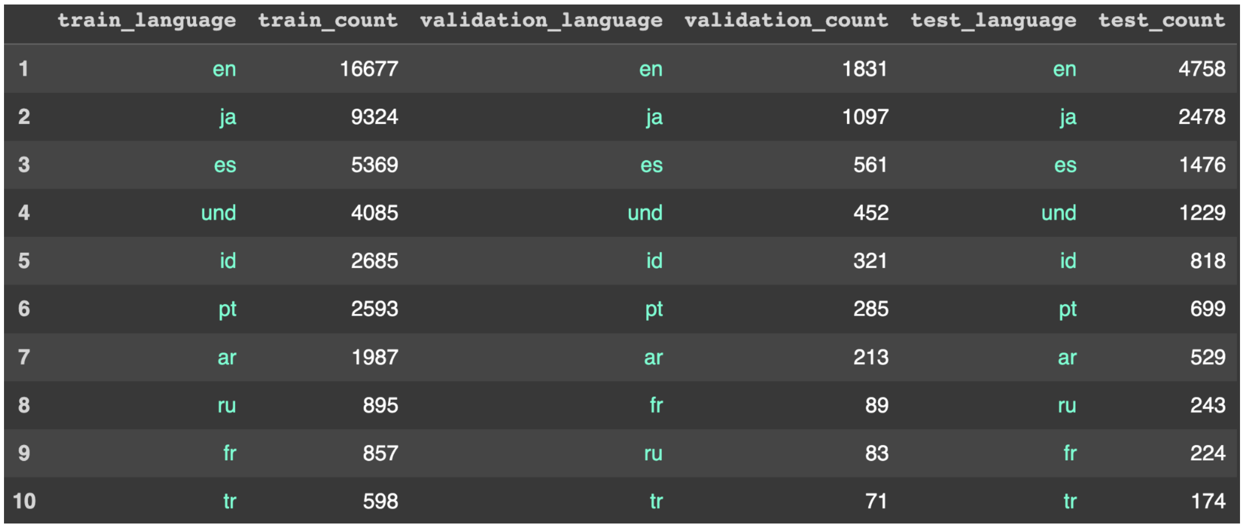
The figure below is the bar chart of the number of languages in the train dataset. Clearly, English, Japanese and Spanish appear most in the dataset.



This is the result printed from the console, showing the missing languages that appear in the validation dataset or the test dataset, but not the train dataset.

```
These validation classes doesn’t appear in training dataset:

ta_LATN, tn, dv, ht

The total counts are: 5

These testing classes doesn’t appear in training dataset:

ht, mn, mk, sk, mr, yo, lv, eu, ta_LATN, la, zu

The total counts are: 16
```

In this table, it shows that the top ten languages in all three datasets are the same.



	train_language	train_count	validation_language	validation_count	test_language	test_count
1	en	16677	en	1831	en	4758
2	ja	9324	ja	1097	ja	2478
3	es	5369	es	561	es	1476
4	und	4085	und	452	und	1229
5	id	2685	id	321	id	818
6	pt	2593	pt	285	pt	699
7	ar	1987	ar	213	ar	529
8	ru	895	fr	89	ru	243
9	fr	857	ru	83	fr	224
10	tr	598	tr	71	tr	174

2. Methods

The data is first preprocessed, vectorised and then joined with other extra features that are expected to help improve the accuracy of the models. Throughout the task, sklearn is used and the usage of pipeline helps enhance the readability of the code.

2.1 Preprocessing

Preprocessing is done by removing unnecessary contexts, lowering cases and lemmatisation. Unnecessary contexts are mostly things that can be found in all languages. This refers to HTML tags, URLs and emojis. After this, the contexts will be converted into feature vectors by `CountVectorizer` and `TfidfTransformer`. `CountVectorizer` takes words or characters as features, and generates the corresponding number of occurrences in the context. `TfidfTransformer` tends to look at the whole picture instead of a single sentence to give a penalty to words/characters that simply appear a lot in all contexts. For this task, CountVectorizer is trained in both word and character manner, which will become clearer in the Pipeline section.

Note that stop words and punctuations are not removed in this task. Stop words are not removed because they may be a good indication of the language. For instance, “you” is a frequent stop word in English, but does not appear in any other languages. Thus, if a sentence has “you” in it, it has a high chance of being English. For the same reason, punctuations are also not removed. For instance, the bracket『』appears quite frequently in Japanese and Chinese, but not in English or German. Thus, the punctuations should be kept.

As for the label, it is encoded by `LabelEncoder`. It has `fit` and `transform` methods. However, simply fitting on train dataset and then transforming on test dataset will not work because there are unseen classes in the test dataset. Thus, in this task, the train dataset and the test dataset are fit with the `LabelEncoder` together and then separated into train dataset and test dataset labels.

2.2 Extra Features Generation and Union

Extra features are first generated as arrays, and by FeatureUnion, the extra features can be joined with other features generated by `CountVectorizer` and `TfidfTransformer`. In this task, three extra features are generated, which are number of spaces, average word length and number of capital letters. These three extra features are named as `num_space`, `avg_word_len` and `num_capital` respectively in the code.

- num_space: the number of spaces in the context
- avg_word_len: $\frac{total\ number\ of\ characters}{total\ number\ of\ words}$
- num_capital: the number of capital letters in the context

The reason for choosing num_space is that most alphabetical languages have spaces between words, while Chinese, Japanese do not. And the reason for choosing avg_word_len is that some languages tend to combine words into a vocabulary, and thus leads to longer average word length. In addition, some languages, German for instance, have capital letters for nouns, and will result in higher number of capital letters.

2.3 Pipeline

Pipeline is an extremely useful tool that allows the user to put several transformers into it one by one, and those transformers will be executed in that order. In this task, preprocessing and generating extra features are written as transformers by inheritancing `BaseEstimator` and `TransformerMixin`, and then put into the pipeline for execution. The pipeline is illustrated below.

```
Pipeline([("Preprocessor", Preprocessing("preprocess")),
          ("countvect"), CountVectorizer()),
          ("features", FeatureUnion([
                        ("tfidf", TfidfTransformer()),
                         ("num_space", ExtraFeature(X_num_space)),
                        ("avg_word_len",
ExtraFeature(X_avg_word_len)),
                        ("num_capital", ExtraFeature(X_avg_word_len)),
          ("clf", LogisticRegression())])
```

3. Linear Model - Logistic Regression

The first classifier used for this task is the logistic regression for multi-class classification. GridSearchCV is first used for tuning hyperparameters with train dataset and evaluate via validation dataset, and then the test dataset is used for showing the final performance.

3.1 GridSearchCV

GridSearchCV is used for hyperparameter tuning for determining the finest model. In the task, the following combinations are tried for the task. Param_grid_1 and Param_grid_2 both used penalty and solver for the classifier. The main difference is that, Param_grid_1 used characters as features in CountVectorizer, while Param_grid_2 used words. With characters as features, ngram_range allows using 1, 2, or 3 characters as a group for generating features.

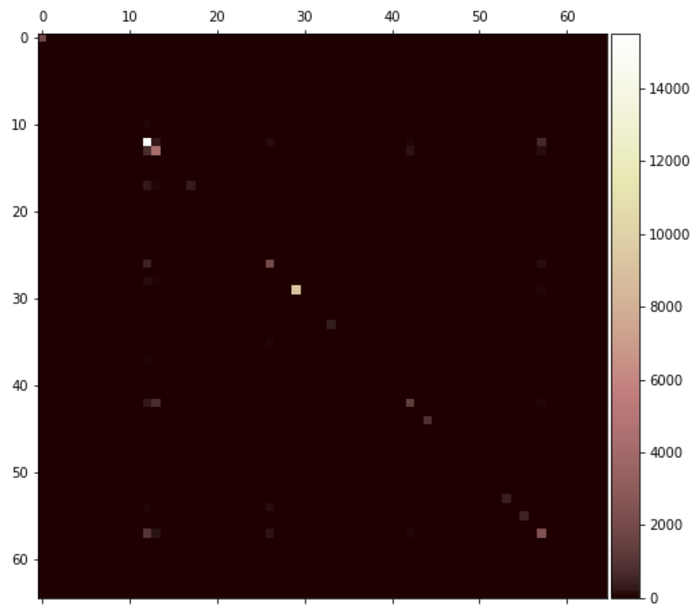
```
param_grid_1 = {"clf__penalty": ["l2", "none"],
               "clf__solver": ["newton-cg", "lbfgs", "liblinear"],
               "countvect__analyzer": ["char"],
               "countvect__ngram_range": [(1, 1), (2, 2), (3, 3)]}
param_grid_2 = {"clf__penalty": ["l2", "none"],
               "clf__solver": ["newton-cg", "lbfgs", "liblinear"]}
```

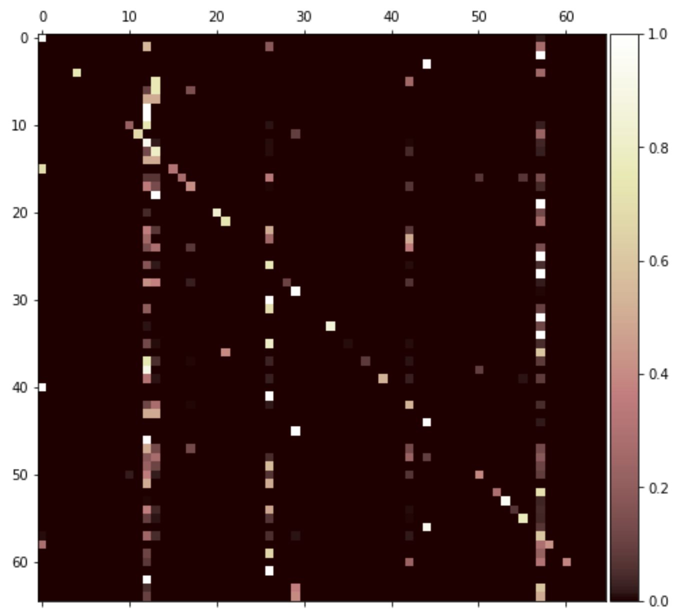
3.2 Performance

Model with param_grid_1 has the best performance with the following combination:

```
"clf__penalty": "l2",

"clf_solver": "newton-cg",

"countvect__analyzer": "char",

"countvect__ngram_range": (1, 1)
```

The accuracy on the validation dataset is 82.25%. However, with param_grid_2, the accuracy is 81.70%, which is not a big difference from that of param_grid_1. It is worth noticing that single character as features slightly improves the performance of the model.



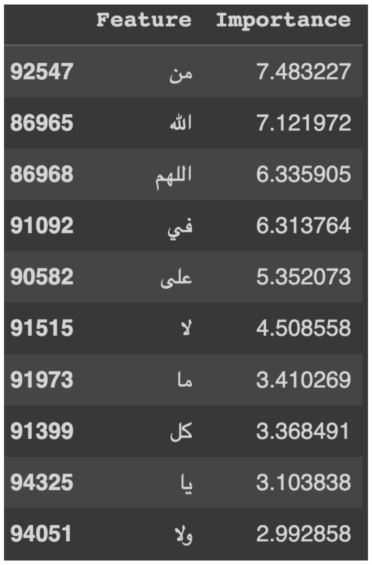


The first confusion matrix here shows that most light blocks are located in the diagonal area, which means the predicted label matches the actual label quite well. However, due to the fact that some classes of languages have few entries, and thus, it doesn't appear as the light colour in the diagonal area even if it's predicted correctly. The brighter the colour in this figure also indicates that the language has more entries in the train dataset. The brightest block is probably English.

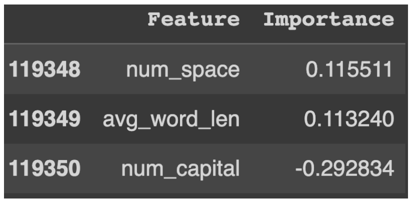
To solve this problem, the second confusion matrix is normalised by the number of entries of that language in the train dataset. The diagonal area has more light blocks than the first confusion, which means some languages with fewer entries are actually classified correctly. However, by comparing the first and second confusion matrix, we can see that the model tends to classify the languages to the languages that have more entries.

3.3 Feature Importance

Feature importance shows how important the features are to the classification results. Here, the model with param_grid_2 will be used for the illustration of feature importance.

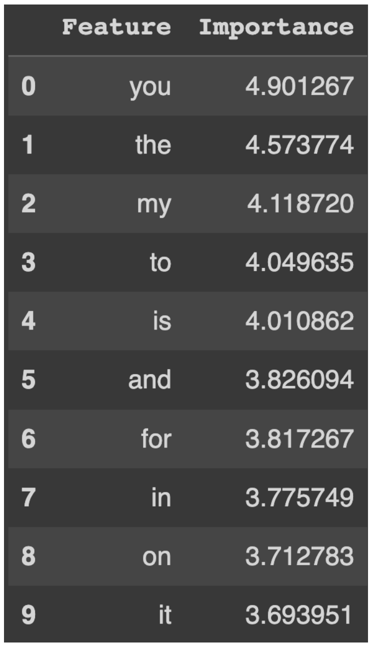


	Feature	Importance
92547	من	7.483227
86965	الله	7.121972
86968	اللهم	6.335905
91092	في	6.313764
90582	على	5.352073
91515	لا	4.508558
91973	ما	3.410269
91399	كل	3.368491
94325	يا	3.103838
94051	ولا	2.992858

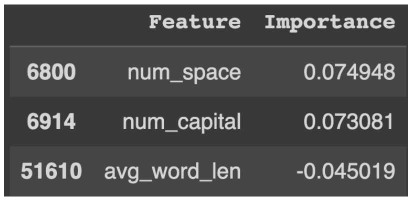


	Feature	Importance
119348	num_space	0.115511
119349	avg_word_len	0.113240
119350	num_capital	-0.292834

If all language classes are considered, there are a total of 119351 features including the features generated by CountVectorizer and the extra features. The Arabic letters have more contribution for the task. The extra features have little influence on the results. This might be due to the reason that we have limited understanding for languages other than those with English alphabets, so that the extra features came up by us are not particularly aimed for languages without English alphabets.



	Feature	Importance
0	you	4.901267
1	the	4.573774
2	my	4.118720
3	to	4.049635
4	is	4.010862
5	and	3.826094
6	for	3.817267
7	in	3.775749
8	on	3.712783
9	it	3.693951



	Feature	Importance
6800	num_space	0.074948
6914	num_capital	0.073081
51610	avg_word_len	-0.045019

On the other hand, if only English, Japanese and Spanish (the top three most entries in datasets) are considered, the feature importances are significantly different. There are a total of 68300 features. The stop word "you" is the most important feature, which proves the idea of not removing stop words correctly. Since stop words are words that appear frequently in a certain language, removing them is not a good idea. The importance of number of spaces and number of capitals also improves significantly. This is because as English speakers, we are more likely to come up with the related features to differentiate them. On the other hand, `avg_word_len` fails to be a useful feature. Although Japanese can be easily differentiated by it because the value is mostly 1, English and Spanish probably have similar word length.

4. Multilayer Perceptron (MLP)

4.1 Background

After implementing and tuning our Logistic Regression Model, we implemented a Multilayer Perceptron Classification (MLPC) algorithm to compare its performance with the Logistic Regression Model (Linear Model). In the Multilayer Perceptron algorithm, a neural network learns a mapping from the representation in the training data to the output variable we want to predict (in our case, it's the language of the tweets). The MLPC network consists of 3 or more layers of perceptrons. The data are first applied to the input layer, and then flow in the forward direction to the hidden layer(s), and to the output layer. While there is only 1 input layer and 1 output layer, the number of hidden layers and the number of neurons (single perceptrons) each layer contains can be adjusted, for example, in the process of tuning the network parameters. The high predictive capability of neural networks is due to their hierarchical or multi-layered structure. Before training, the weights of connections between neurons in the network are initialised randomly. During learning, a network receives feedback on the error in its current results, so that the weights between neurons can be adjusted to minimise the error, allowing for a more accurate prediction in the next forward pass through the network. This technique of learning the weights in a neural network is called backpropagation.

4.2 Hyperparameters and Grid Search

When a MLPC algorithm is implemented using a sklearn library, it is required to specify network hyperparameters (otherwise, the default hyperparameters are used). We tried several different combinations of hyperparameters to identify the ones that enable the best performance.

First set

```
clf = MLPClassifier(activation='relu', solver='sgd',
learning_rate='adaptive', max_iter=20, verbose=True, early_stopping=True)
```

```
Accuracy of MLPClassifier : 0.5186028853454822
```

The **activation function** in a neural network (also called the **transfer function**) represents the function that maps the weighted sum of inputs that flow into a single neuron to values between 0 and 1 (or -1 and 1, depending on the specific type of the activation function. Then, the relation between the output of the activation function and the firing threshold determines if the neuron will be activated, or not (i.e. if the output of the activation function is below the threshold, the neuron will not be activated). The 'relu' (Rectified Linear Unit) is a monotonic function that is somewhat similar to a linear function. But, it is differentiable and its derivative is also monotonic, which allows for backpropagation and computational efficiency at the same time. Its range is [0 to infinity). Currently, it is the most commonly used type of the activation function in deep learning.

The **solver** in a neural network is focused on the optimization problem of loss minimization. It represents a hyperparameter that combines the forward inference in a network with the backpropagation gradients in order to determine the appropriate parameter updates that attempt to minimise the training loss. The 'sgd' stands for Stochastic Gradient Descent (SGD) optimization algorithm. It updates the weights of the network by a linear combination of the negative gradient and the previous weight update applied.

The **learning rate** in a neural network is a hyperparameter that determines the magnitude of change in the model state in response to the estimated error each time the model weights are updated. In other words, it represents the rate at which the model learns or approximates a function to optimally map the input to the outputs. The learning rate in the MLPC algorithm can be set as 'constant' or 'adaptive', or 'invscaling', but this is only relevant when the 'sgd' solver is applied. With different solvers, the learning rate changes in a different manner, which will be discussed in the later part of this report.

'Adaptive' learning rate means that the learning rate is kept constant, at the value equal to learning_rate_init parameter (which is equal to 0.001 by default, which was preserved in the example above), as long as the training loss is decreasing. When the early stopping is on, it means that when the validation score is not increased by at least tol (set by the default to 0.0001), the learning rate will be divided by 5. The default value of the tol parameter was preserved in the example above. When the 'constant' learning rate is set for the 'sgd' solver, the learning rate stays at the learning_rate_init parameter for the entire duration of the training.

In this example, the network had a default **hidden layer size**, i.e. it had a single hidden layer with 100 neurons. The **early stopping** parameter was set to "False" (the default value).

Second set

Then, we chose the activation function 'relu', and the solver 'adam' for 20 iterations (the remaining parameters were set to default).

```
clf = MLPClassifier(activation='relu', solver='adam',
learning_rate='constant', max_iter=20)
```

```
Accuracy of MLPClassifier : 0.869969627942293
```

The 'adam' solver is a SGD-based (Stochastic Gradient Descent) optimization algorithm and it is broadly used in NLP. The main difference between classical SGD methods and 'adam' is that while in SGD a single learning rate (alpha) is preserved for all the weight updates and hence stays the same during the training, 'adam' computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradient.

This combination of parameters ('relu' activation function and 'adam' solver) yielded performance with 87.00 % (rounded to 2 decimal places).

Third set

```
clf = MLPClassifier(activation='tanh', solver='adam', max_iter=20,
verbose=True)
```

```
Accuracy of MLPClassifier : 0.8652239939255885
```

In the example above, we used the 'adam' solver again, but this time, we changed the activation function to 'tanh'. The 'tanh' stands for 'Hyperbolic tangent' activation function. Similarly to the 'relu' function, this function is differentiable and monotonic. However, its shape is sigmoid, its derivative is not monotonic, and its range is (-1 to 1). All remaining hyperparameters were kept at the default value. As we can infer looking at the obtained accuracy, there is a very tiny difference in performance between the 'adam' solver applied to our problem with a 'relu' activation function (87.00%), and with the 'tanh' activation function (86.52%). 'Relu' function appears to be slightly more suitable.

Fourth set

```
clf = MLPClassifier(activation='relu', solver='adam', max_iter=20,
verbose=True, hidden_layer_sizes = (45,2,11))
```

```
Accuracy of MLPClassifier : 0.5263857251328777
```

In the next trial, we used the ‘adam’ solver with ‘relu’ activation function again, but this time we changed the default **hidden layers arrangement** to (45, 2, 11). This means that our network had 3 hidden layers, having 45, 2, and 11 neurons, respectively. As we can infer from the MLPC accuracy, this change significantly decreases the model performance, hence, such architecture is not suitable for our classification problem.

Fifth set

Prior to repeating the training with the fifth set of hyperparameters, we applied the GridSearchCV algorithm to identify the most appropriate combination from a predefined parameter space. Followingly, the training was repeated with the set of hyperparameters identified to be the most suitable via GridSearchCV. The following parameter space was defined:

```
parameter_space = {

    'hidden_layer_sizes': [(10,30,10),(20,)],

    'activation': ['tanh', 'relu'],

    'solver': ['sgd', 'adam'],

    'momentum': np.arange(0.1, 1.1, 0.1),

    'alpha': [0.1, 0.05],

    'learning_rate': ['constant','adaptive'],

}
```

It contains 2 new hyperparameters, which were previously kept at the default values, and hence were not investigated further. The first one, **momentum**, is used only when ‘sgd’ solver is chosen and refers to the parameter which removes the random convergence which may occur due to the stochastic (random) nature of the ‘sgd’ optimizer. Hence, ‘momentum’ improves the performance of the model with ‘sgd’ solver.

Alpha hyperparameter controls the magnitude of the L2 regularisation term. The regularisation term is divided by the sample size when it is added to the loss. The function of the regularisation term is to lower the complexity of a neural network model during training, in order to prevent overfitting. L2 regularisation is the most common type of a regularisation technique.

The GridSearchCV identified the following set of hyperparameters as the most suitable[2]:

```
{'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (20,),
'learning_rate': 'constant', 'momentum': 0.8, 'solver': 'adam'}
```

```
clf = MLPClassifier(activation='relu', solver='adam', alpha=0.0001,
learning_rate='constant', momentum = 0.8,  max_iter=20, verbose=True)
```

```
Accuracy of MLPClassifier : 0.8669324221716022
```

Sixth set

Before applying another set of parameters, we carried out another GridSearch, to focus more on the hidden layer sizes and arrangements, as well as activation functions.

The following parameter space was investigated:

```
parameter_space = {

   'hidden_layer_sizes': [(10,30,10),(50,50,50),(50,100,50)],

   'activation': ['tanh', 'relu'],

}
```

GridSearchCV chose the following parameters:

```
Best parameters found:

 {'activation': 'tanh', 'hidden_layer_sizes': (50, 100, 50)}
```

Thus, the training was repeated with these hyperparameters.

```
clf = MLPClassifier(activation='tanh', solver='adam', alpha=0.0001,
max_iter=20, hidden_layer_sizes = (50, 100, 50), verbose=True)
```

```
Accuracy of MLPClassifier : 0.863705391040243
```

[2] Actually, the alpha parameter of 0.0001 was applied in this trial instead of the recommended 0.05. That's because alpha=0.0001 was not included in the parameter space in this GridSearch run, but it will be revealed to be more suitable than 0.05 in the next GridSearch run.

Seventh set

Another GridSearch was carried out first, this time it focused on the hidden layer arrangement, size, and the alpha parameter.

```
parameter_space = {
    'hidden_layer_sizes': [(50,100,50), (100,200,100), (50,100,100,50)],
    'alpha': [0.0001, 0.001, 0.1],
}
Best parameters found:
 {'alpha': 0.0001, 'hidden_layer_sizes': (100, 200, 100)}
```

These parameters were used to train the network for 50 iterations:

```
clf = MLPClassifier(activation='relu', solver='adam', alpha=0.0001,
max_iter=50, hidden_layer_sizes = (100, 200, 100), verbose=True)
```

```
Accuracy of MLPClassifier : 0.8602885345482156
```

From the performance score we can infer that there was no improvement if we compare the accuracy to the one obtained with the best set of hyperparameters used so far (**second set**).

4.3 Best Performance Model

As the **second set** of hyperparameters was demonstrated to be the most suitable, we repeated the training with this set, this time carrying out 100 iterations. The final accuracy was on the validation dataset, y_val:

```
Accuracy of MLPClassifier : 0.8714882308276386
```

Subsequently, this model was tested on the test data set, y_test (previously unseen data). The performance on the test data set was as follows:

```
Accuracy of MLPClassifier : 0.8628765060240964
```

5. Conclusion

We have developed a linear classification model (logistic regression), and a multilayer perceptron classification model in order to classify the language of tweets. The final linear model was reported to identify the language of tweets with 82.48% accuracy, tested on previously unseen data. The final multilayer perceptron model could classify the language of tweets with 86.29% accuracy, tested on previously unseen data. Therefore, we conclude that the MLPC model gave us more accurate final predictions than the linear model. This is the verdict we expected, as neural networks generally outperform linear models, because they have more degrees of freedom, therefore allowing for more flexibility. However, the linear model could potentially still be chosen instead of the MLPC model, because although it has a worse prediction accuracy, it has two significant advantages over the MLPC model. First, the MLPC, as all neural networks, is a black-box model, meaning that it does not allow for the interpretation of the model parameters, which affects the interpretability and 'fairness' of the model predictions. Second, the training of the MLPC model is more computationally expensive, hence, it takes a lot more time and consumes more energy. Therefore, to make a decision between a linear model and a neural network model, we have to consider whether higher prediction accuracy is desired, or better model explainability and lower computational cost are preferred.

References

The following resources were consulted and facilitated the project completion and write-up:

Austin (2019). *Classify Sentences via a Multilayer Perceptron (MLP)*. [online] Austin G. Walters. Available at: https://austingwalters.com/classify-sentences-via-a-multilayer-perceptron-mlp/.

Baheti, P. (2022). *Activation Functions in Neural Networks [12 Types & Use Cases]*. [online] www.v7labs.com. Available at: https://www.v7labs.com/blog/neural-networks-activation-functions#:~:text=Try%20V7%20Now- [Accessed 16 Oct. 2022].

Bento, C. (2021). *Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis*. [online] Medium. Available at: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141#:~:text=Threshold%20T%20represents%20the%20activation [Accessed 16 Oct. 2022].

Brownlee, J. (2016). *Crash Course On Multi-Layer Perceptron Neural Networks*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/neural-networks-crash-course/.

Deepchecks. (n.d.). *What is Learning Rate in Machine Learning?* [online] Available at: https://deepchecks.com/glossary/learning-rate-in-machine-learning/#:~:text=The%20learning%20rate%2C%20denoted%20by [Accessed 16 Oct. 2022].

GeeksforGeeks. (2022). *Difference between Multilayer Perceptron and Linear Regression*. [online] Available at: https://www.geeksforgeeks.org/difference-between-multilayer-perceptron-and-linear-regression/.

Jason Brownlee (2017). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

Jia, Y. (n.d.). *Caffe | Solver / Model Optimization*. [online] caffe.berkeleyvision.org. Available at: https://caffe.berkeleyvision.org/tutorial/solver.html [Accessed 16 Oct. 2022].

Oppermann, A. (2020). *Regularization in Deep Learning — L1, L2, and Dropout*. [online] Medium. Available at: https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036.

Panjeh (2020). *scikit learn hyperparameter optimization for MLPClassifier*. [online] Medium. Available at: https://panjeh.medium.com/scikit-learn-hyperparameter-optimization-for-mlpclassifier-4d670413042b.

Scikit-learn.org. (2010). *sklearn.neural_network.MLPClassifier — scikit-learn 0.20.3 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

SHARMA, S. (2017). *Activation Functions in Neural Networks*. [online] Medium. Available at: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

Srinivasan, A.V. (2019). *Stochastic Gradient Descent — Clearly Explained !! - Towards Data Science*. [online] Medium. Available at: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31.

Preethi (2022).*What is Logistic Regression?* [online] Medium. Available at: https://medium.com/@tpreethi19/what-is-logistic-regression-4251709634bb

Himanshu Chandra (2020). *Pipelines & Custom Transformers in scikit-learn: The step-by-step guide (with Python code)* [online] Medium. Available at:

https://towardsdatascience.com/pipelines-custom-transformers-in-scikit-learn-the-step-by-step-guide-with-python-code-4a7d9b068156