**Task1:**

Question1: 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?

**Ans: For example the language like 'mai' and 'tha', they are not alphabet based which needs a strong pipeline to handle and vectorize them. Also, there are some language both based on alphabet and even some highly similar words.**

Question2: How many instances per label are there in the training and test set? Do you think this is a balanced dataset? Do you think the train/test split is appropriate? If not, please rearrange the data in a more appropriate way.

**Ans:**

**500 instance per label are there in the training and test set. I think it's a balanced dataset per class. But the size of training set is equal to test set, this split is not right since test set is usually smaller than training set.**

**Code:**

**test\_df\_to\_train, test\_df\_remaining = train\_test\_split(test\_df, test\_size=0.5, random\_state=42, stratify=test\_df['label']) #fix random seed**

**train\_df = pd.concat([train\_df, test\_df\_to\_train], ignore\_index=True)**

**test\_df = test\_df\_remaining**

Question3: 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)

**Code:**

**fixed\_languages = ['eng', 'deu', 'nld', 'dan', 'swe', 'nor']**

**self\_selected\_languages = ['est', 'tha', 'guj', 'tam', 'vie', 'lat', 'urd', 'por', 'fra', 'rus', 'ara', 'heb', 'hin', 'jpn', 'kor', 'zho', 'spa', 'ita', 'tur', 'ell']**

**final\_language\_list = fixed\_languages + self\_selected\_languages**

**sub\_traindf = train\_df[train\_df['label'].isin(final\_language\_list)]**

**sub\_testdf = test\_df[test\_df['label'].isin(final\_language\_list)]**

Question4: 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?

**Code:**

**from sklearn.preprocessing import LabelEncoder**

**le\_fitted = LabelEncoder().fit(sub\_traindf['label'])**

y\_train\_dev, y\_test = le\_fitted.transform(sub\_traindf['label']), le\_fitted.transform(sub\_testdf['label'])

Question5: What other features could you use to determine the language? Please include additional linguistic

features to your machine learning model for this task

**Ans:**

**CountVectorizer** transforms a collection of text documents into a matrix of token counts. It tokenizes the text documents and builds a vocabulary of known words, encoding new documents using that vocabulary. The emphasis is on the frequency of words, making it suitable for applications like spam detection where occurrence is paramount.

Conversely, **TF-IDF Vectorizer** involves a more nuanced process, taking into account not just the occurrence (Term Frequency, TF) but also the distribution (Inverse Document Frequency, IDF) of a word across the entire corpus, aiming to highlight words that are more unique to a particular document. This vectorizer is particularly useful in applications like topic modeling where the semantic meaning of words is crucial.

**I will add TF-IDF vectorizer for this task.**

Question5: hyperparameter combination

**Ans:**

图表, 散点图

描述已自动生成

Question6: best hyperparameter combination

{'classifier\_\_penalty': 'l2', 'classifier\_\_solver': 'liblinear', 'vectorizer\_\_ngram\_range': (1, 1)}

Question7: What is the advantage of grid search cross-validation?

Grid Search Cross-Validation provides a systematic, exhaustive search over specified parameter values for an estimator, ensuring the identification of the most optimal parameters, and thereby enhancing the predictive accuracy of the model. Additionally, it minimizes overfitting by utilizing cross-validation to evaluate the performance of each parameter combination.

Question8: Use a confusion matrix to do your error analysis

and summarize your answers in your report.

图表, 散点图

描述已自动生成

Question9: Generate a feature importance table for the top ten features(please have the features named) for the languages English, Swedish, Norwegian, and Japanese. What is

more important, extra features or the outputs of the vectorizer, discuss.

1. **English**

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. **Swedish**

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. **Norwegian**

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. **Japanese**

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

Question10: How does the ablation affect the performance of the classifier?

The result for two best-distinguished language with all chars/500 chars/100 chars refit with best model remains the same, which means they are easy to be classified and do not affected by number of chars much.

**Part2:**

Question1: Play around with 5 different sets of hyperparameters including layer sizes, activation functions, solvers, early stopping, vectorizer parameters, and report your best hyperparameter combination Do you achieve higher performance? Why/why not?

The best params are:

'callbacks\_\_earlystopping\_\_patience': 10,

'module\_\_nonlin': <function tanh at 0x7ffa9858af80>, 'module\_\_num\_units': 800,

'optimizer': <class 'torch.optim.sgd.SGD'>},

'vectorizer\_max\_features': 1000

With the parameters above I achieved the best performance. The patience number 10 is more suitable than 5 because sometimes the loss would stay on a plateau and does not drop for few epochs. With number 5 it is easier to stop over early. The SGD optimizer is practically a better choice with appropriate learning rate setting. The vectorizer\_max\_features is the most important factor here, with only 100 max\_features is hard to train a good classifier above 80.