Case study 1:

1. Approach:

Due to lack of time an unconventional approach was taken by testing some strategies on scripts from external sources. The scripts were chosen from the academia, competitions and recommendations on forums such as Quora. These jupyter notebooks were adapted for the data and the results compared.

The following steps were taken (external scripts used*):

- Literature study (Books, academic papers, other internet sources)
- Tests:
  - Exploratory data analysis (EDA)
  - Shallow* and deep models*
  - Feature engineering*
  - Model selection*
- Final approach
  - Initial model
  - Grid search with cross validation to optimise model parameters
- Application

During the EDA the data was manually scanned as a start and immediately missing values, capital letters, punctuation marks, variable length samples and severe class imbalance were seen. The data set size is also small. All of these aspects can influence the machine learning effectiveness and will be discussed later.

Shallow models and feature engineering were tested by modifying the following scripts and using ideas from the following blog post:
https://www.kaggle.com/zarajamshaid/language-identification-machine-learning
https://github.com/danielv775/Natural-Language-Identification-Graduate-Project
https://towardsdatascience.com/sentiment-analysis-of-tweets-using-multinomial-naive-bayes-1009ed24276bjuju

Deep models were tested changing:
https://github.com/floydhub/language-identification-template

A script was also developed incorporating a Long Short Term Memory Cell Neural Network to investigate if context can improve the results.

Based on the outcome of this initial exploratory strategy analysis, the choices for the final approach were made, optimised and applied.

2. Libraries used

numpy
NumPy is the fundamental package for array computing with Python.

Pandas
pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive.

sklearn
scikit-learn is a Python module for machine learning built on top of SciPy. SciPy is open-source software for mathematics, science, and engineering. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation.

re
Alternative regular expression module, to replace re

strings
Python strings for humans

matplotlib
Python plotting package

scikitplot
An intuitive library to add plotting functionality to scikit-learn objects

3. Tests

3.1. Exploratory data analysis (EDA)

Refer to eda_data_preprocessing.ipynb

The data was analysed in a jupyter notebook. The difference in count between the ‘text’ and ‘language’ column of the data indicated missing values. The descrepency between the count and unique values of the ‘text’ column indicated duplicates. These problematic rows were removed.

Counting the individual class samples revealed a massive imbalance. The ratio of English samples to Afrikaans- and Nederlands rows are 3:1 and 30:1 respectively. The total data set size of around 3000 is already small, therefore the small count of Nederlands samples suggest that the simplest shallow model would be the most appropriate.

Histograms of sample lengths indicated that the English and Nederlands samples were similar in length, but the Afrikaans samples were about twice as long. The data set could therefore be balanced better by splitting the longer Afrikaans samples. This was not pursued at this stage.

Next text normalisation was applied:

- all the punctuation was removed
- the capital letters were changed to lower case
- numbers were removed
- white spaced were removed

Abbreviations were not picked up with a quick notepad search, thus not expanded. Stop words, sparse terms and particular words were not removed as per some other natural language processing (NLP) tasks as these are language specific.

After normalising 17 more duplicates were picked up and removed. The total data set is now 2733. With a 40:1 English to Nederlands ratio and only 67 Nederlands samples.

Due to this very small sample of Nederlands, it was decided to split 10% of the cleaned set as a hold-out set. An unprocessed split could result in an even higher imbalance. The set was shuffled before the split to improve the distribution of the classes. The cleaned data sets were saved to CSV files to be used in other notebooks. An unprocessed hold-out set was also saved to check the total application pipe-line and the difference in class imbalance for interest.

3.2 Shallow and deep models

Refer to Language Identification - Machine Learning_pdt_rev0.ipynb (only an exploratory script and not meant for submission)

The two main choices for the path ahead are using an algorithm with a deep hierarchical structure or a flat approach. From the non-deep learning family a Random Forest Classifier was tested against a Linear Support Vector Classifier, Logistic Regression Classifier and multinomial Naive Bayes classifier.

Since the challenge is to classify the Nederlands samples correctly, the normalised true positives of the Nederlands samples were used as an early indication of model performance. The Random Forest did the worst, enforcing the early intuition about the small Nederlands sample requiring a simplistic approach.

Refer to LSTM.ipynb (only an exploratory script and not meant for submission)

An LSTM with hyperparameters that were successful on a spam detection data set were experimented with. Again the performance on the Nederlands samples were inferior to the shallow models by a margin, but butter than the Random Forest.

3.3 Feature Engineering

Refer to the 3 notebooks mentioned above

The main approaches experimented with were an embedding layer for the LSTM, Scikit-Learn's CountVectorizer and Scikit-Learn's TfidfVectorizer.

For the CountVectorizer using a uni gram was superior, but testing the same models with the TfidfVectorizer revealed better results.

3.4. Model selection

Refer to TfidfVectorizer_3_models.ipynb (only an exploratory script and not meant for submission)

From the exploratory analysis it was decided to use a shallow model with the TfidfVectorizer for feature extraction. Again the true positive rates of the Nederlands samples were used in combination with F1 score to compare model performance due to the high class imbalance and few Nederlands samples.

The multinomial Naive Bayes classifier proved superior inline with the idea that the simplest approach would fit the small Nederlands sample set the best.

4. Final approach

4.1 Initial model

Refer to pipeline.ipynb

The processed CSV data file were ingested and the ‘language’ column converted to numeric labels to be suitable for the subsequent processing. Up sampling of the minority classes were used to balance the training data set. This will be discussed later.

The default settings were used for the multinomial Naive Bayes classifier of Scikit-Learn. The parameters of the TfidfVectorizer were analyser=’char’ and ngram_range=(1,3). These settings delivered good results in the exploratory analysis.

The TfidfVectorizer converts raw text samples to a matrix of TF-IDF features. It is equivalent to CountVectorizer followed by a TfidfTransformer.

The CountVectorizer converts the text samples to a matrix of token counts. The weighting and normalisation is done by the TfidfTransformer.

TF-IDF stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

The basic idea of the Naive Bayes classifier is to find the probabilities of classes assigned to texts by using the joint probabilities of words and classes. This is done with Bayes’ theorem. The multinomial version is suitable for discrete values such as word counts and is the baseline approach for text classification.

4.1.1 Metrics and performance

The first metric used was accuracy on the 33% test split. The value of 0.957 is high, but could be misleading due to the class imbalance. A better indication of accuracy is using cross-validation where the data set is divided into a number of folds, each with a training and test set. The value was even higher at 0.965.

These are good initial indications, but for multi-class and especially imbalanced data sets, the classification report contains more valuable information. Over-all the values of precision and recall are balanced on each class and therefore F1 scores are also high. Assuming each class is equally important, this is a good sign. From now on F1 score will be used as the deciding metric and the value for Nederlands as the most important, since the performance on this class is the lowest since it contains the least amount of samples and the samples are close to Afrikaans.

The normalised confusion matrix was the best indication of classifier performance. The true positive rate of 0.77 for Nederlands is high considering the small amount of samples, only 5 out of 22 were misclassified as Arfikaans, which makes sense since Afrikaans is close to Nederlands.

A visual confirmation of the performance showed that the classifier worked correctly.

For imbalanced classes the receiver operating characteristic (ROC) curve is a good metric. This is more common for binary classification, but a multi-class implementation is possible and was used. These area under the curve AUC is close to 1 which is very good. 1 is perfect.

The final model was trained on the combined train and test set. An evaluation of metrics with cross validated results were done, although the high unsampling of the Nederlands samples would boosted the performance on this class all Nederlands samples were classified correctly. The model was saved to disk to be applied to the hold-out set.

4.2 Grid search with cross validation to optimise model parameters

Refer to pipeline_gridsearch.ipynb

It is good machine learning practice to test the hyper parameter space for improvements. Therefore the same machine learning pipeline described in under 4.1 was adapted with a grid search function.

The following parameters were varied:

- TfidfVectorizer
  - n_gram_range
    - The upper and lower boundary of the range of n-values for different n-grams to be extracted, e.g. (1, 3) means unigrams and trigrams. The default is (1,1).
  - use_idf
    - Enable inverse-document-frequency reweighing. Default is True
  - norm
    - Unit norm, default is l2.
- multinomial Naive Bayes classifier
  - alpha
    - Additive (Laplace/Lidstone) smoothing parameter. The default 1 is called Laplace smoothing, while <1 is called Lidstone smoothing. A value of 0 is no smoothing.

Values were chosen based on examples found during research e.g. the Blog post of Smetanin referenced earlier. The following optinum values were found:

- n_gram_range = (1, 3)
- use_idf = False
- norm = l2
- alpha = 0.01

4.2.1 Metrics and performance

The same metrics were evaluated as before. Looking at the performance of the classifier on the Nederlands samples, the precision increased, with a drop in recall and overall reduction of F1 score. The overall effect was that the true positive rate decreased from 0.77 to 0.5 compared to the initial model. It could be that the metric used for optimisation (accuracy by default) has to change due to the class imbalance. This was not pursued.

5. Testing

Refer to application.ipynb

During the application phase the performance of the initial model was tested first on the processed holdout set. Due to the shuffling of the rows when the training and hold-out data sets were generated data leakage would occur if the unprocessed hold-out set is used for out of sample performance evaluation.

The model was loaded and predictions analysed.

Focusing on the normalised true positive rate of Nederlands at a value of 1, the classifier performs well.

Refer to application_unprocessed.ipynb

To test the machine learning pipeline, the raw hold-out data set were ingested and predictions analysed.

The model performed the best on this data set, with only 5 samples misclassified, probably due to the data leakage.

Interestingly the unprocessed hold-out set contains half the amount of Nederlands samples. Thus although it tests the full pipeline, the performance on the Nederlands set is better evaluated with the processed set.

Refer to application_gridsearch.ipynb

As expected the normalised true positive rate of Nederlands at 0.88 is lower for the optimised model. Therefore the initial model should generalise better and be used for deployment until the class imbalance can be handled better during optimisation.

6. Bonus Questions

1. Logistic Regression or Linear Support vector classifier as described and tested. Questions 2 to 6 are answered in Report2 for case study 2.