Machine Learning Approaches to Fake News Detection

P470-P775-P187-P453

March 23, 2022

Contents

1 Introduction

In this assignment we build a classifier to predict whether an article comes from a reliable source. We aim to fit a classifier with maximal predictive accuracy, whilst preserving interpretability. The training set contains information from 1000 documents with a vocabulary of 12,246 words, presented in a bag-of-words format which specifies the counts of each word used in each document.

1.1 Exploratory Data Analysis

The split between fake and reliable news articles is displayed in Figure 1, showing that the data is well balanced between the two classes, hence, no balancing is needed.

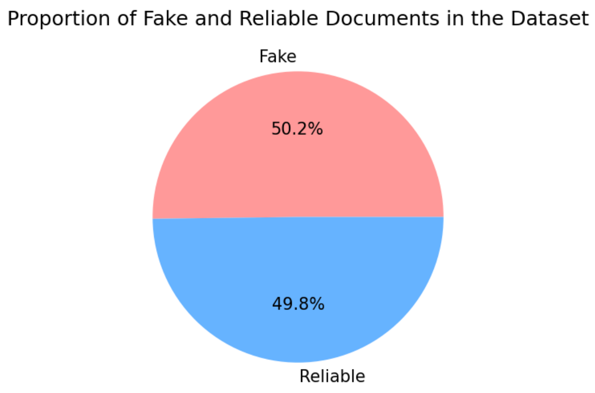

Figure 1: Pie chart to show class balance

Principal Component Analysis ('PCA') was applied for dimensionality reduction by obtaining a new basis of the dataset which maximises the explained variance of components. The number of times that a word occurs has no upper bound other than the number of words in an article, hence, standardisation was applied to obtain uniform measurement scales across words. Standardisation prevents the measurement scale of bagged word counts from affecting the computed principal components.

PCA was applied to both a standardised version of the original training dataset and a tf-idf transformed version to obtain principal components which explain 95% of variation. Based on the standardised data, 759 principal components are obtained while the tf-idf data yields 821 components. The large number of components obtained suggests that a small set of words may not be sufficient to identify fake news. This implies that there is a lack of words which explain a high proportion of variability in the data. Figure 2 shows that the first two principal components are not sufficient to separate reliable and fake data as they account for only 3.71% of the total variability.

For some classifiers we used tf-idf, a transformation applied to bag-of-words data with the aim of improving the weighting of important features in the data. It is proportional to the number of times a word appears in a document and is offset by the number of documents in which the word appears, adjusting for words that commonly appear.

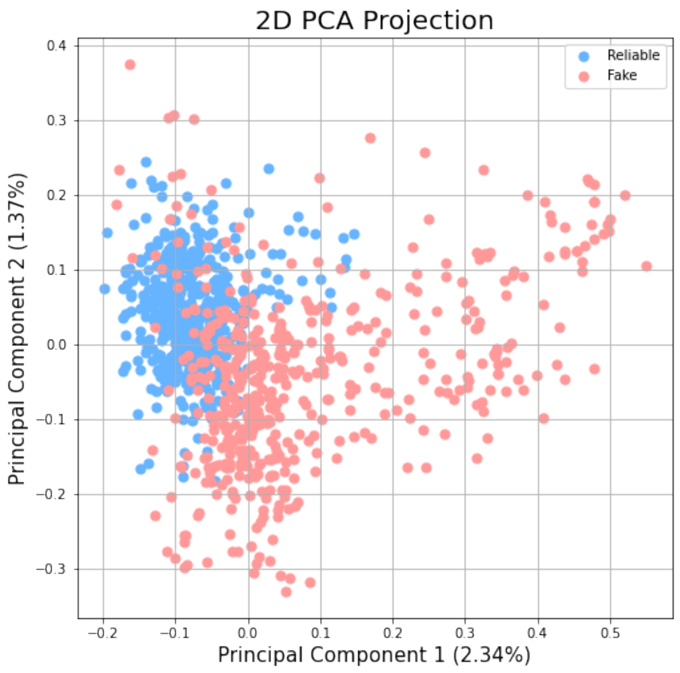

Figure 2: Projection of the tf-idf data onto the first two principal components

1.2 Cross Validation

We used 5-fold Cross Validation ('CV') to estimate generalisation accuracy for our models, which we now refer to as CV accuracy. Optimally, we would test the final model on unseen data not included in the model selection procedure, but we deem the dataset insufficiently large for such defined splits. We chose a 5-fold split since it is considered good practice and has less computational cost than higher folds [1].

In the discussion, we obtained our cross-validation confidence intervals by following the naïve methods given in [2]. In particular, we split the training data, $\mathcal{I} = \{x_1, \ldots, x_{1000}\}$, into 5 equal parts $\mathcal{I}_k$, $k = 1, \ldots, 5$. Denote the model fitted excluding $\mathcal{I}_k$ with $\hat{f}^{-\mathcal{I}_k}(x)$. For each $k = 1, \ldots, 5$, and all $x_i \in \mathcal{I}_k$ we define,

$$a_i := 1 - L(\hat{f}^{-\mathcal{I}_k}(x_i), y_i),$$

where L is the $0-1$ loss. Define the standard error of the mean

$$\widehat{SE} := \frac{1}{\sqrt{n}} \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (a_i - \bar{a})},$$

where,

$$\bar{a} := \frac{1}{n} \sum_{i=1}^{n} a_i.$$

Then the 95% confidence interval is the following assuming normality,

$$(\bar{a} - z_{0.975} \cdot \widehat{SE},\, \bar{a} + z_{0.975} \cdot \widehat{SE}).$$

This approach may result in optimistic bounds due to underestimated standard error. We also re-estimated our CV accuracies to avoid biases from optimising the hyperparameters.

1.3 Hyperparameter Tuning

We used Bayesian Optimisation for hyperparameter tuning, intending to maximise CV accuracy. Alternatively, we could do a grid search, but this is inefficient over a wide parameter range since it does not use knowledge from previous iterations to inform the next evaluation. Bayesian Optimisation models the target function with a probabilistic model known as a surrogate model. In general, we want to evaluate points where we have high uncertainty in the value of the target function and where we believe the target function is large relative to our current best evaluation. This is known as the exploration-exploitation trade-off, which is managed by an acquisition function. We used a Gaussian Process for our surrogate model as these generally lead to effective and cheap acquisition functions. However, Bayesian Optimisation of the CV accuracy can lead to different hyperparameter tuning results on separate runs due to the random train-test splits used.

2 Methods and Results

2.1 Logistic Regression

We began by examining Logistic Regression ('LR') for classifying fake news. The dataset only has 1000 observations, making it unreasonable to fit a model with all 12246 features. While we could fit this model, we may obtain many solutions each with an equally good fit for the data. Overfitting to the training data is also a concern with a high number of covariates.

To combat overfitting, L2-regularised models were fitted to the original, standardised and tf-idf data. The scaling parameter for regularisation was chosen using Bayesian Optimisation, resulting in up to 475 features being excluded from the original model. However, there were still far more features than observations.

Alternatively, we fit a LR model using the principal components obtained on each transformed dataset via PCA. Fitting a LR model on the principal components of the standardised data gave 100% training accuracy and 84.7% CV accuracy, whilst a LR model on the principal components of the tf-idf data gave 100% training accuracy and 95.1% CV accuracy. This difference in performance implies that tf-idf better highlights the variability in the training data, which is useful for the identification of fake news. The principal component tf-idf LR model is from now on referred to as the LR model.

The linearity of the LR model and the ineffective selection of features without PCA makes it a poor fit for this task. However, a CV accuracy of 95.1% is formidable and shows that the full set of 12246 features is not required to accurately classify fake news.

2.2 Neural Networks

Neural Networks ('NNs') are powerful and flexible models that are preferred when interpretation is not a key aim of modelling. Here, we fit NNs to improve accuracy over LR. NNs are prone to overfitting with high-dimensional inputs, hence, L2-regularisation was used to minimise this risk. Likewise, NNs have non-convex optimisation so it can be difficult to tune the hyperparameters to find a near-optimal fit.

The NNs were fit with both the regular and tf-idf transformed datasets, and the best models were attained with tf-idf data. Using Bayesian Optimisation we searched over sets of activation functions, optimisers, initial learning rates for ADAM, L2 regularisers, and network structures, described in Table 1 to determine optimal parameters. To reduce the search range of the hyperparameters we only considered optimisers ADAM and L-BFGS that don't have a tunable step size parameter.

Parameter	**Search range**
L2 regularisation	$10^{-4} - 100$
Optimiser	ADAM, L-BFGS
Initial learning rate	$10^{-4} - 0.1$
Activation function	sigmoid, tanh and ReLu
Neurons in first layer	3-500
Neurons in second layer	0-100
Neurons in third layer	0-50

Table 1: Search ranges for NN hyperparameters.

The best performing classifiers were generally shallow networks with activation function $\tanh$, optimised with L-BFGS. The best CV accuracy attained was 96.0% with a network of only a single hidden layer and five nodes fitted on the transformed data. However, several setups gave similar results, as shown in Table 2. A similar procedure was implemented on the principal components obtained using the tf-idf data as for LR, however, this showed no improvement. PCA potentially disregards features which explain significant variability in the response. Overall, NNs provided strong CV accuracy, but allowed for endless tuning and were not at all interpretable.

CV Accuracy (%)	**Network structure**	**L2 regularisation**	**Optimiser**	**Activation**
96.0	[5]	0.0256	L-BFGS	$\tanh$
95.9	[29, 7]	0.122	L-BFGS	$\tanh$
95.9	[20, 16]	0.752	L-BFGS	$\tanh$

Table 2: Neural network models with the highest CV accuracy

2.3 Trees and Random Forests

For increased interpretability we investigated Decision Trees, which can be visualised intuitively. To maintain continuity with Random Forests' ('RFs') optimisation, we parameterised the Decision Trees by maximum depth. A cross-validated search gave an optimal depth of 4, resulting in 95.3% training accuracy and 93.9% CV accuracy. Shown below is the decision tree and the Gini importances of the 7 deciding features. The Gini importance is defined as the normalised total reduction of the Gini criterion by a feature [3]. With this classifier, 'image', 'images' and 'hillary' are the most influential words for identifying fake news.

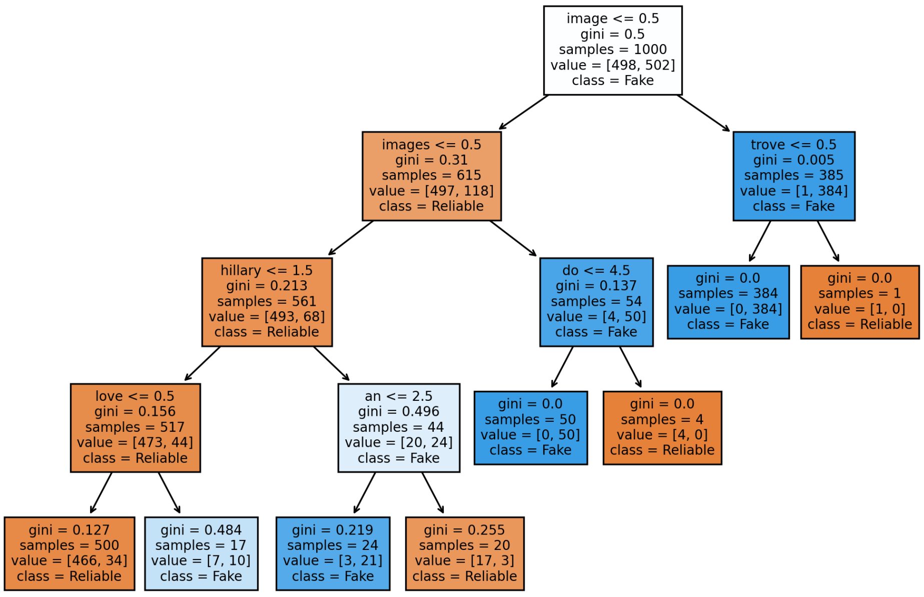

Figure 3: Graphical representation of decision tree

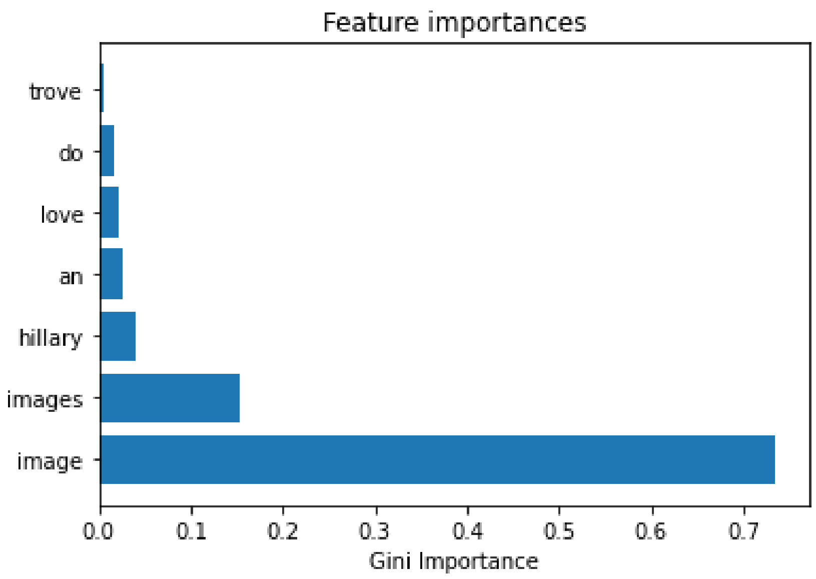

Figure 4: Importances for 7 deciding features

However, trees are generally unstable, where small changes in the dataset can notably alter the model. Instead, we considered RFs, which classify based on an ensemble of trees.

RFs consider a fixed number of randomly chosen features at each split point. By default, this was set as $\sqrt{p}$, which we used as the model was insensitive to this choice. Next, starting with 500 trees, we used out-of-bag ('OOB') test accuracy estimation to find their optimal maximum depth. This gave a depth of 38, resulting in 100% training accuracy and 96.2% OOB score. OOB was efficient, but for comparability to the other classifiers we also checked consistency with CV accuracy, which gave 95.5%. This model cannot be presented as a tree, but we can still display the feature importances. Shown in Figure 5 are the importances for the 20 most key features. From before, 'image' and 'images' are still influential, whilst 'hillary' is not. A far wider range of features are now meaningful too.

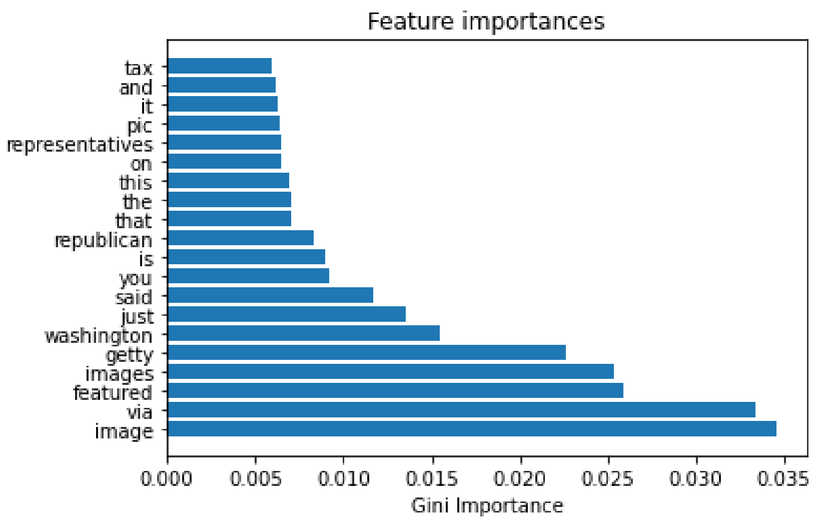

Figure 5: Importances for top 20 key features of RF

Figure 6 also shows that a range of depths achieve an OOB score around 96%. Deeper trees are more variable and computationally expensive, so we are interested in reducing depth without compromising accuracy. Similarly, using fewer trees may make fitting the model quicker without greatly reducing predictive performance. Bayesian Optimisation found 370 trees and maximum depth of 27 to be optimal, resulting in 95.8% CV accuracy. Achieving a roughly optimal predictive performance with RFs is straightforward, but truly optimising the parameters is not since a range of combinations of trees and depths give similar accuracy and their fit to the data changes each time they are run.

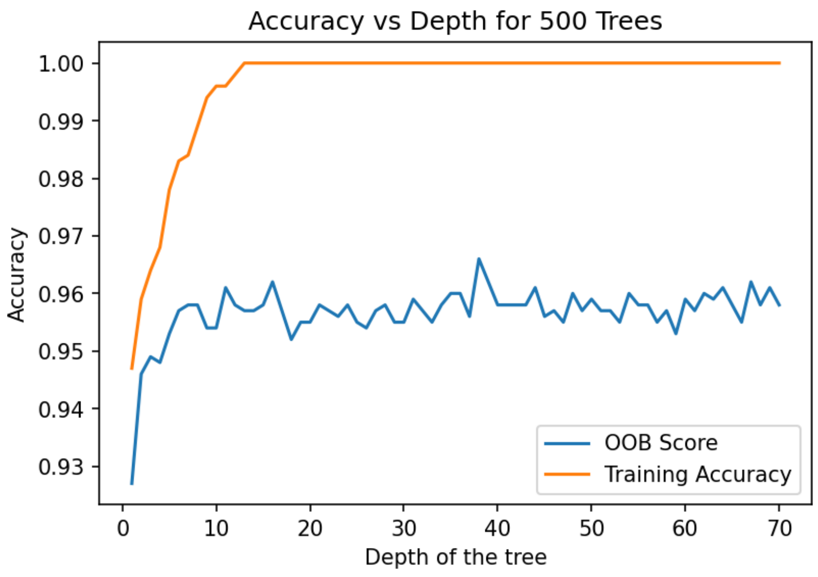

Figure 6: Training and generalisation accuracies for varying depths

2.4 Boosting

To potentially further improve on RFs' CV accuracy we investigated Boosting, an ensemble learning approach which combines a set of weak learners into a strong learner to reduce bias. The weak learners are added sequentially, such that new learners build on previous learners' errors. Several algorithms were considered, including Adaptive Boosting ('AdaBoost'), Gradient Boosting ('GB'), Extreme Gradient Boosting ('XGBoost') and of particular interest Light Gradient Boosting Machines ('LGBM'). LGBM is well suited for our sparse dataset; the algorithm bundles together exclusive features – features which never take nonzero values simultaneously – to increase efficiency [4]. In our case, LGBM bundles together words which never appear in the same document. Table 3 provides a comparison of the tuned models, with LGBM the best performing in both accuracy and runtime.

Classifier	**Training Accuracy (%)**	**CV Accuracy (%)**	**Runtime (s)**
AdaBoost	100	95.7	89.03
GB	100	96.2	207.75
XGBoost	100	95.6	127.53
LGBM	100	96.7	9.55

Table 3: Optimal Boosting classifier comparisons for accuracy and computational cost

We used Decision Trees as our set of weak learners for LGBM, though LGBM uses leaf-wise tree growth instead of depth-wise tree growth. The distinction between the two is shown in Figures 7 and 8. LGBM optimizes accuracy and runtime by using this leaf-wise tree growth method and employing a histogram-based algorithm, which splits the continuous variables into different buckets.

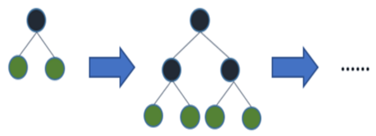

Figure 7: The depth-wise tree growth method adopted by most decision tree learning algorithms [5]

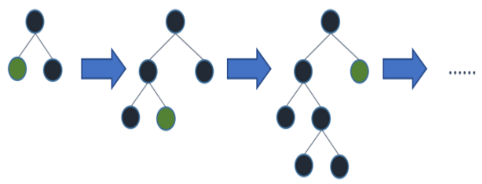

Figure 8: The leaf-wise tree growth method used by LGBM, which tends to achieve lower loss than level-wise algorithms [5]

LGBM offers many hyperparameters to tune to achieve the best fit [6]. For this report, we focused on the key hyperparameters as per the LGBM documentation [7] and tuned them accordingly.

The number of decision leaves is the hyperparameter that most controls complexity. It is linked to the maximum depth, as was considered in RFs, since $2^{\textit{Maximum depth}}$ recovers the same number of leaves as the depth-wise tree. Leaf-wise tree growth may cause overfitting for small datasets, thus particular attention was given to the maximum depth parameter. Another important structural hyperparameter is the minimum data in leaf, which controls overfitting as it stops the LGBM from adding tree nodes that only describe a small number of observations. Also, there is a natural trade-off between the learning rate and the number of trees.

Other notable hyperparameters include bagging fraction (the proportion of training samples used to train each tree), bagging frequency, feature fraction (the proportion of features to sample when training each tree), L1 regularisation, L2 regularisation, maximum bin size for feature values, and minimum data per bin. A description of the hyperparameters along with reasonable search ranges and the final optimal value under Bayesian Optimisation is given in Table 4. The parameters absent from the table took their default values under the optimisation.

Parameter	**Description**	**Search range**	**Optimal value**
Number of leaves	Tree structure, training speed, accuracy	[8 - 4096]	8
Max. depth	Tree structure, training speed, overfitting	[3 - 12]	12
Min. data in leaf	Tree structure, training speed, overfitting	[3 - 50]	17
Learning rate	Accuracy	[0.01 - 0.3]	0.3
Number of trees	Accuracy, training speed	[100 - 10000]	3587
Max. bin size	Accuracy, training speed, overfitting	[2 - 87]	87
Feature fraction	Overfitting, training speed	[0.2 - 1]	0.2

Table 4: Tuning of key hyperparameters for LGBM

The CV accuracy of the LGBM specified above was 96.7%. The most important words for the LGBM were image, said and featured, as shown in the feature importance plot, where information gain is essentially the difference in entropy before and after splits.

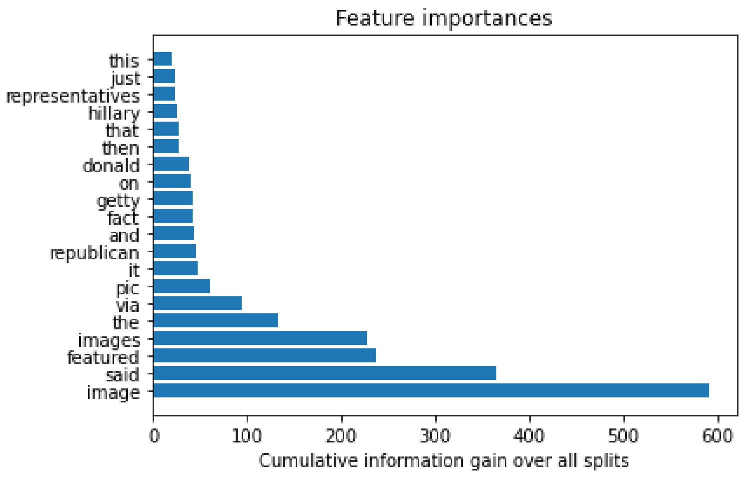

Figure 9: Importances for top 20 key features of LGBM

3 Discussion

Table 5 gives the training accuracy, re-estimated CV accuracy with naïve confidence limits, and runtime for each of the final classifiers. Each classifier achieves 100% accuracy on the training set, which could suggest overfitting, however, as shown by cross-validation, all models generalise well with LGBM (96.2%) performing best. LGBM also has the narrowest confidence bounds. LR with interaction terms may have improved the performance of this method. The accuracy of NNs may have been limited by the medium-sized dataset, as these approaches are typically data intensive. In particular, a larger training set may have enabled NNs to exceed the performance of boosting.

Interactions were not considered for LR due to computational limitations. Regardless, the overall complexity of LR was increased due to the use of principal components. LGBM was the most computationally costly, taking nearly 10 seconds to fit, whilst NNs and RFs were considerably faster. The default implementation of LGBM was efficient and accurate, so it is the hyperparameter tuning which increased runtime, but we deemed this worthwhile for much greater accuracy. Specifically, increasing the number of trees from the default of 100 to 3587 and having a large maximum depth of 12 had a notable effect on runtime.

Classifier	**Training Accuracy (%)**	**CV Accuracy (%)**	**CV CI (%)**	**Runtime (s)**
LR	100	94.9	[93.5 - 96.3]	6.75
NN	100	95.7	[94.4 - 97.0]	0.57
RF	100	95.4	[94.1 - 96.7]	2.49
LGBM	100	96.2	[95.0 - 97.4]	9.55

Table 5: Training accuracy, CV accuracy, CV confidence intervals, and runtime for the optimal classifier under each family.

NNs are the least interpretable since the classification process cannot be displayed visually, nor can the relative importance or interactions of the features be explored. Moreover, whilst LR provides a framework for interpreting the average effect of each feature on classification in simple cases, this is more difficult when principal components are used. Finally, RFs and LGBM preserved the most information as features were ranked by importance, allowing us to identify words that signify fake articles.

The important features given by LGBM suggests the articles relate to US politics, so an interesting extension would be to see whether the classifiers generalise to other topics. If the language used in fake news articles is consistent across topics, the classifiers may still provide accurate predictions. Likewise, it would be interesting to see whether accuracy improves with more data, or whether the overlap of language used in reliable and unreliable sources produces a natural ceiling to predictive accuracy.

Lastly, there was no preference for false negatives or false positives given in the problem. If a higher weight was placed on missclassifying fake news as reliable, known as false negatives, then recall may have been a more useful evaluation metric than accuracy. Similarly, if we were more concerned with false positives, we could have used precision to optimise the decision threshold.

4 Conclusion

The final classifier chosen was the LGBM with an estimated prediction accuracy of 96.2% on the test set. Although, it may be slightly lower to allow for optimism of cross-validation. This model was chosen as it gave the highest accuracy and was reasonably interpretable. The LGBM works particularly efficiently with sparse datasets, however, due to the large number of hyperparameters which were tuned, the runtime of the LGBM increased significantly. If the dataset were larger, NNs may have achieved higher accuracy, but interpretability would still have been limited. LR may require further feature selection to improve performance by considering feature interactions, but remains a useful tool for exploration.

References

[1] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer New York, 2009. ISBN: 9780387848587. URL: `https://books.google.co.uk/books?id=tVIjmNS3Ob8C`.

[2] Stephen Bates, Trevor Hastie, and Robert Tibshirani. "Cross-validation: what does it estimate and how well does it do it?" In: *arXiv preprint arXiv:2104.00673* (2021).

[3] F. Pedregosa et al. *Gini Importance*. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier.feature_importances_`.

[4] Guolin Ke et al. "Lightgbm: A highly efficient gradient boosting decision tree". In: *Advances in neural information processing systems* 30 (2017).

[5] Microsoft Corporation. *LightGBM: Leaf-wise Growth*. `https://lightgbm.readthedocs.io/en/latest/Features.html`.

[6] Microsoft Corporation. *LightGBM Parameters*. `https://lightgbm.readthedocs.io/en/latest/Parameters.html`.

[7] Microsoft Corporation. *LightGBM Parameter Tuning*. `https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html`.

Appendix

```python
# import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_score
import lightgbm as lgb
from sklearn.model_selection import KFold
from skopt import BayesSearchCV
import matplotlib.pyplot as plt
import random
import scipy.stats as st


#############
# Read data #
#############

# set random seed
np.random.seed(52)

# inputs of the training set
X_train = pd.read_csv('X_train.csv', index_col = 0)

# outputs of the training set
y_train = pd.read_csv('y_train.csv', index_col = 0, squeeze = True)

# inputs of the test set
X_test = pd.read_csv('X_test.csv', index_col = 0)


###############################
# Default LGBM implementation #
###############################

default_lgb = lgb.LGBMClassifier()
default_lgb.fit(X_train, y_train)

# define cross validation
kf = KFold(n_splits = 5, shuffle = True, random_state = 1)
scores = cross_val_score(default_lgb, X_train, y_train, cv = kf)

# training and CV accuracy for default model
print('Accuracy of LGBM on training set: ', default_lgb.score(X_train, y_train))
print('Mean 5-fold CV accuracy for default LGBM parameters: %0.3f' % (scores.mean()))


#################################################################
# Finding the best parameters of LGBM using Bayesian Optimisation #
```

```
###################################################################

# define the model
model = lgb.LGBMClassifier()

# obtain range for max_bin
max_bin = max(X_train.nunique())

# define search space
params = dict()
params['learning_rate'] = (0.01, 0.3)
params['n_estimators'] = (100, 10000)
params['num_leaves'] = (8, 4096)
params['max_depth'] = (3, 12)
params['min_data_in_leaf'] = (3, 50)
params['max_bin'] = (2, max_bin)
params['lambda_l1'] = np.linspace(0, 100, 100)
params['lambda_l2'] = np.linspace(0, 100, 100)
params['bagging_fraction'] = (0.2, 1)
params['bagging_freq'] = (0, 15)
params['feature_fraction'] = (0.2, 1)
params['min_gain_to_split'] = (0, 15)
params['min_data_in_bin'] = (3, 20)

# define no. of evaluations
iterations = 100

# define the search
search = BayesSearchCV(estimator = model,
                      search_spaces = params,
                      n_iter = iterations,
                      cv = kf)

# Perform the search, note that this may take a few hours, depending on the machine,
# and will give warnings which are expected as LGBM tries different parameter values
# to the default
search.fit(X_train, y_train)

# Report the best result
print('The best parameters are %s with a 5-fold CV accuracy of %f'
      % (search.best_params_, search.best_score_))

# Optimal model based on above
clf = lgb.LGBMClassifier(learning_rate = 0.3,
                         max_depth = 12,
                         num_leaves = 8,
                         n_estimators = 3587,
                         min_data_in_leaf = 17,
                         feature_fraction = 0.2,
```

```python
                         max_bin = 87)
clf.fit(X_train, y_train)
print('Accuracy of LGBM on training set: ', clf.score(X_train, y_train))


######################
# Feature Importance #
######################

importances = clf.booster_.feature_importance(importance_type = 'gain')
importances_df = pd.concat([pd.DataFrame(importances, columns = ['Importance']),
                            pd.DataFrame(X_train.columns, columns = ['Word'])], axis=1)
plt.figure()
plt.title('Feature importances')
ax = plt.barh(importances_df.sort_values(by = 'Importance',
                                         ascending = False)[:20]['Word'],
              importances_df.sort_values(by = 'Importance',
                                         ascending = False)[:20]['Importance'],
                                         align='center');
plt.xlabel('Cumulative information gain over all splits')


######################
# Export Predictions #
######################

# compute predictions on the test inputs
y_pred = clf.predict(X_test)

# export the predictions on the test data in csv format
prediction = pd.DataFrame(y_pred, columns=['Class'])
prediction.index.name='Index'
prediction.to_csv('P470-P187-P775-P453 - Predictions.csv')


####################################
# CV accuracy confidence intervals #
####################################

# returns cv accuracy estimate, and a confidence interval
def naive_cv(X, Y, estimator, n_folds = 5):

    # standard normal .975 quantile
    z = st.norm.ppf(.975)

    # define split
    fold_id = np.array(range(X.shape[0])) % n_folds
    fold_id = random.sample(list(fold_id), len(fold_id))
    accs = []
    for k in range(n_folds):
        # define validation indices for fold
        ind_val = list(np.where(np.equal(fold_id, k)))[0]

        # define training indices for fold
        ind_train = list(np.where(np.not_equal(fold_id, k)))[0]

        # fit model on training set
        estimator.fit(X[ind_train,:], Y[ind_train])

        # predict on validation set
        predictions = estimator.predict(X[ind_val, :])

        # store 1 - loss for each validation point
        acc_k = (predictions == Y[ind_val])
        accs.extend(acc_k)

    # estimate standard error for cv accuracy
    accs_std = np.std(accs) / np.sqrt(X.shape[0])

    # define confidence interval based on normality
    conf_int = [np.mean(accs) - z * accs_std, np.mean(accs) + z * accs_std]

    return dict({'mean':np.mean(accs), 'std.dev':accs_std, '95% CI':conf_int})


# train final model on new seed and print confidence interval
random.seed(104)
clf_cv = naive_cv(X = np.array(X_train), Y = y_train, estimator = clf, n_folds = 5)

print('LGBM:', clf_cv)
```