Fake News Detection Project
Team: Victoria, Princess, Nicola, Rose, Janka

INTRODUCTION

- Project background and context
    - Fake News recognition is one of the most challenging cybersecurity problems of our time. We have never been able to access information so easily before. The widespread use of fake news can cause serious social and political damage.
    - For example, the United Nations reported on the role Facebook played in fueling genocide in Myanmar. Or more recently, the spread of fake news related to COVID-19 has put global health at risk, of which the WHO released a warning explaining the effects of misinformation related to the pandemic.
    - More and more research focuses on the necessity to identify the impact of fake news, come up with solutions, and reduce the spread of fake news.
- Aim and the problem we want to find answers for
    - Our aim is to make an analysis of the detection of fake news and to explore different machine learning models to find the most effective one.

IMPLEMENTATION AND EXECUTION

- Development approach: Agile
    - Management: daily scrum meetings, initiating sprints, dealing with backlogs

Roles	Team	Responsibilities
Product Manager	Nicola	Presentation, Sentiment Analysis
Data Scientist	Victoria	EDA, Linear Regression, XGBoost
Data Engineer	Princess	EDA, Sentiment Analysis
ML Engineer	Rose	Support Vector Machine model
ML Engineer	Janka	Naive Bayes, collating final report

DATA SET

After setting up the Kaggle API and comparing different datasets, we downloaded the ‘Fake and real news dataset’. It consists of two different csv files: ‘Fake’ and ‘New’, as it is pre-classified into fake and real articles based on manual research by the authors. The two sets together consist of 44898 pieces of articles including information about title, date, subject and text.

DATA COLLECTION

Install kaggle library

In [2]:
```
# !pip install kaggle
```

Add credentials used for Kaggle API

In [3]:
```
# # try in bash (if not working in Jupyter) create folder .kaggle and copy kaggle.json (kaggle api key) to it
# !mkdir ~/.kaggle
# !cp ~/Downloads/kaggle.json ~/.kaggle/kaggle.json
```

Search for datasets

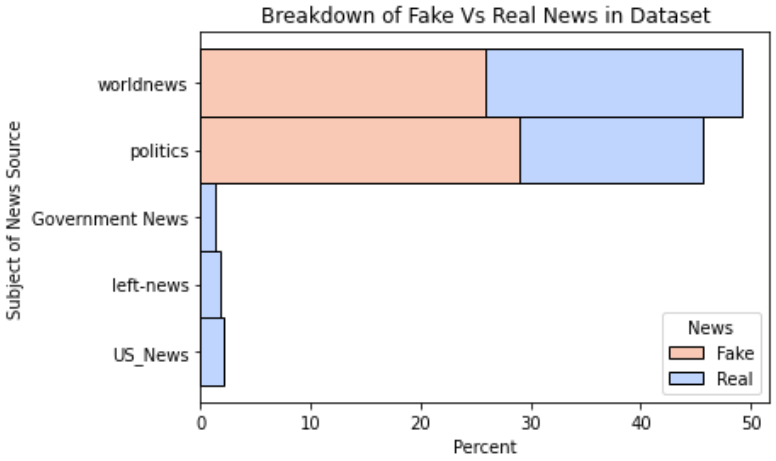
In [4]:
```
# search for the dataset
# !kaggle datasets list -s 'fake-news'
```

Dowload datasets

In [5]:
```
# # https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset?select=True.csv
# !kaggle datasets download -d clmentbisaillon/fake-and-real-news-dataset
```

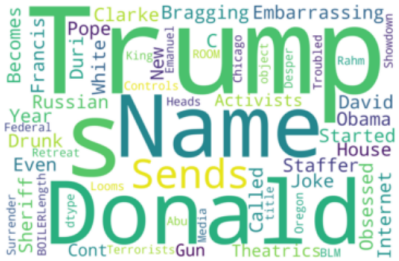
EDA

We removed the duplicates, then labelled the datasets (Fake=0, Real=1). After the datasets were merged, we were left with 21192 samples of real,and 17455 samples of fake articles. We were interested in checking the distribution of fake and real articles by the subjects of news sources. We used a histogram to visualise the data, and combined some subjects into politics and worldnews for better comparison.



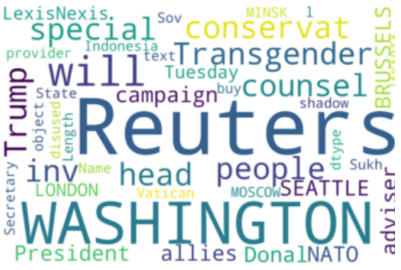
As we can see, the amount of real news articles are much higher for both world news and politics, although not all samples of the dataset have subjects assigned to them. Fake news articles appear higher within politics, which is not surprising considering that they might be created to fulfil some political agenda. Before we prepared the data for Natural Language Processing, we used WordCloud from the NLTK library to visualise the most common words that come up in the text and title columns.

WordCloud of titles



Fake(left) vs Real(right)

Fake(left) vs Real(right)



WordCloud of texts

Fake articles use more motivational words to trigger an emotional response, and these words are more apparent in the titles. The word usage of real articles are more objective both in their titles and content. We wanted to explore this relationship further so we completed a text sentiment analysis to see if there is a relationship between negative sentiments in titles and the news article being fake. This is explored in the Sentiment Analysis Section. Additional cleaning of the text was needed. The goal of the cleaning was to check for null values and empty places, and to find and remove nonsense texts. Additionally, we needed to remove urls and html tags from the text. We also identified a pattern,

that is most real news articles had their source at the beginning of their text: ' <CITY>, REUTERS()', which we needed to remove so it would not affect the training of the models.

We found no significant connection between date and target, nor other interesting relationships. Finally, we decided to consider the text and label columns for machine learning with the implementation of NLP processes.

Correlation between date and target

In [46]:
```
sns.heatmap(raw_df[['day', 'month', 'year', 'true']].corr(), annot=True)
sns.set(rc={'figure.figsize':(10,6)})
```

PREPROCESSING FOR NLP

Here, the libraries used were: nltk, re, contractors, and beautifulsoup for different cleaning processes. Ultimately, the following actions were taken:

- Removal of special characters
- Removal of punctuation
- Converting characters to lower characters
- Removing stopwords (words such as 'the', 'of', 'for')
- Removing patterns (urls, tags,etc.)
- Lemmatization (the stemming of words without loss of meaning to context)

Code snippets for demonstrating some of the examples above

```
# casefold
raw_df['text'] = raw_df['text'].map(lambda x: x.casefold())
```

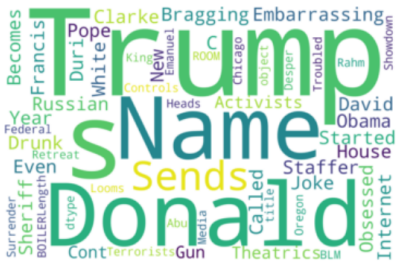
```
def remove_punctuation_spec_char(text_):
    punctuation = string.punctuation
    text_ = ' '.join([word for word in text_.split() if word not in punctuation])
    text_ = ' '.join([re.sub(r'[^a-zA-Z]','',word) for word in text_.split()])
    return text_

raw_df['text'] = raw_df['text'].map(lambda x: remove_punctuation_spec_char(x))
```

```
# Function to remove HTML tags
def remove_tags(text):

    # parse html content
    soup = BeautifulSoup(text, "html.parser")

    for data in soup(['style', 'script']):
        # Remove tags
        data.decompose()

    # return data by retrieving the tag content
    return ' '.join(soup.stripped_strings)
```

SENTIMENT ANALYSIS:

This section depicts the process of using sentiment analysis to see if there is a relationship between negative sentiments in titles and the news article being fake. This analysis was carried out after seeing the word cloud titles diagram which seemed to depict that fake news had more negative emotive words in the title when compared to the real news title. For instance, some of the prominent words found within the word cloud are emotive negative words such as “Bragging”, “Embarrassing" and “Troubled”. This pattern wasn’t as apparent in the real news dataset.

Examining Dataset




```
In [2]:
import pandas as pd
import numpy as np
pos1 = open("data/pos.txt", "r")
neg1 = open("data/neg.txt", "r")

mat = [n.split('\n') for n in pos1]
print(mat)
```

```
[['the rock is destined to be the 21st century\'s new " conan " and that he\'s going to make a splash even greater than arnol
d schwarzenegger , jean-claud van damme or steven segal . ', ''], ['the gorgeously elaborate continuation of " the lord of th
e rings " trilogy is so huge that a column of words cannot adequately describe co-writer/director peter jackson\'s expanded v
ision of j . r . r . tolkien\'s middle-earth . ', ''], ['effective but too-tepid biopic', ''], ['if you sometimes like to go
to the movies to have fun , wasabi is a good place to start . ', ''], ["emerges as something rare , an issue movie that's so
```

The data set came from Kaggle and we downloaded the ‘Positive and negative movie reviews data’.It consisted of two different .txt files: ‘Pos’ and ‘Neg’, and was pre-classified into positive and negative movie reviews. It was a large precleaned data set of 10660 rows when combined. The dataset was not in a dataframe and had to be converted into one. We added in a column called ‘sentiment’ into both datasets using numpy to fill the negative review columns as with ones and positives with zeros. There was an equal amount of negative and positive reviews. We then combined the datasets and reshuffled it to reduce bias in the model.

	text	sentiment
8121	audiard successfully maintains suspense on dif...	0
5662	family fare .	0
6107	desta vez , columbus capturou o pomo de ouro .	0
4010	the movie has generic virtues , and despite a ...	1
3874	if swimfan does catch on , it may be because t...	1
...	...	...
1447	illiterate , often inert sci-fi action thrille...	1
4598	enough is not a bad movie , just mediocre . th...	1
5097	don't hate el crimen del padre amaro because i...	1
9944	the cast is phenomenal , especially the women .	0
6489	what really makes it special is that it pulls ...	0

10660 rows × 2 columns

.

Building the Model

When building the model we set the features to 1000 to try and find the 1000 most common words and then created tokens for the most thousand words or symbols. We then fitted the tokenizer on the text. We also got the most common words and turned them into an integer array.We then used the Keras sequential model, tensorflow and tokenizer to build the model. We had three layers in our model; Embedding Layer, GRU layer and Dense layer. The embedding layer was there to more accurately learn the relationships between words. The GRU layer was there to keep track of previous inputs as in sentences it is important to do this to track sentiments. The Dense layer was there to calculate the probability of it to belong to the category of negative.

Testing the Model

```
def prep_input(arr, timesteps=timesteps, tokenizer=tokenizer):
  token_arr = tokenizer.texts_to_sequences(arr)
  pad_arr = pad_sequences(token_arr, maxlen=timesteps, padding='post')
  return np.flip(pad_arr, axis=1)
```

```
def test_input(reviews):
  prepped_input = prep_input(reviews)
  output = model.predict(prepped_input)
  answer = output
#rounds is up or down
  if answer > 0.6:
    print("This is negative")
  else:
    print("This is positive")

```

```
title1 = "We can forgive even bad fathers': Mia Hansen-Løve on making a movie haunted by Ingmar Bergman"
test_input([title1])
title = "happy happy love love yh"
test_input([title])
```

```
This is negative
This is positive
```

We then created a function to format input for feeding into the model and then created a function to run new inputs on the model. We then applied this to the fake news dataset.

Relationship between Negative Sentiment and Fake News Titles

It must be noted that when creating text analysis there are weaknesses as words can take on different meanings depending on the context of the overall sentence. This can make it difficult to display complex feelings found in text, such as humour and sarcasm. It is also important to note that the text sentiment analysis was built based on movie reviews, and not news article titles.

The model we built was able to accurately label the text as positive or negative. This was verified using samples as well as by sampling portions of the dataframe where the analysis had been run. The model wasn't accurate 100% of the time, as can be seen with the below sample.

```
sample = cleaned_df['title']
sample1 = sample.iloc[0]
test_input([sample1])
```

```
1/1 [==============================] - 0s 45ms/step
This is positive
```

click to scroll output; double click to hide

```
sample1
```

```
' Donald Trump Sends Out Embarrassing New Year's Eve Message; This is Disturbing'
```

Image showing incorrectly calculated piece of text. The output should be negative.

We also wanted to use the results from the sentiment analysis in a graph that represented the frequency of positive vs negative sentiments within the news dataset.

```
In [*]: cleaned_df['sentiment'] = cleaned_df.title.apply(lambda x:
                                                1 if test_input([x]) == 'This is negative' else 0)
```

```
1/1 [==============================] - 0s 25ms/step
This is positive
1/1 [==============================] - 0s 25ms/step
This is positive
1/1 [==============================] - 0s 39ms/step
This is negative
1/1 [==============================] - 0s 31ms/step
This is negative
1/1 [==============================] - 0s 28ms/step
This is negative
1/1 [==============================] - 0s 31ms/step
This is positive
1/1 [==============================] - 0s 29ms/step
```

Image showing the output of the sentiment analysis per row as it is tested

If we had more time, this would be the next step for this analysis step. The sentiment values would need to be added to the 'sentiment' column in order to plot a graph.
The trouble with the current script was when trying to input the sentiment values into the column as we couldn't get the script to input the desired output correctly.

```
cleaned_df.sample(10)
```

	title	text	true	sentiment
15442	TEXAS COLLEGE STUDENTS OUTRAGED: Traditional 9...	the shocker about this is that students on bot...	0	1
19713	Mueller impanels Washington grand jury in Russ...	special counsel robert mueller has impaneled a...	1	1
28985	Australia finds wreck of first Allied submarin...	australia has discovered the wreck of its firs...	1	1
3875	WATCH: Van Jones Says EXACTLY What Needs To B...	the election result tonight is hard to swallow...	0	1
979	Fox News Says 'Health Insurance Doesn't Matte...	figuring out the best way to deal with healthc...	0	1
1565	Here's The Disgusting Propaganda The White Ho...	the world is eagerly awaiting the testimony of...	0	1
29742	Death toll from fighting in South Sudan's Grea...	the death toll from interclan fighting in sout...	1	1
33306	Afghan Taliban say kidnapped U.S. professor is...	the afghan taliban said on monday that kevin k...	1	1
598	Mooch Changes His Tune, Says Reporter 'Absolu...	so here s a funny hypothetical you re a white ...	0	1
16502	LAW ENFORCEMENT ON HIGH ALERT Following Threat...	no comment is expected from barack obama membe...	0	1

Image showing how script has imputed a sentiment value of 1 in all rows

MACHINE LEARNING MODELS

NAIVE BAYES MODEL WITH COUNT VECTORIZATION

Naive Bayes classifier makes decisions by comparing two probabilities of an event. If the probability of the article being fake is higher than being real, the model will classify it to be fake. As we will use the texts of the articles, the feature would be described as the count of the most frequent words for the model. Sci-kit library was used for the Countvectorizer and for Naive Bayes model.

The text is prepared with removal of punctuations, no whitespaces, no stopwords, no special characters and lemmatization was also performed. The training was done by showing it all of the features of all the words, and let it try to figure out the more meaningful differences between a fake article and a real one, by simply looking for common frequent words.

```
In [12]:
vectorizer = CountVectorizer(stop_words='english')

# Used stop_words parameter to remove additional ones that nltk in data cleaning did not remove
# converts text to binary data
```

```
In [13]:
all_words = vectorizer.fit_transform(df.text.values.astype('U')) #needed to convert data to unicode
```

```
In [14]:
all_words.shape
#we receive a sparce matrix
#the columns represent the tokens of our text
```

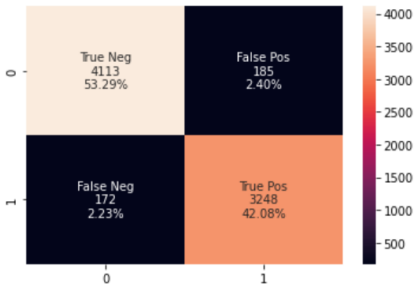
```
Out[14]: (38590, 207937)
```

```
In [15]:
vectorizer.vocabulary_
#we can see the tokens by the count, one by one
```

```
Out[15]: {'donald': 50350,
          'trump': 188764,
          'just': 93177,
```

All together, 7361 samples were classified correctly and 357 samples were classified incorrectly. The accuracy of the Naive Bayes model is: 95.37%.

Confusion Matrix illustrated with a heatmap



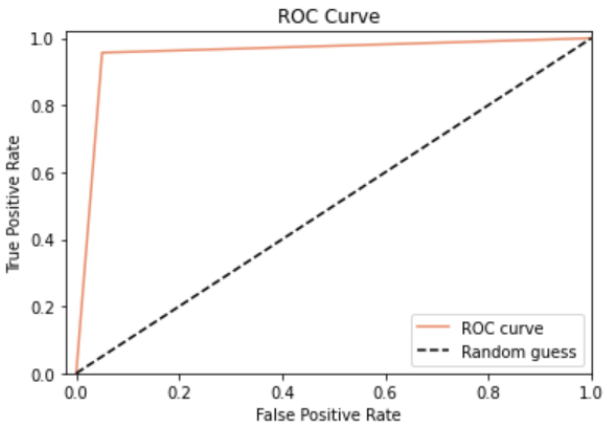
Classification report

```
print(classification_report(y_test, classifier.predict(x_test)))
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      3420
           1       0.96      0.96      0.96      4298

    accuracy                           0.95      7718
   macro avg       0.95      0.95      0.95      7718
weighted avg       0.95      0.95      0.95      7718
```

Confusion matrix and classification model was plotted, from which the ROC curve was created. Naive Bayes model is a simple and fast training model, and it generally has good performance on learning and predicting from text materials.

ROC Curve



The area under the ROC curve demonstrates the performance of the classification algorithm. What we can see is that our classifier is very good at minimising false negatives (fake articles which are classified as real) and true negatives (real articles which are classified as fake).

The model was also tested on another dataset - a pre labelled set of news articles, the same one linear regression model was tested for generalisation. The two models yielded similar results, their performance and accuracy are not good, they cannot be properly generalised.

Print classification report

In [35]:
```
1 print(classification_report(y_test, classifier.predict(x_test)))
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.65      0.72       272
         1.0       0.53      0.71      0.61       147

    accuracy                           0.68       419
   macro avg       0.67      0.68      0.67       419
weighted avg       0.71      0.68      0.68       419
```

Support Vector Machine Model With Count Vectorisation

The support vector machine is a binary classifier that classifies a dataset by drawing a hyperplane with the maximised margin. To convert the text messages into numerical information that is compatible with this machine learning algorithm, the TfidfVectorizer() function is implied to count the frequency of each individual word.

30% of the dataset is used as the testing dataset. 14% of the dataset is used as the validation dataset to avoid overfitting, 56% of the dataset is used to train the SVM classifier.

```
In [4]: df_text=df['text']
```

```
In [5]: vectorizer = TfidfVectorizer(max_df=0.5, min_df=2, ngram_range=(1,1), max_features=1000)

        text = vectorizer.fit_transform(df_text)
        vectorizer.get_feature_names()
```

```
 'actually',
 'added',
 'adding',
 'address',
 'administration',
 'adviser',
 'after',
 'again',
 'against',
 'agencies',
 'agency',
 'agenda',
 'ago',
 'agreed',
 'agreement',
 'ahead',
 'aid',
 'air',
 'all',
 'allegations',
```

```
In [6]: X_trainvad, X_test, y_trainvad, y_test = train_test_split(text, df['true'],test_size = 0.3)
        X_train, X_vad, y_train, y_vad=train_test_split(X_trainvad,y_trainvad, test_size=0.2)
```

Then the hyper-parameters are selected by implying the gridsearchCV() hyper-tuning function. The hyper-parameters for selection is C parameter and the kernel parameter. The C parameter decides how much misclassification is allowed in the sense to avoid overfitting on the training dataset. The kernel parameter decides whether the hyperlane is a linear hyperlane or in other shapes. The kernel chosen is a linear kernel, because the dataset has a large number of features (1000 features in this case) compared to its size. The best set of parameters is C=100 and kernel= ‘linear’.

hypertuning

```
In [*]: hyper_parameters = {'C':[0.1,30,60], 'kernel':['rbf','poly','linear']}
        rs_clf_feat = GridSearchCV(svm.SVC(), param_grid = hyper_parameters, cv=3, scoring='f1', return_train_score = True)
        rs_clf_feat.fit(X_train, y_train.ravel())
        best_params = rs_clf_feat.best_params_
        print('the best hyper parameter:')
        print (pd.DataFrame(best_params, index=[0]))
        print('the grid_search report:')
        pd.DataFrame.from_dict(rs_clf_feat.cv_results_)
```

The hyper-parameters selected from the hypertuning stage are then used for the SVM classifier. The SVM classifier is trained on the training dataset and then tested on the validation dataset. The f1 scores for the training dataset and the validation dataset are compared to detect if there is an overfitting. It turns out both f1 scores are pretty close and there is no overfitting on the training dataset.

overfitting detection

In [14]:
```
clf = svm.SVC (C=100, kernel = 'linear')
classifier = clf.fit(X_train, y_train)
y_pred = classifier.predict(X_vad)
y_pred_train=classifier.predict(X_train)
```

In [18]:
```
from sklearn.metrics import f1_score
print('validation f1 score',f1_score(y_vad, y_pred))
print('training f1 score', f1_score(y_train, y_pred_train))
```

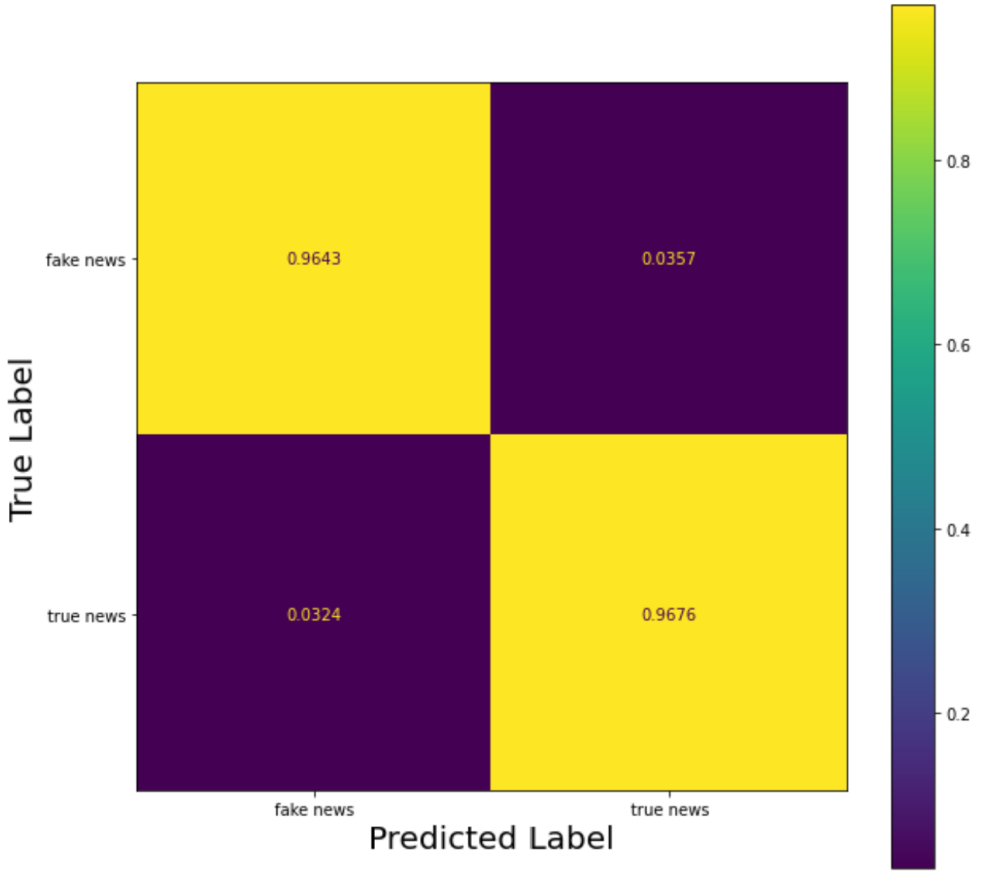
```
validation f1 score 0.9692513368983958
training f1 score 0.9940275908479139
```

The final stage is to test the SCM classifier on the unseen testing dataset. A report of all relevant scores is generated, and the normalised confusion matrix is shown below. Mathematically speaking, the diagonal of the normalised confusion matrix indicates the prediction accuracy for each class (in this case, the classes are ‘fake news’ and ‘true news’’).

testing and testing result

5]:
```
y_pred_test=classifier.predict(X_test)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      5245
           1       0.97      0.97      0.97      6332

    accuracy                           0.97     11577
   macro avg       0.97      0.97      0.97     11577
weighted avg       0.97      0.97      0.97     11577
```



As the C parameter is set to be 100, which means there is a certain level of misclassification in this classifier, the hinge loss is then calculated to see if the extend of misclassification is acceptable.

hinge loss

In [20]:
```
from sklearn.metrics import hinge_loss
pred_decision = clf.decision_function(X_test)
print(hinge_loss(y_test, pred_decision))
```

```
0.10159973505314619
```

In [ ]:

LOGISTIC REGRESSION

Logistic Regression predicts the probability of an event occurring. The curve on a graph for a logistic regression model is defined by logistic function (S-shaped) and bounded between 0 and 1.

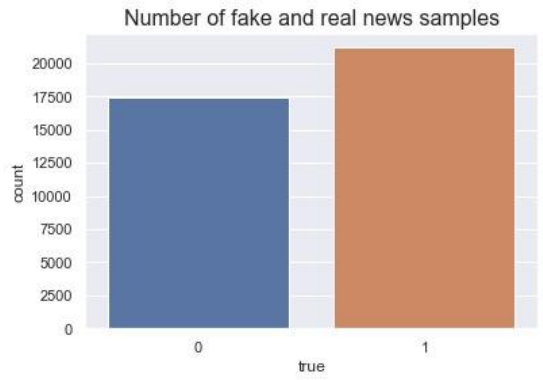
Data preprocessing using Natural Language Toolkit

- Used **nltk.corpus stopwords** library to remove common english words
- Used **nltk.corpus wordnet** library to tag words with position of speech (role words play when used together in sentences)

- Using nltk.stem **WordNetLemmatizer** to stem words to initial form (eg. revealed -> reveal, dogs -> dog, etc)

Check target values ratio

```
sns.countplot(x='true', data=raw_df)
plt.title('Number of fake and real news samples', fontdict={'fontsize':16})
plt.show()
```



Vectorizing text using TfidfVectorizer

- TfidfVectorizer scale down the impact of tokens that occur very frequently in a given corpus

```
vectorizer = TfidfVectorizer()
vectorizer.fit(X)

X = vectorizer.transform(X)
```

```
X.shape
```

```
(38589, 199275)
```

```
vectorizer = TfidfVectorizer(max_features=5000,ngram_range=(1,3))
vectorizer.fit(X)

X = vectorizer.transform(X)
```

```
X.shape
```

```
(38589, 5000)
```

Split data into training and testing sets

- Used sklearn.model_selection train_test_split library
- 75% train set to 25% test set ratio
- Pseudo random state 42 used for reproducible result

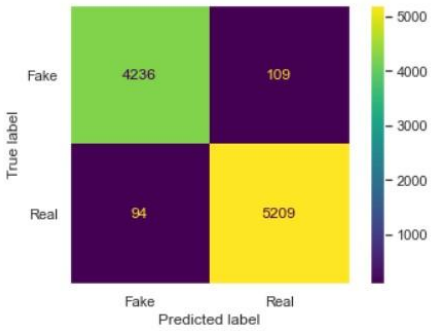
```
# Split data into train and test sets in ratio 75%/25% (pseudo randomized random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((28941, 199275), (28941,), (9648, 199275), (9648,))
```

Trained Logistic Regression (default parameters) and tested on test set

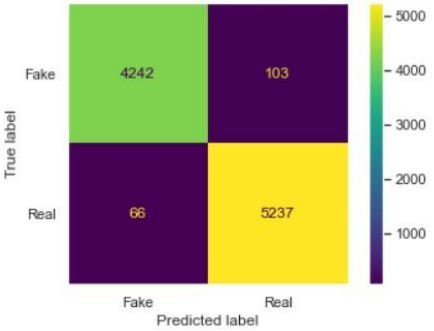
- Parameters: **{'C': 1.0, 'max_iter': 100, 'solver': lbfgs}**
- Classification report, Confusion matrix

	precision	recall	f1-score	support
Fake	0.98	0.97	0.98	4345
Real	0.98	0.98	0.98	5303
accuracy			0.98	9648
macro avg	0.98	0.98	0.98	9648
weighted avg	0.98	0.98	0.98	9648



TfidfVectorizer (max_features=5000,ngram_range(1,3))

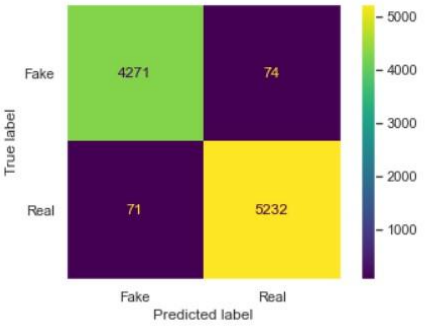
	precision	recall	f1-score	support
Fake	0.98	0.98	0.98	4345
Real	0.98	0.99	0.98	5303
accuracy			0.98	9648
macro avg	0.98	0.98	0.98	9648
weighted avg	0.98	0.98	0.98	9648



Parameters tuned Logistic Regression

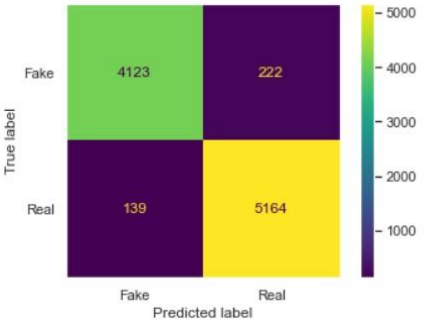
- Used GridSearchCV to find best hyperparameters
  - Tried stratified cross-validation for imbalance data, get same result as simple cv (hold-out cross-validation). Simple cv took less time.
- Classification report, Confusion matrix **{'C': 10000.0, 'max_iter': 1000, 'solver': 'saga'}**
  - Logistic Regression parameter ‘C’ C is the inverse of regularization strength (lambda). The trade-off parameter of logistic regression that determines the strength of the regularization, and higher values of C correspond to less regularization.

	precision	recall	f1-score	support
Fake	0.98	0.98	0.98	4345
Real	0.99	0.99	0.99	5303
accuracy			0.98	9648
macro avg	0.98	0.98	0.98	9648
weighted avg	0.98	0.98	0.98	9648



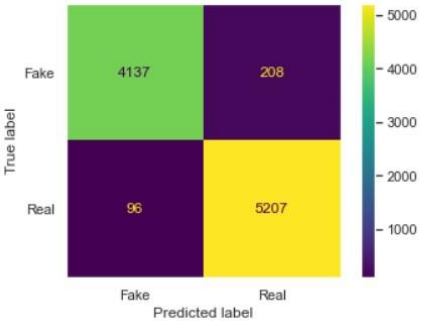
- Classification report, Confusion matrix **{'C': 0.1, 'max_iter': 100, 'solver': 'newton-cg'}**

	precision	recall	f1-score	support
Fake	0.97	0.95	0.96	4345
Real	0.96	0.97	0.97	5303
accuracy			0.96	9648
macro avg	0.96	0.96	0.96	9648
weighted avg	0.96	0.96	0.96	9648

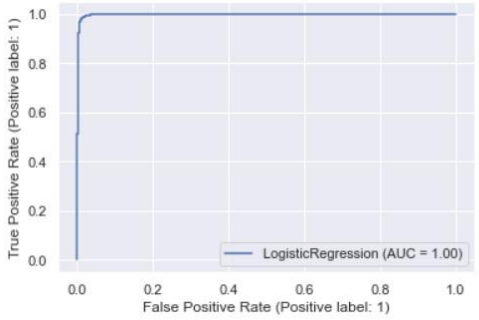


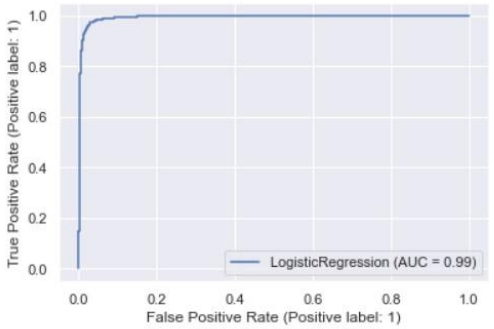
TfidfVectorizer (max_features=5000,ngram_range(1,3))

	precision	recall	f1-score	support
Fake	0.98	0.95	0.96	4345
Real	0.96	0.98	0.97	5303
accuracy			0.97	9648
macro avg	0.97	0.97	0.97	9648
weighted avg	0.97	0.97	0.97	9648



- Receiver operating characteristic (ROC) curve
  {'C': 10000.0, 'max_iter': 1000, 'solver': 'saga'} vs {'C': 0.1, 'max_iter': 100, 'solver': 'newton-cg'} + TfidfVectorizer(max_features=5000,ngram_range(1,3))





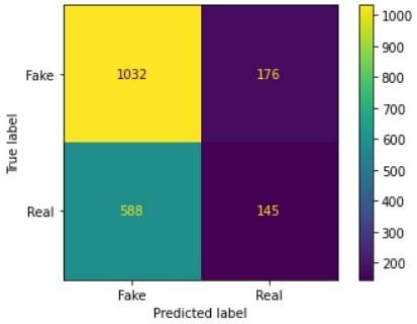
Test how well generalised trained and tuned Logistic Regression algorithm

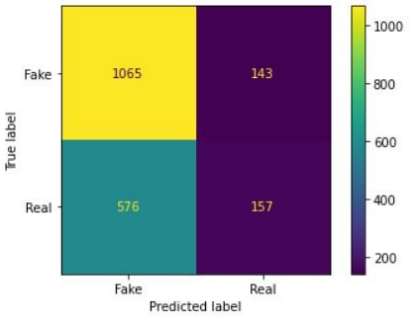
- Downloaded another dataset from Kaggle, Cleaned, Preprocessed
- Vectorized using previously saved in the pickle file vectorizers fitted with train dataset
- Load previously trained machine learning algorithms
- Return classification report for prediction on a new dataset
  {'C': 10000.0, 'max_iter': 1000, 'solver': 'saga'} vs {'C': 0.1, 'max_iter': 100, 'solver': 'newton-cg'} + TfidfVectorizer(max_features=5000,ngram_range(1,3))

	precision	recall	f1-score	support
Fake	0.64	0.85	0.73	1208
Real	0.45	0.20	0.28	733
accuracy			0.61	1941
macro avg	0.54	0.53	0.50	1941
weighted avg	0.57	0.61	0.56	1941

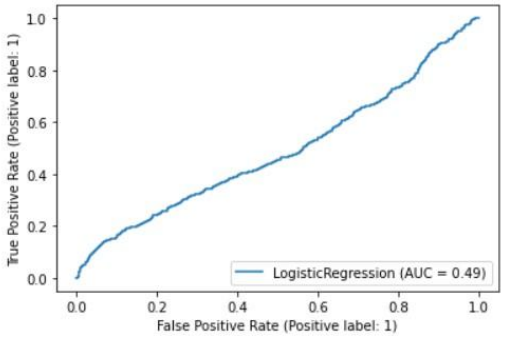
	precision	recall	f1-score	support
Fake	0.65	0.88	0.75	1208
Real	0.52	0.21	0.30	733
accuracy			0.63	1941
macro avg	0.59	0.55	0.53	1941
weighted avg	0.60	0.63	0.58	1941

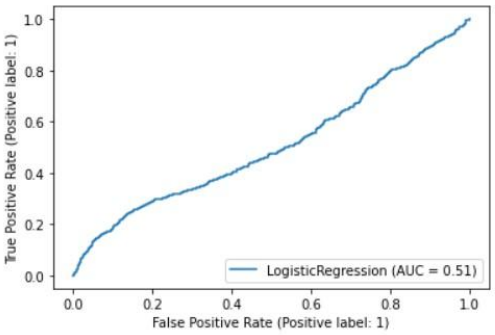
- Plot confusion matrix





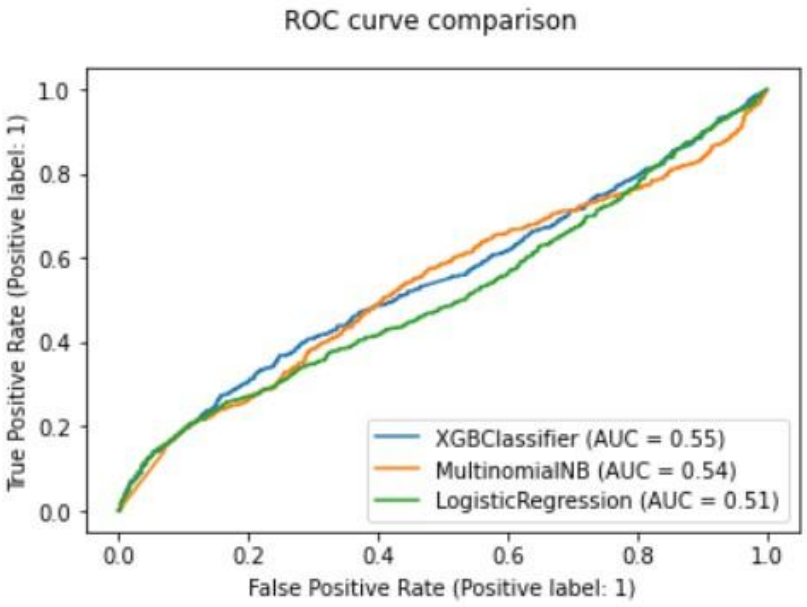
- Receiver operating characteristic (ROC) curve





Comparison ROC plot for Logistic Regression, XGBoost and Naive Bayes tested on new dataset

https://www.kaggle.com/datasets/ruchi798/source-based-news-classification



RESULTS & CONCLUSION:

This work focuses on different types of supervised machine learning models to review their performance on detecting fake news from text articles. The articles in the dataset were pre-classified into fake and real categories. A thorough data cleaning was performed on the text files with different approaches for preparing the data for the prediction models. For example, stopwords, special characters, and punctuation marks, tags or URLS were all removed for all models, but words were also tagged for position of speech for training the linear regression model. Sentiment analysis was carried out to reveal any relationship between negative words and titles or content of fake articles inspired by the world cloud plotted during the explanatory analysis. The extracted features were trained using Linear Regression, Super Vector Machine and Naive Bayes models. Overall, the accuracy ranged between 95 and 98%, which shows the importance of preprocessing the text. The Linear Regression Model outperformed the others with an accuracy score of 98%. It was also tested on a different database to test generalisation with an accuracy of 63%. Surprisingly, on a different dataset the Naive Bayes model performed slightly better with an accuracy of 68%. It is not easy to expand the models with good performance on different datasets. Future work should be implementing different models to find which method works best with which text data extracted from different sources and to fine-tune the existing models.

Reference:

https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset?select=True.csv