JOSE VERA

FAKE NEWS CLASSIFIER

DATASET AND GOAL

- Dataset: Fake news and real news dataset from kaggle
  - Total of 79592 samples and labels.
  - Fake.csv: 85668 samples and labels
  - Real.csv: 93924 samples and labels
- Classify documents or sentences as pertaining to real news or fake news
- Training Size (80%) = 35918
- Testing Size: (20%) = 8980

```
fake_news = pd.read_csv('Fake.csv')
real_news = pd.read_csv('True.csv')

fake_news['class'] = 0
real_news['class'] = 1
```

```
X_train, X_test, y_train, y_test = train_test_split(data['text'], data['class'], test_size=0.2, random_state=42)
```

PREPROCESSING

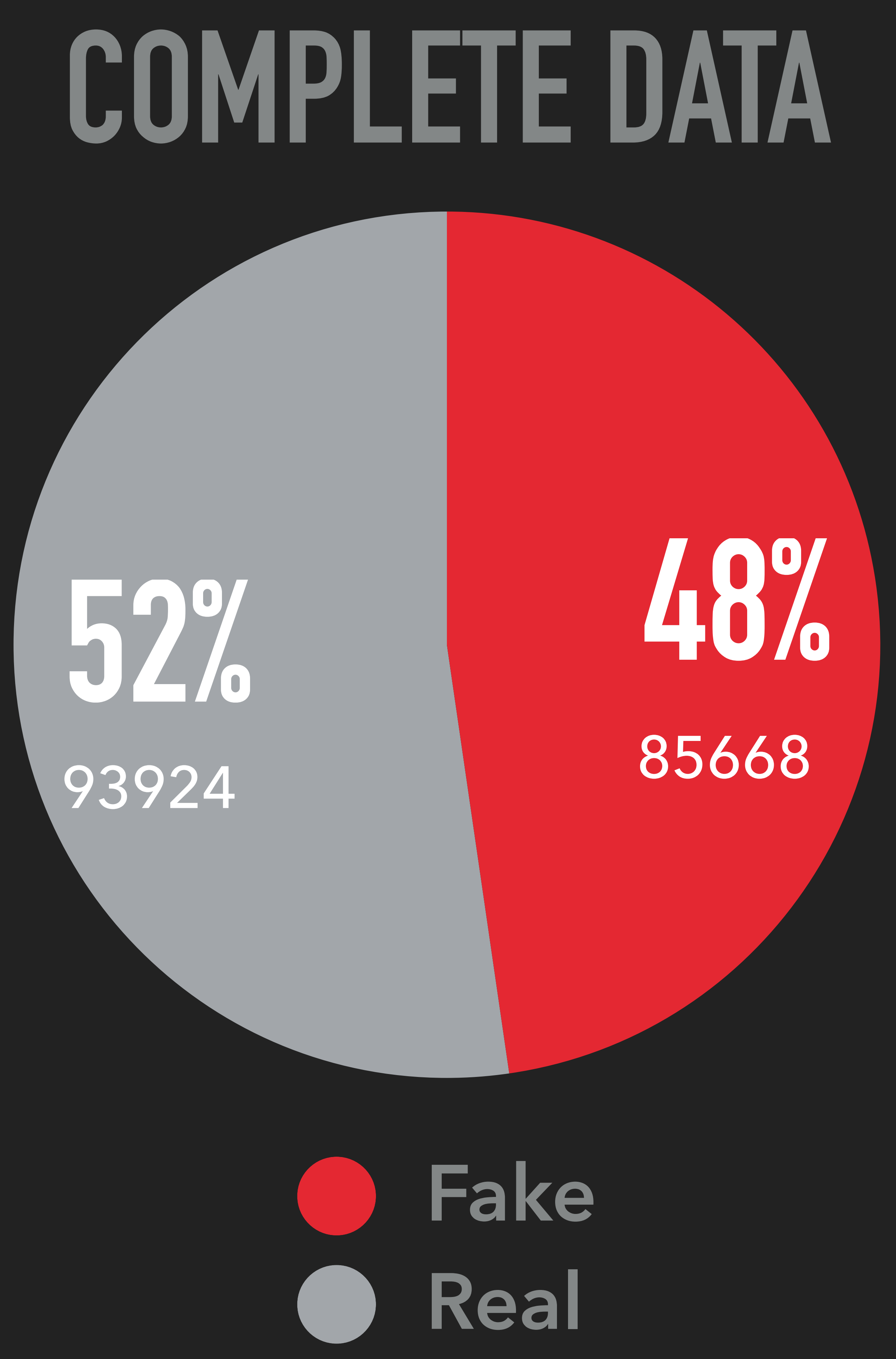
- Combine and shuffle real and fake news

```python
# Create a list of DataFrames to concatenate
dfs = [fake_news[['title', 'text', 'class']], real_news[['title', 'text', 'class']]]
data = pd.concat(dfs, ignore_index=True)
data = data.sample(frac=1).reset_index(drop=True) ##shuffle data
data['text'] = data['title'] + ' ' + data['text']  ## Combine title and text
```

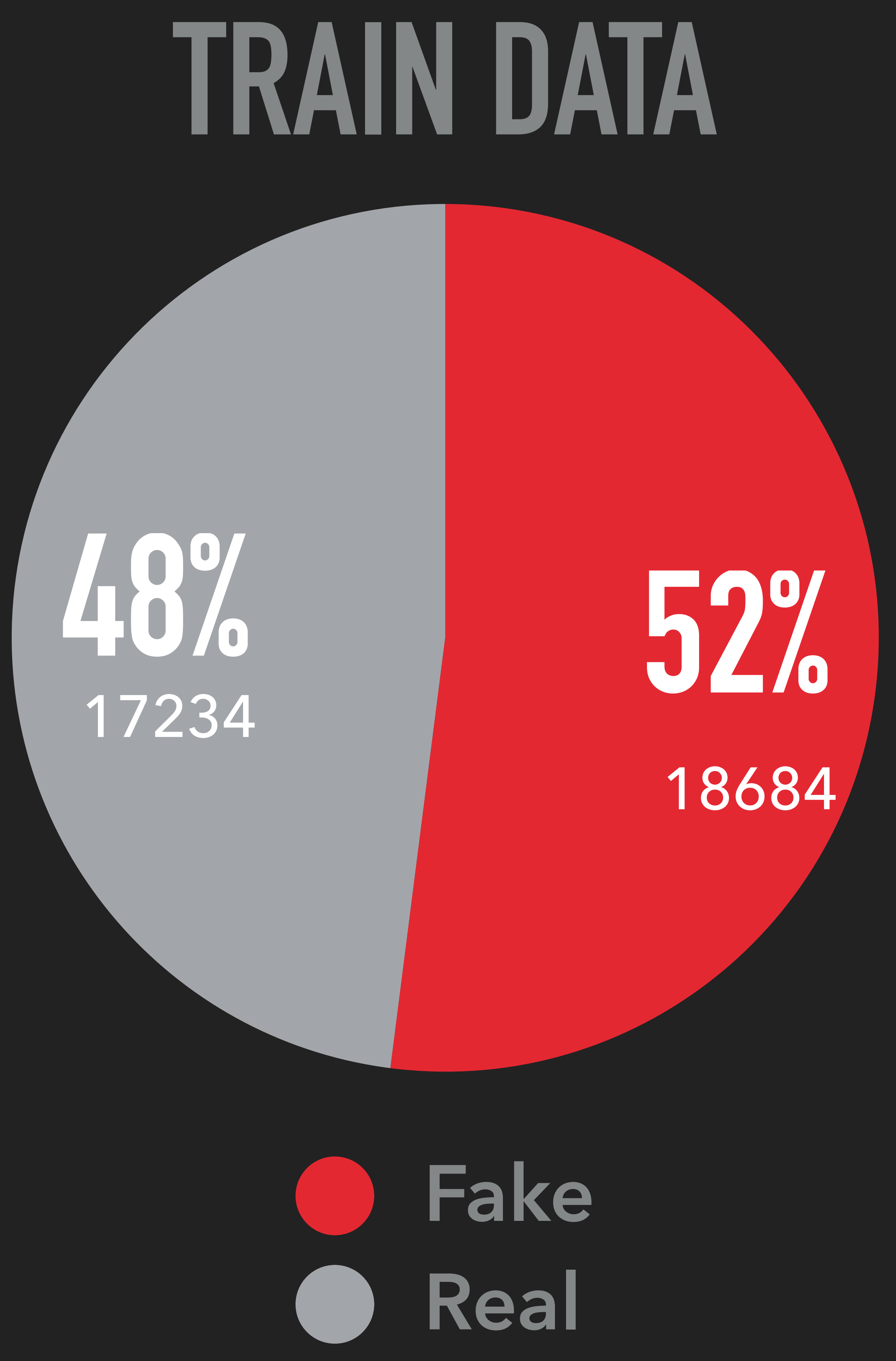
- Check for ingore_step
  - Use nltk to remove stop words
  - Use nltk's PorterStemmer for stemming

```python
if ignore_step != 'lowercase':
    data['text'] = data['text'].apply(lambda x: x.lower())  # Lowercase
##Remove Stopwords
data['text'] = data['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))
##Perfrom Stemming
data['text'] = list(map(lambda x: ' '.join(ps.stem(word) for word in x.split()), data['text']))
##Remove non alphatical characters
data['text'] = data['text'].apply(lambda text: re.sub(r'[^a-zA-Z\s]', ' ', text))
```

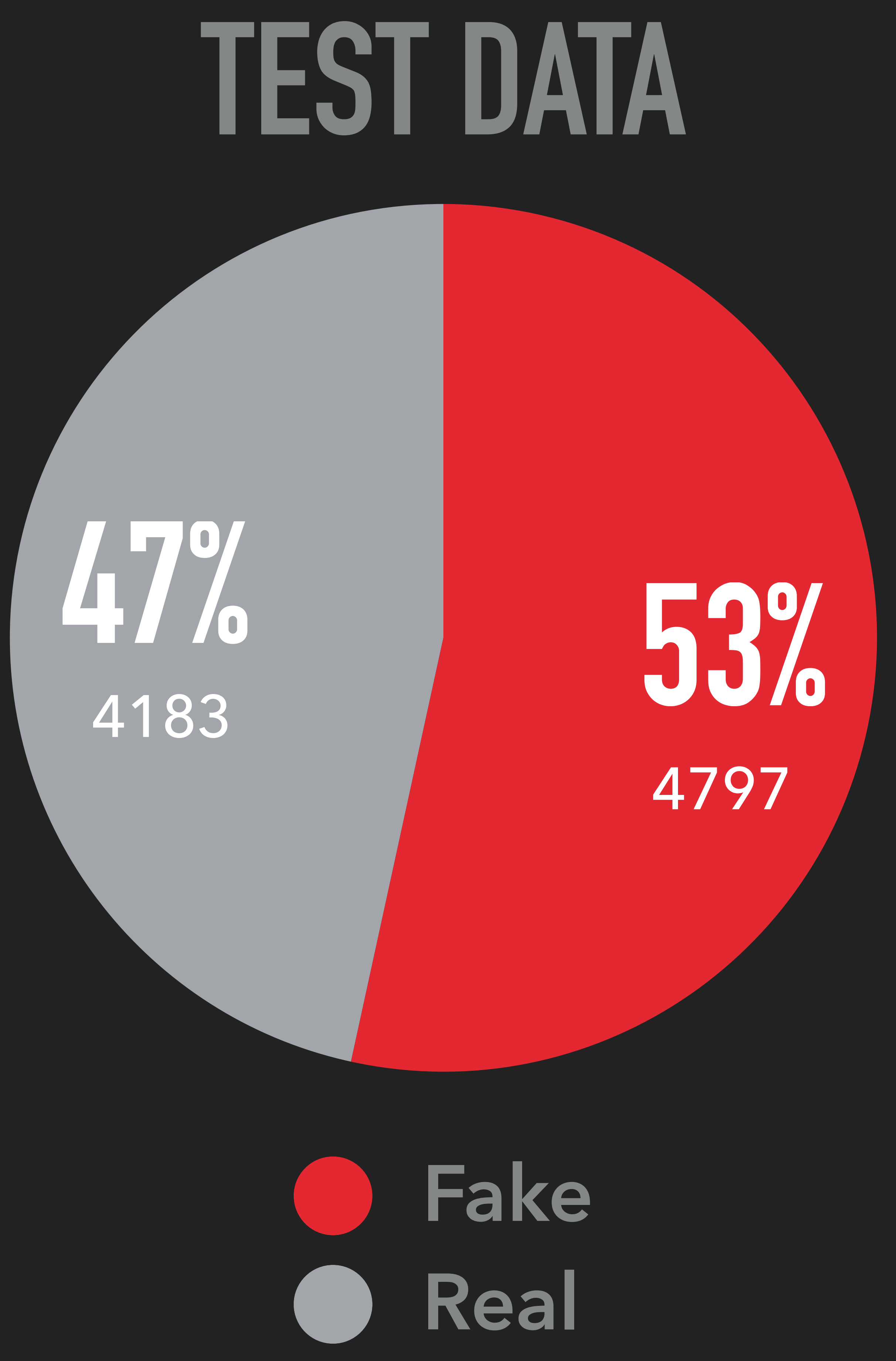
TEST/TRAIN SPLIT



79592 Samples



35918 Samples



8980 Samples

TRAINING

- Create bag of words matrix from input
  - CountVectorizer from sklearn
- Separate into two matrices for each class
- Calculate the log(P(C)): the log probability of each class

```python
def train_naive_bayes(X_train):
    ##Binary count vectorizer object
    vectorizer = CountVectorizer(binary=True).fit(X_train)
    X_train_bow_matrix = vectorizer.transform(X_train).toarray()
    ##Separate BOW into different matrices
    X_fake = X_train_bow_matrix[y_train == 0, :]
    X_real = X_train_bow_matrix[y_train == 1, :]
    # Calculate P(c) term
    log_prior = {}
    numb_doc = len(X_train_bow_matrix)
    numb_classes = 2
    class_counts = np.bincount(y_train)
    for label in range(numb_classes):
        log_prior[label] = np.log(class_counts[label]/numb_doc)
```

TRAINING

- Create vocabulary from training set
- Sum up column to get counts of words for each class
- Sum up the sums to get total number of words in each class
- Calculate probabilities with add1 smoothing
- Return log_prior, log_likely, V_list

```python
# Create Vocabulary of D
V = vectorizer.get_feature_names_out()
##Get necessary counts to calcualte probability
real_word_counts = np.sum(X_real, axis=0)
fake_word_counts = np.sum(X_fake, axis=0)
real_words_total = np.sum(real_word_counts)
fake_words_total = np.sum(fake_word_counts)
real_doc_count = len(X_real)
fake_doc_count = len(X_fake)

#Calculate probabilites using lapace smoothing of 1
fake_probs = {}
real_probs = {}
for word in range(len(V)):
    fake_count = fake_word_counts[word]
    real_count = real_word_counts[word]
    fake_probs[V[word]] = np.log((fake_count + 1) / (fake_words_total + len(V)))
    real_probs[V[word]] = np.log((real_count + 1) / (real_words_total + len(V)))
# Create log_likelihood dictionary
log_likelihood = {}
log_likelihood[0] = fake_probs
log_likelihood[1] = real_probs

V_list = V.tolist()

return log_prior,log_likelihood,V_list,
```

TESTING APPROACH

- Create BOW matrix for test set
- Create log likelihood matrix:
  - Each entry represents the log-likelihood of observing that word given a class.
- Using matrices can significantly speed up calculation for sum of log likelihoods
  - The @ operator performs matrix scalar multiplication using broadcasting
- Choose class with highest sum

```python
def test_naive_bayes(X_test, log_prior, log_likelihood, C, V):

    vectorizer = CountVectorizer(vocabulary=V, binary=True)
    testdoc = vectorizer.transform(X_test).toarray()

    # Create a matrix of log likelihoods for all words in the vocabulary for each class
    log_likelihood_matrix = np.array([list(log_likelihood[c].values()) for c in C]).T

    # Calculate the sum of log likelihoods for each document and class using broadcasting
    sum_c = (testdoc @ log_likelihood_matrix) + list(log_prior.values())

    # Choose the class with the highest sum
    best_c = np.argmax(sum_c, axis=1)

    return best_c, sum_c
```

RESULTS

All Steps	Actual Real	Actual Fake
Predicted Real	TP = 4159	FP=138
Predicted Fake	FN =139	TN =4544

No Lowercasing	Actual Real	Actual Fake
Predicted Real	TP = 4217	FP=165
Predicted Fake	FN =122	TN =4476

Sensitivity (recall): 0.9676

Specificity: 0.9705

Precision: 0.96788

Negative predictive value: 0.9703

Accuracy: 0.9691

F-score: 0.9677

Sensitivity (recall): 0.97188

Specificity: 0.9644

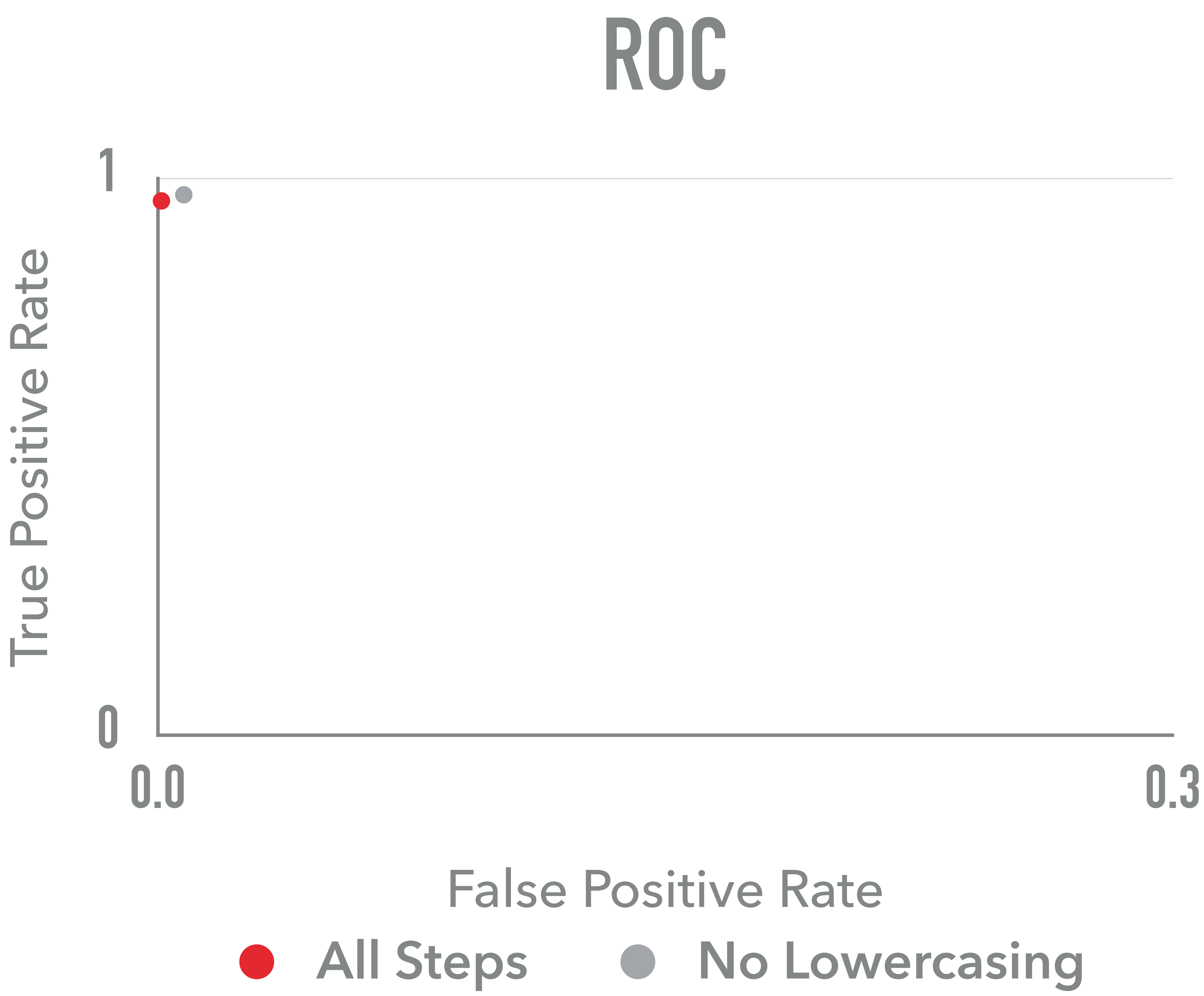
Precision: 0.9623

Negative predictive value: 0.97

Accuracy: 0.9680

F-score: 0.9670

RESULTS


ROC
1
0
True Positive Rate
0.0
0.3
False Positive Rate
All Steps
No Lowercasing

SENTENCE RESULTS

```
Sentence 'vaccines stimulate the immune system to develop immunity or resistance to a pathogen' was classified as 'Real'.
P(Real | S) = -70.7975
P(Fake | S) = -73.1993
Sentence 'vaccines are not safe effective for people to use, and may in fact cause autism' was classified as 'Fake'.
P(Fake | S) = -65.9053
P(Real | S) = -68.6854
Sentence 'the governor of florida has declared a state of emergency due to the hurricane' was classified as 'Real'.
P(Real | S) = -54.4398
P(Fake | S) = -57.8875
Sentence 'scientists confirm that drinking bleach can cure covid-19' was classified as 'Fake'.
P(Fake | S) = -50.6290
P(Real | S) = -53.2120
Sentence 'google has announced that they have successfully demonstrated quantum supremacy ' was classified as 'Fake'.
P(Fake | S) = -58.0013
P(Real | S) = -61.0427
Sentence '"google says it has achieved 'quantum supremacy" (source: bbc news, october 2019)' was classified as 'Fake'.
P(Fake | S) = -82.5893
P(Real | S) = -87.6328
```

CONCLUSION

- The Naive Bayes classifier performed well in accurately predicting both real and fake news
- The model achieved high sensitivity (recall), indicating its ability to identify most of the true positive cases.
  - The majority of the positive predictions made by the model were correct.
- Overall, the classifier has shown good performance on this dataset.
  - Performance may not be as good for shorter sentences
  - Algorithm might perform worse with slang, and colloquial terms