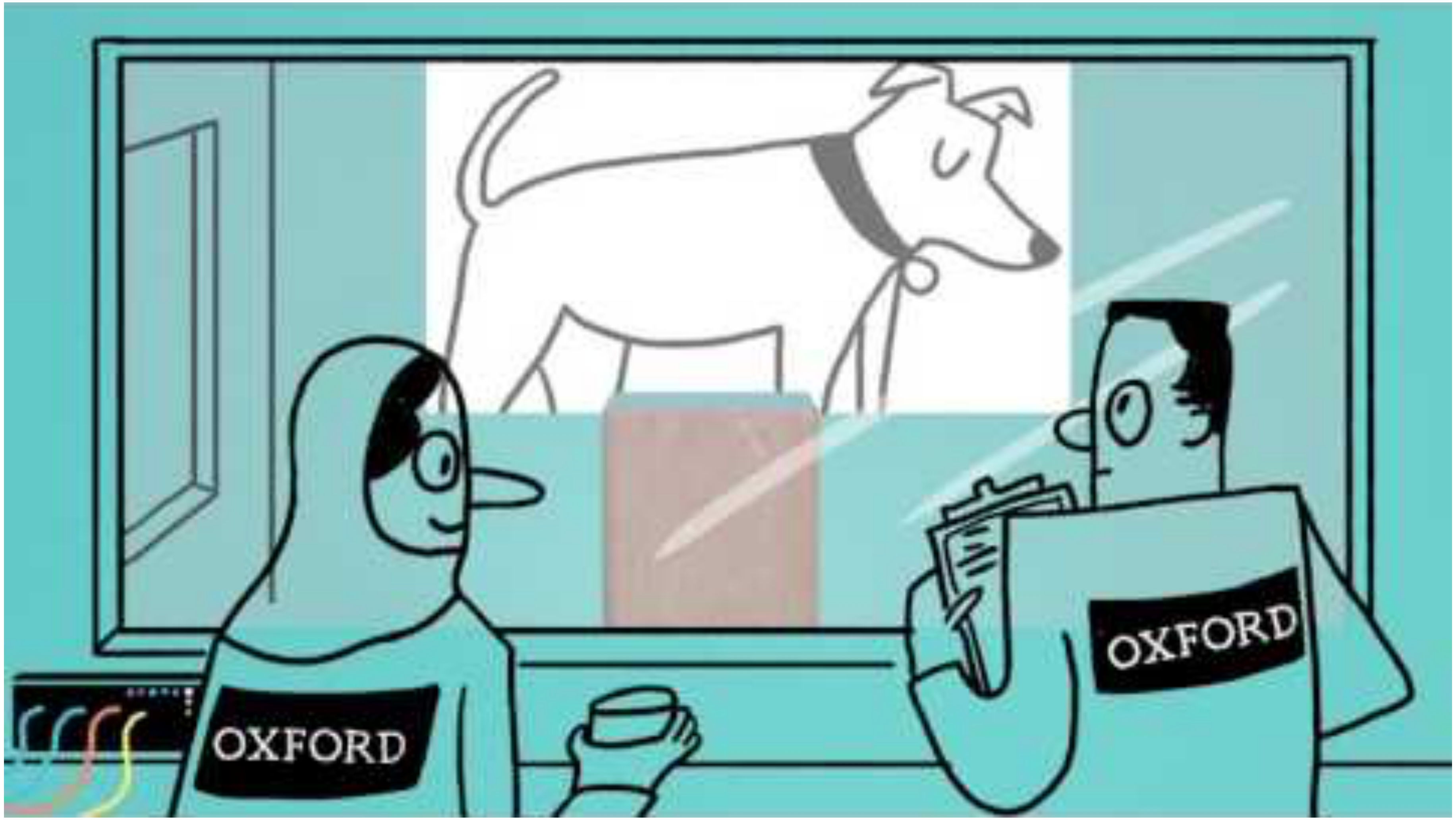
Using Machine Learning for Text Classification

SCHOOL OF INFOCOMM

What is Machine Learning?



https://youtu.be/f_uwKZIAeM0

Acknowledgement & Thanks

Materials and exercises for this lesson are adapted from Intel AI Developer Program.

https://software.intel.com/en-us/ai

Types of Machine Learning

Supervised

Data points have known outcome. We train the model with data. We feed the model with correct answers. Model learns and finally predicts new data's outcome.

Unsupervised

Data points have unknown outcome. Data is given to the model. Right answers are not provided to the model. The model makes sense of the data given to it.

Can reveal something you were probably not aware of in the given dataset.

Text Classification

Text classification is the process of assigning tags, labels or categories to text according to its content

Forms of labels includes binary, categorical, hierarchies of labels. Applications include:

- sentiment analysis
- intent detection
- Support ticket prioritization
- Language detection

Common ML Algorithms For Classification

Regression

Naive Bayes

Nearest Neighbour

Support Vector Machine

Decision Trees

Boosted Trees

Random Forest

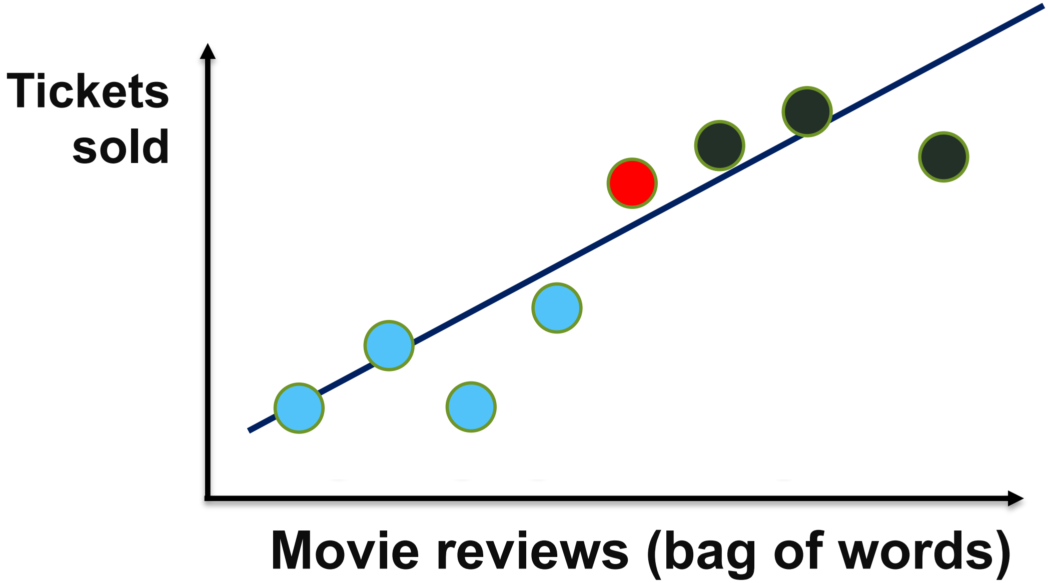
Neural Network

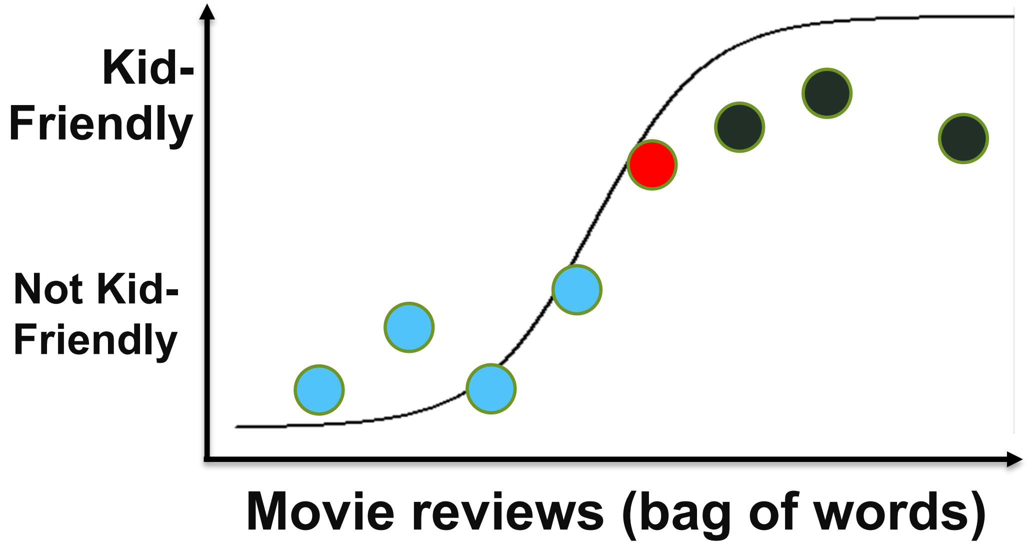
Refresher : Regression

Regression is a **statistical** process to estimate relationships between some variables, with the goal to make some prediction (target) of about some outcome.

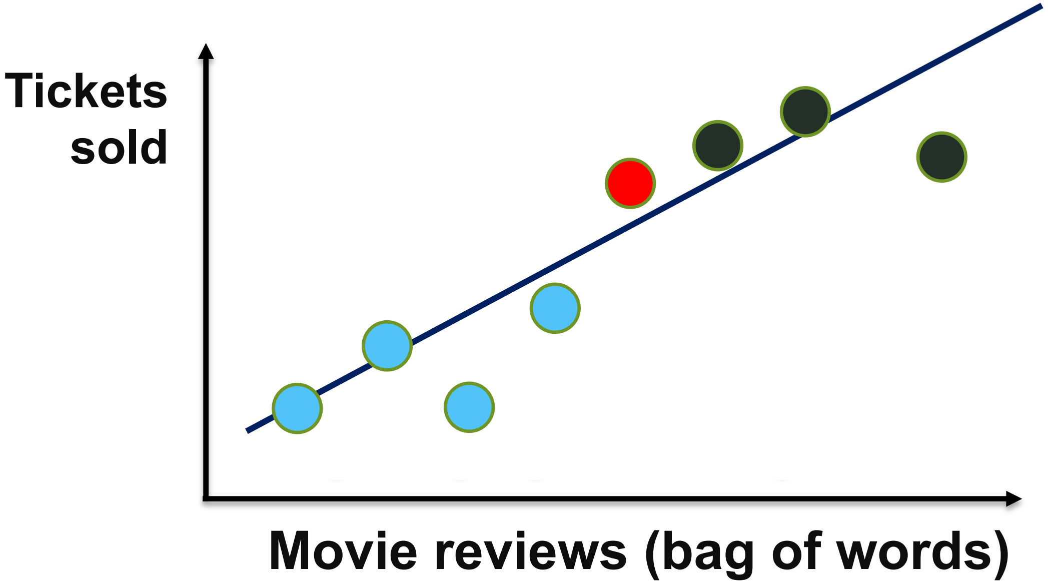
Linear regression is a form of regression where the prediction is a value

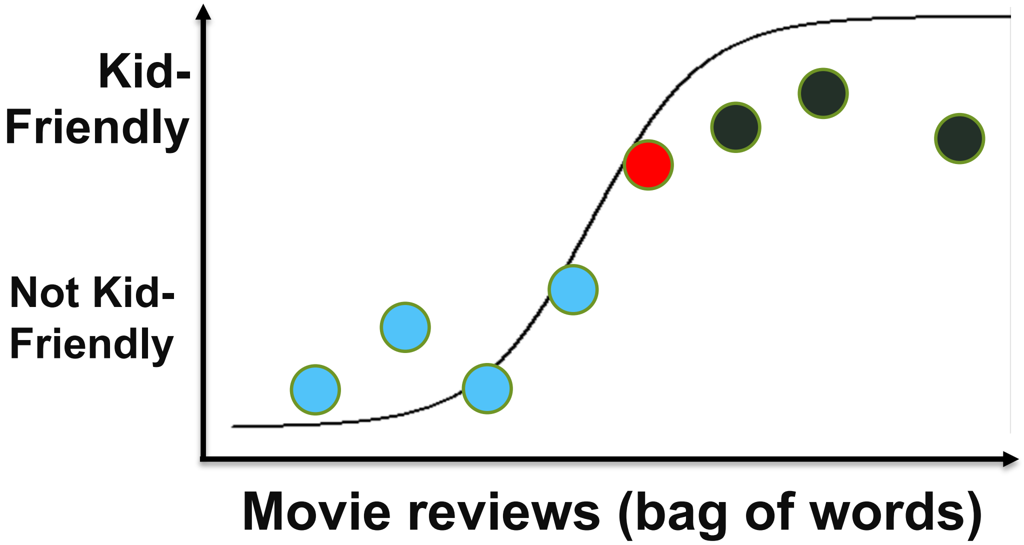
Logistic regression is a form of regression where the prediction is a True or False





Refresher : Regression

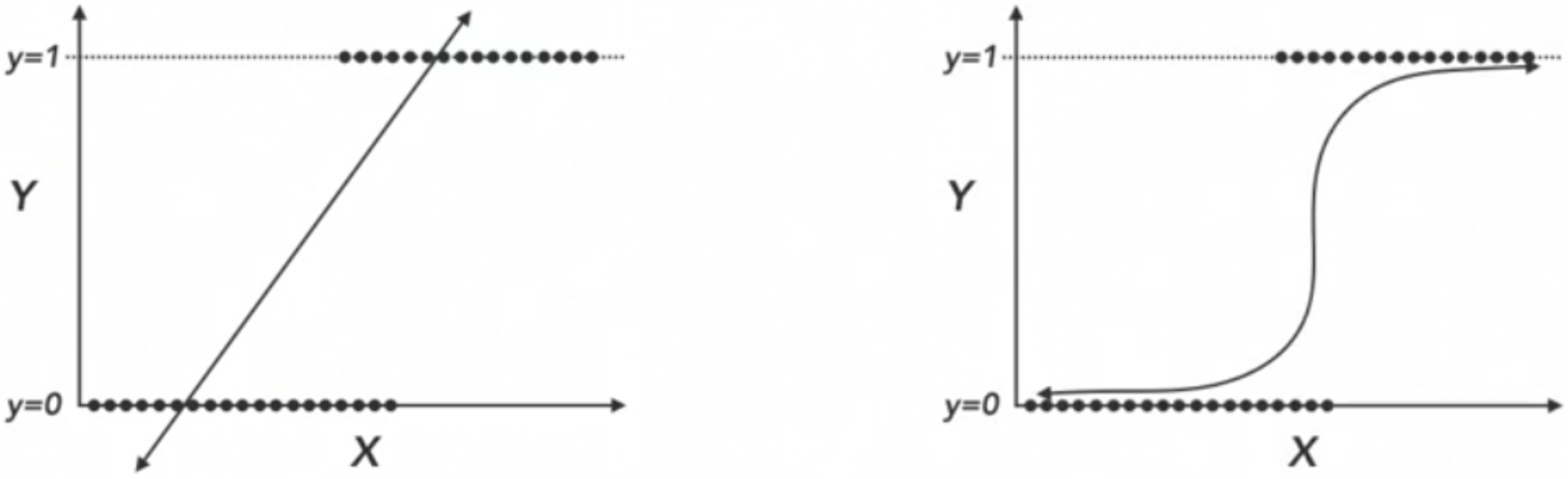




Given the reviews represented by the red dot, linear regression predicts that 1 million box office tickets will be sold

Given the reviews represented by the red dot, logistics regression predicts that the movie is kid-friendly with probability 0.7

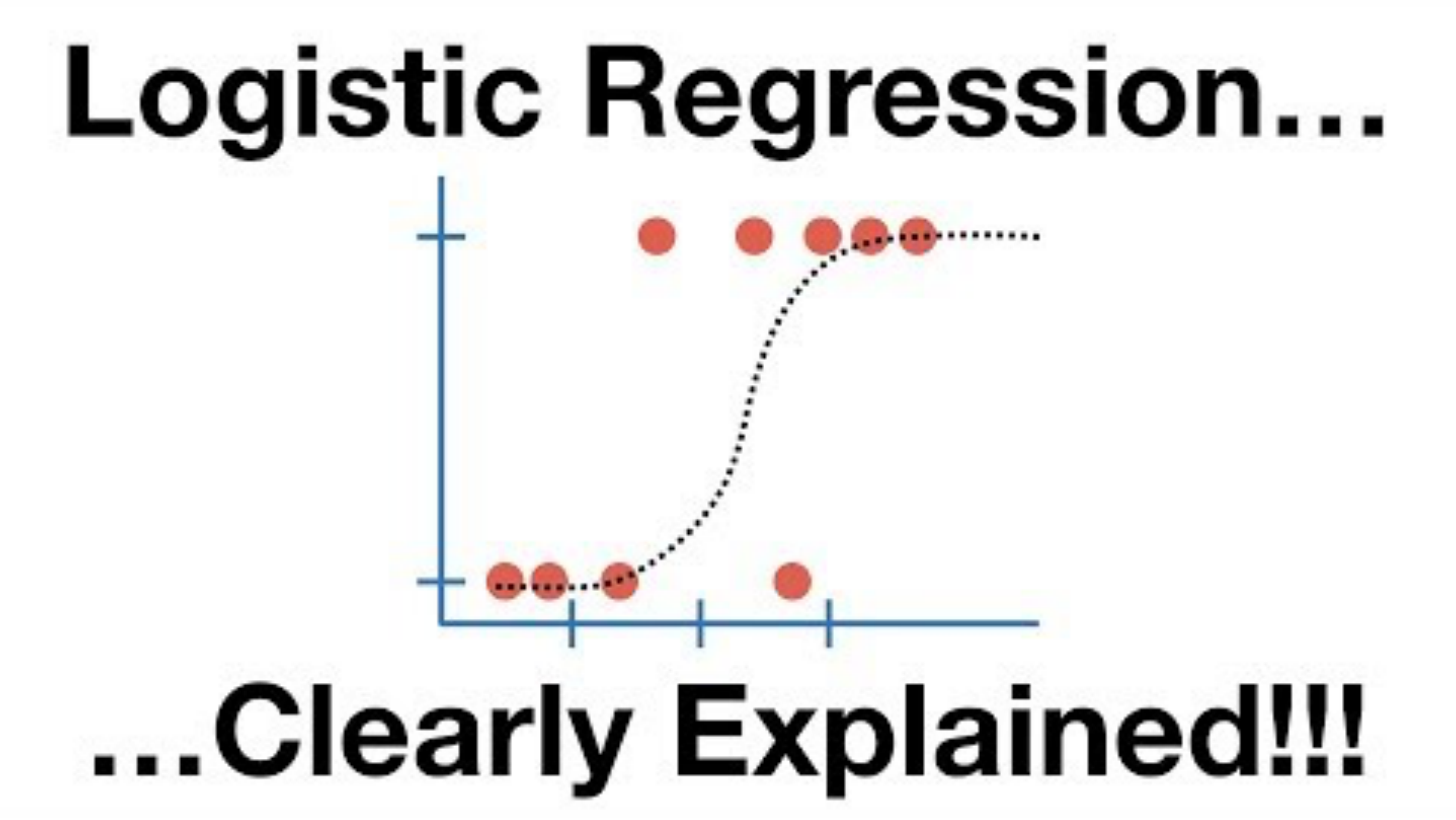
Mathematical Representation



$$y = mx + b$$

$$y = \frac{1}{1 + e^{-(mx + b)}}$$

Learning Video for Logistic Regression



https://youtu.be/yIYKR4sgzI8

Refresher : Naive Bayes

Naive Bayes is a family of probabilistic algorithms that take advantage of **Bayes Theorem** to predict the tag of a text (like a piece of news or a customer review).

Bayes Theorem – what is the probability that something (A) will happen given that something (B) else had happened ?

"Naive" assumption of independence events - when two or more events occur, neither has any effect on the other

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

$$P(X \text{ and } Y \text{ and } Z) = P(X) \times P(Y) \times P(Z)$$

Refresher: Naive Bayes

TEXT	LABEL
A great game	sports
The concert is over	not sport
A fair match	sport
The people cast their votes	not sport
It was a close election	not sport

What is the label for "A fair game" ?

Find :
(a) P(sport | "A fair game")
(b) P(not sport | "a fair game”)

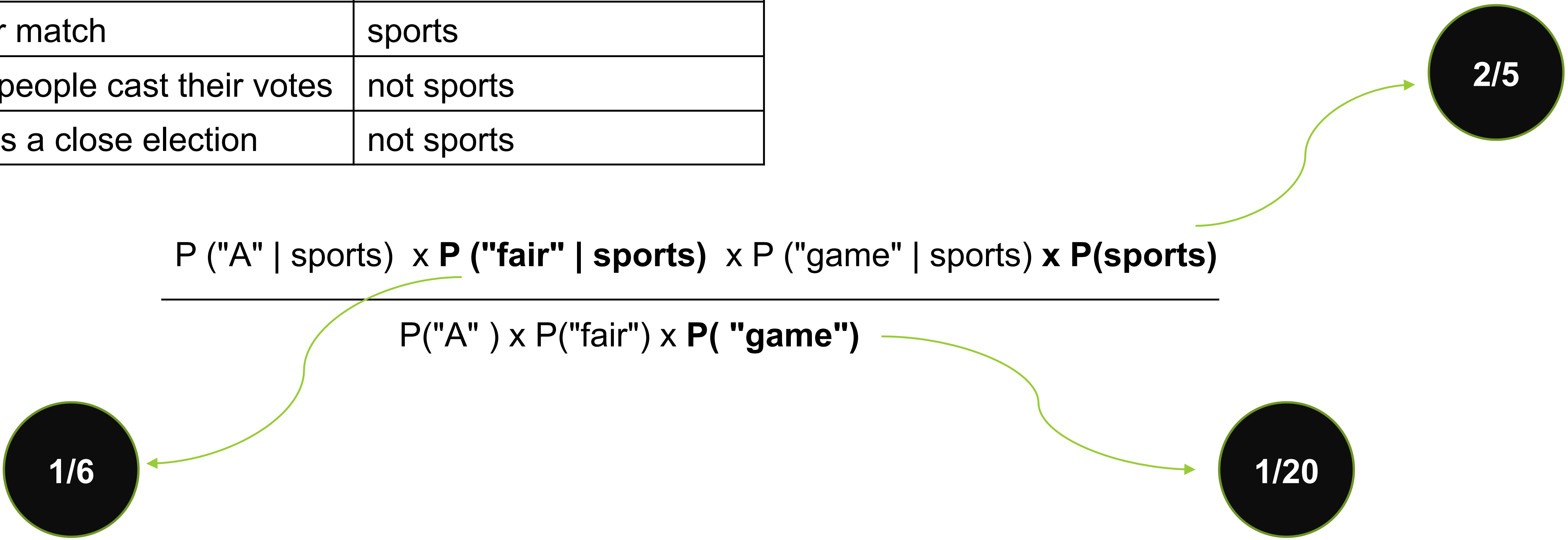
If (a) is the larger value, then the label is **sports**

$$P(\text{sport} \mid \text{"A fair game"}) \xrightarrow{\text{Bayes Theorem}} \frac{P\ (\text{"A fair game"} \mid \text{sport})\ X\ \ P(\text{sport})}{P(\text{"a fair game"})}$$

$$\xrightarrow{\text{Naïve}} \frac{P\ (\text{"A"} \mid \text{sport})\ \ x\ P\ (\text{"fair"} \mid \text{sports})\ \ x\ P\ (\text{"game"} \mid \text{sport})\ x\ P(\text{sport})}{P(\text{"A"}\ )\ x\ P(\text{"fair"})\ x\ (\ \text{"game"})}$$

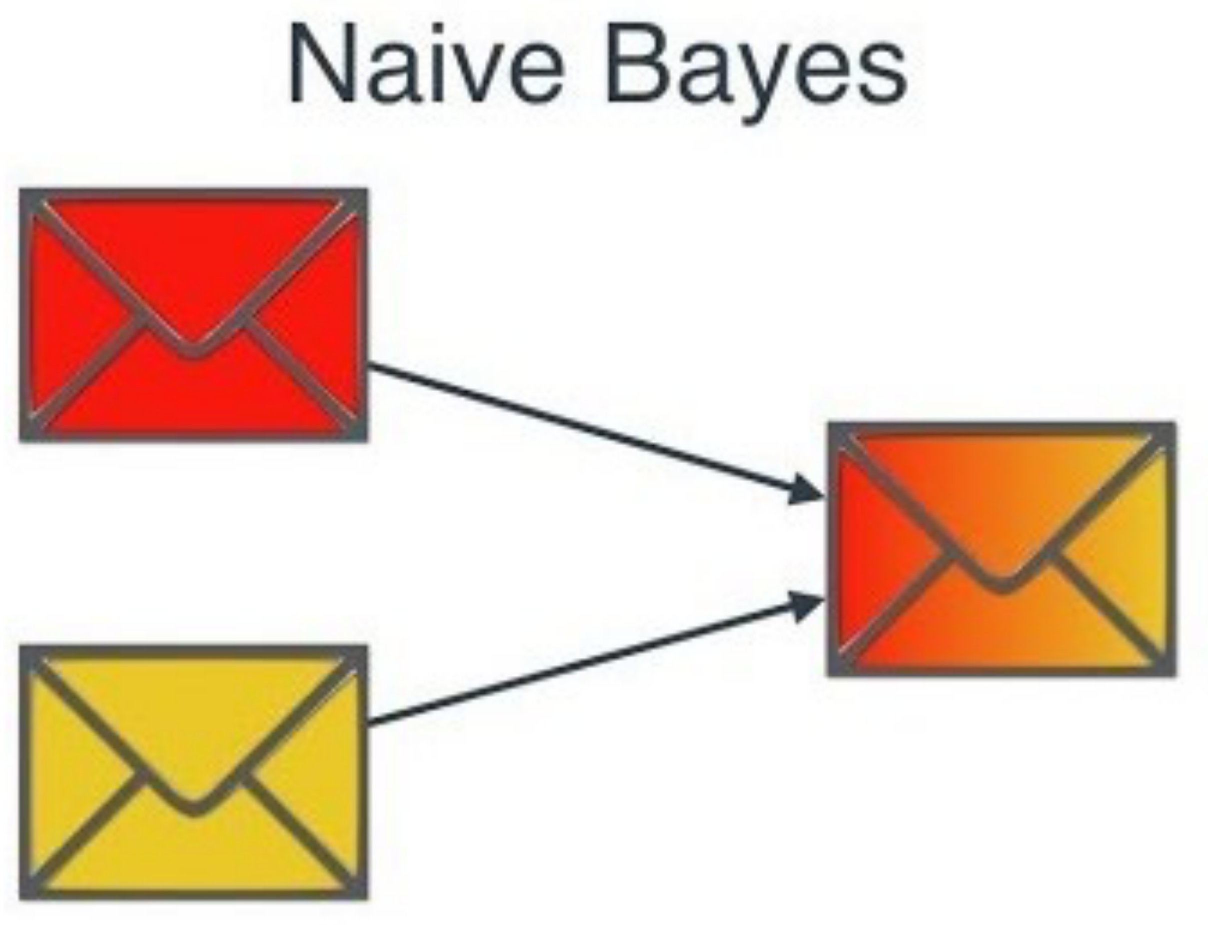
Refresher: Naive Bayes

TEXT	LABEL
A great game	sports
The concert is over	not sports
A fair match	sports
The people cast their votes	not sports
It was a close election	not sports



$$\frac{P(\text{"A"} \mid \text{sports}) \times \mathbf{P(\textbf{"fair"} \mid \textbf{sports})} \times P(\text{"game"} \mid \text{sports}) \times \mathbf{P(\textbf{sports})}}{P(\text{"A"}) \times P(\text{"fair"}) \times \mathbf{P(\textbf{"game"})}}$$

Learning Video for Naive Bayes

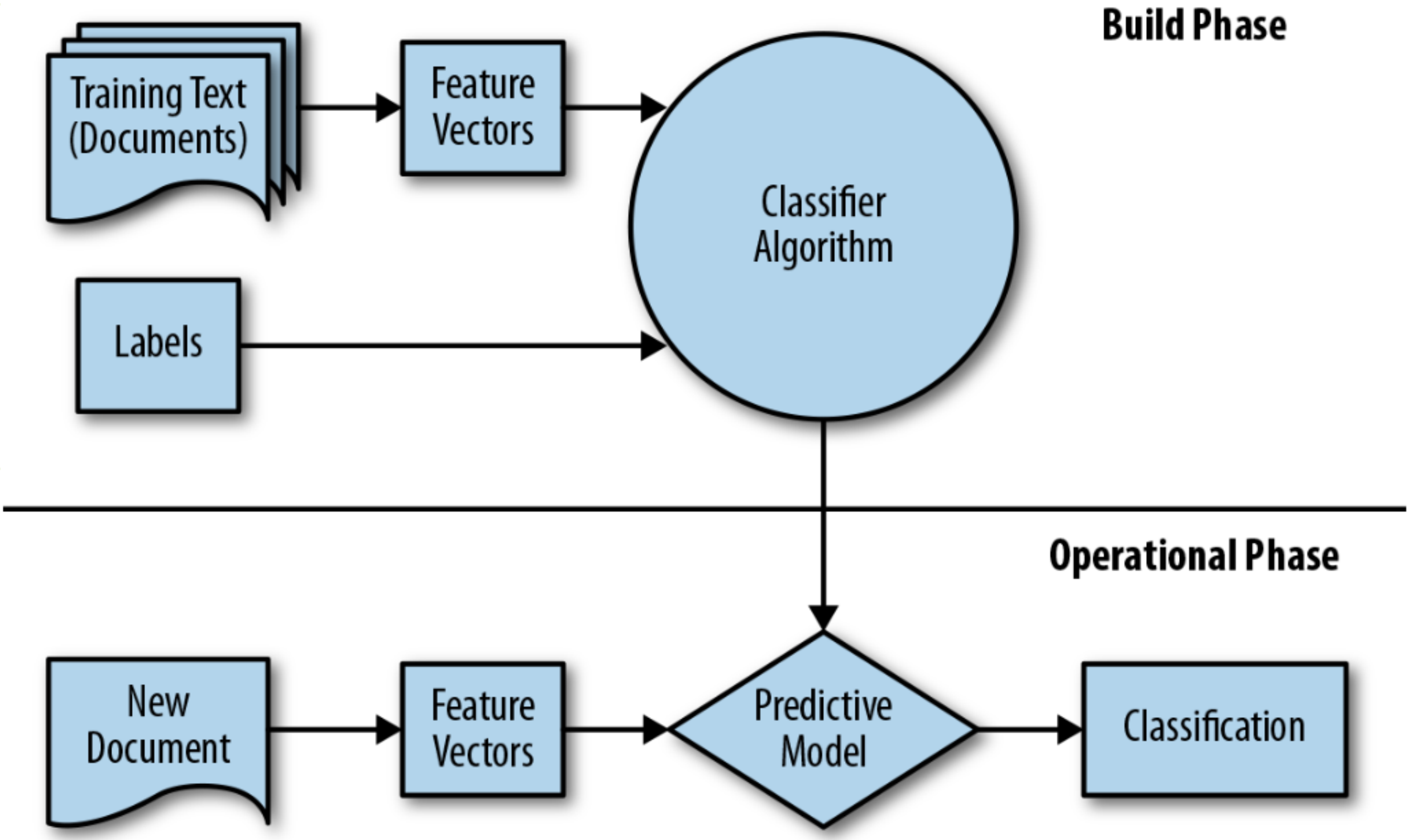


https://youtu.be/Q8l0Vip5YUw

Workflow

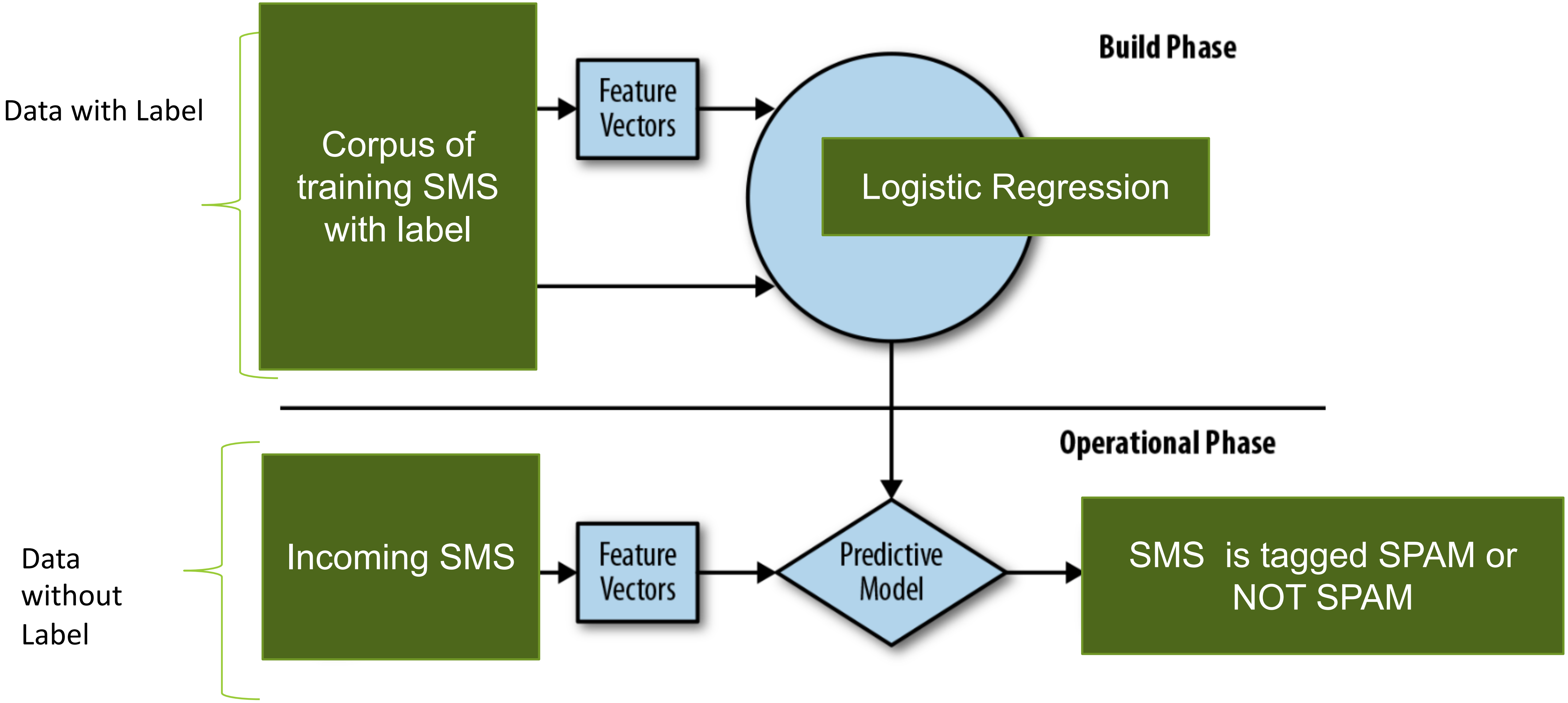
Data with Label

Data without Label



Credit: Applied Text Analysis with Python by Benjamin Bengfort, Rebecca Bilbro, and Tony Ojeda (O'Reilly). 978-1-491-96304-3.

Example



Data with Label

Corpus of training SMS with label

Feature Vectors

Build Phase

Logistic Regression

Operational Phase

Data without Label

Incoming SMS

Feature Vectors

Predictive Model

SMS is tagged SPAM or NOT SPAM

Build Phase

Steps for classification with NLP

1. Prepare the data: Read in labelled data and preprocess the data
2. Split the data: Separate inputs and outputs into a training set and a test set, respectively
3. Numerically encode inputs: Use either Count Vectorizer or TF-IDF Vectorizer
4. Fit a model: Fit a model on the training data and apply the fitted model to the test set
5. Evaluate the model: Decide how good the model is by calculating various error metrics
6. Save the model: Output the model and vectorizer to external files

Step 1: Prepare the data

A classic use of text analytics is to flag messages as spam

Below is data from the SMS Spam Collection Data, which is a set of over 5K English text messages that have been labeled as spam or ham (legitimate)

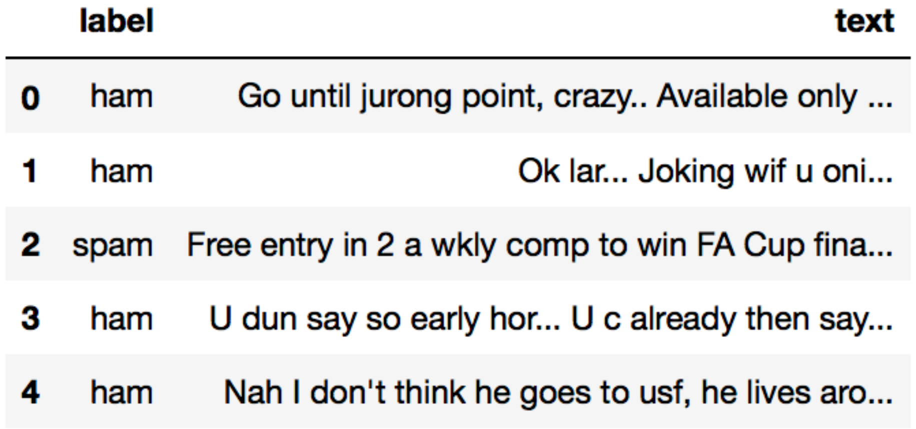
Text Message	Label
Nah I don't think he goes to usf, he lives around here though	ham
Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	spam
WINNER!! As a valued network customer you have been selected to receivea £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.	spam
I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today.	ham
I HAVE A DATE ON SUNDAY WITH WILL!!	ham
…	…

Step 1: Prepare the data [Code]

Input:

```
# make sure the data is labeled
import pandas as pd

data = pd.read_table('SMSSpamCollection.txt', header=None)
data.columns = ['label', 'text']
print(data.head()) # print function requires Python 3
```

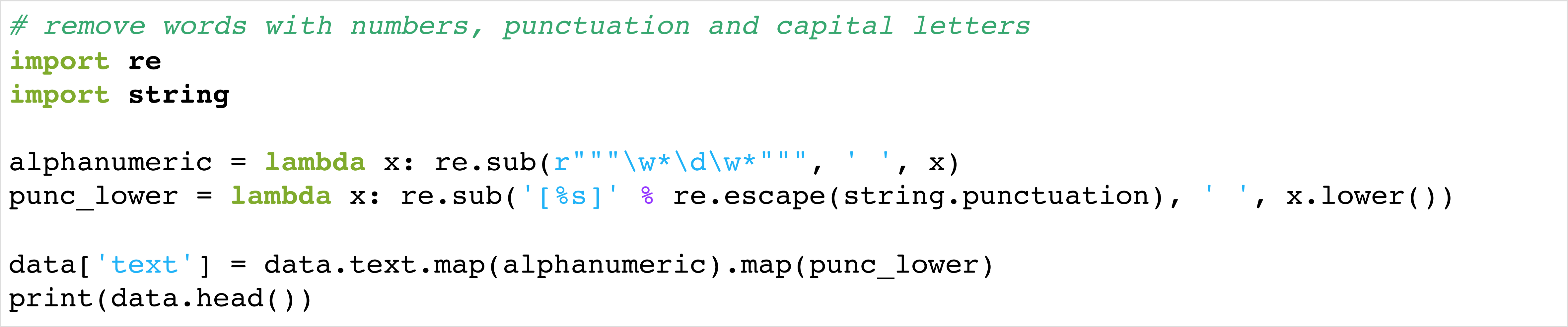
Output:



	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

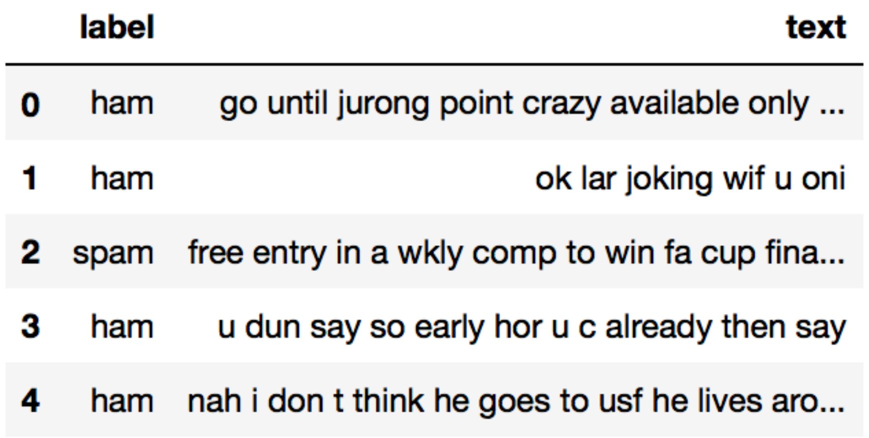
Step 1: Prepare the data [Code]

Input:



```
# remove words with numbers, punctuation and capital letters
import re
import string

alphanumeric = lambda x: re.sub(r"""\w*\d\w*""", ' ', x)
punc_lower = lambda x: re.sub('[%s]' % re.escape(string.punctuation), ' ', x.lower())

data['text'] = data.text.map(alphanumeric).map(punc_lower)
print(data.head())
```

Output:



	label	text
0	ham	go until jurong point crazy available only ...
1	ham	ok lar joking wif u oni
2	spam	free entry in a wkly comp to win fa cup fina...
3	ham	u dun say so early hor u c already then say
4	ham	nah i don t think he goes to usf he lives aro...

Step 2: Split the data (into inputs and outputs)

To fit a model, the data needs to be split into inputs and outputs

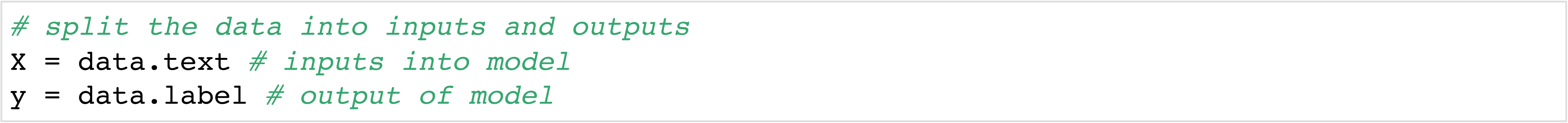
The inputs and output of these models have various names

- Inputs: Features, Predictors, Independent Variables, X’s
- Outputs: Outcome, Response, Dependent Variable, Y

#	label	congrats	eat	tonight	winner	chicken	dinner	wings
0	ham	0	1	0	0	0	0	0
1	ham	0	1	1	0	0	0	0
2	spam	0	0	0	1	0	0	0
.	…	…	…	…	…	…	…	…

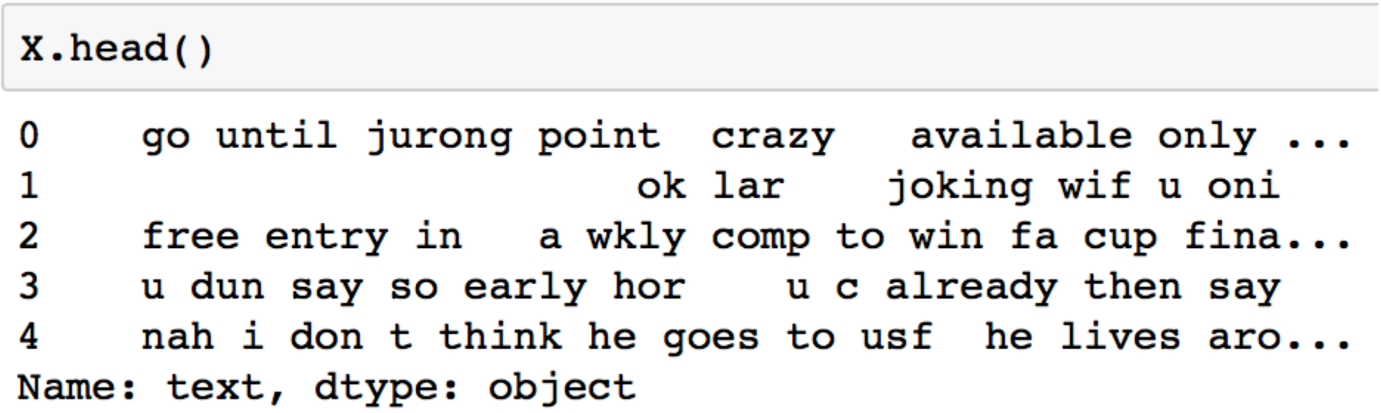
Step 2: Split the data [Code]

Input:



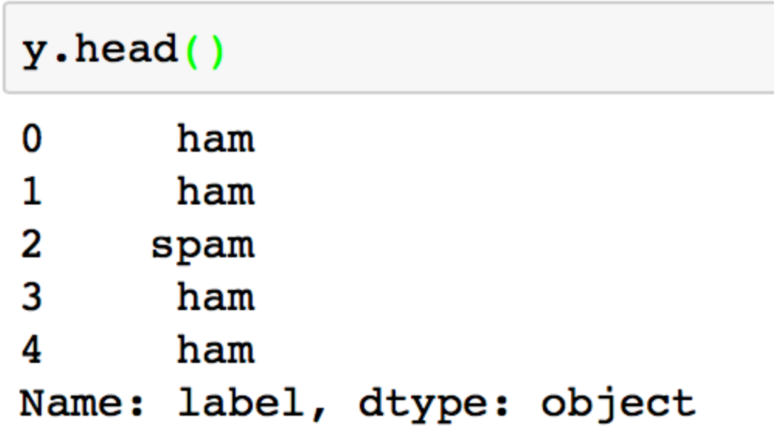
```
# split the data into inputs and outputs
X = data.text # inputs into model
y = data.label # output of model
```

Output:



```
X.head()
```

```
0     go until jurong point  crazy   available only ...
1                         ok lar    joking wif u oni
2     free entry in   a wkly comp to win fa cup fina...
3     u dun say so early hor    u c already then say
4     nah i don t think he goes to usf  he lives aro...
Name: text, dtype: object
```



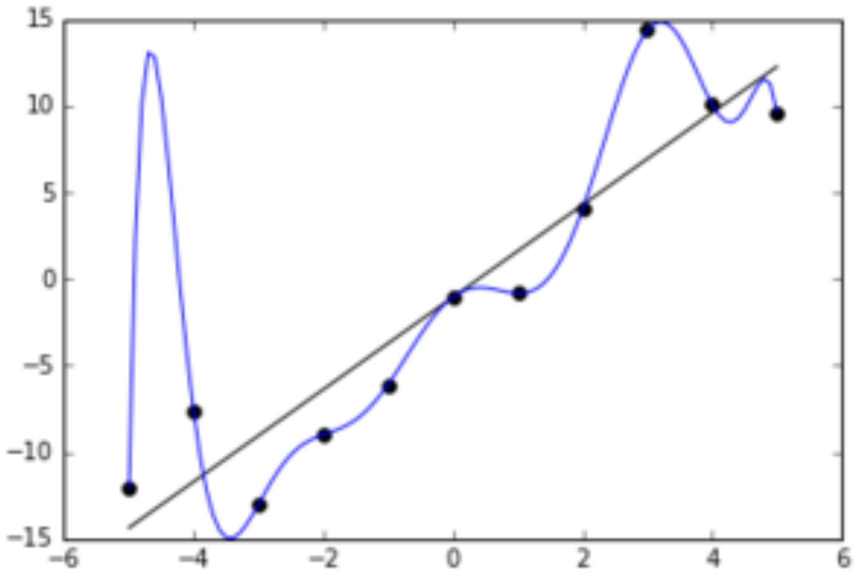
```
y.head()
```

```
0      ham
1      ham
2     spam
3      ham
4      ham
Name: label, dtype: object
```

Step 2: Split the data (into a training and test set)

Why do we need to split data into training and test sets?

- Let’s say we had a data set with 100 observations and we found a model that fit the data perfectly
- What if you were to use that model on a brand new data set?



Blue = Overfitting

Black = Correct

Credit: : https://en.wikipedia.org/wiki/Overfitting

Step 2: Split the data (into a training and test set)

To prevent the issue of overfitting, we divide observations into two sets

- A model is fit on the training data and it is evaluated on the test data
- This way, you can see if the model generalizes well

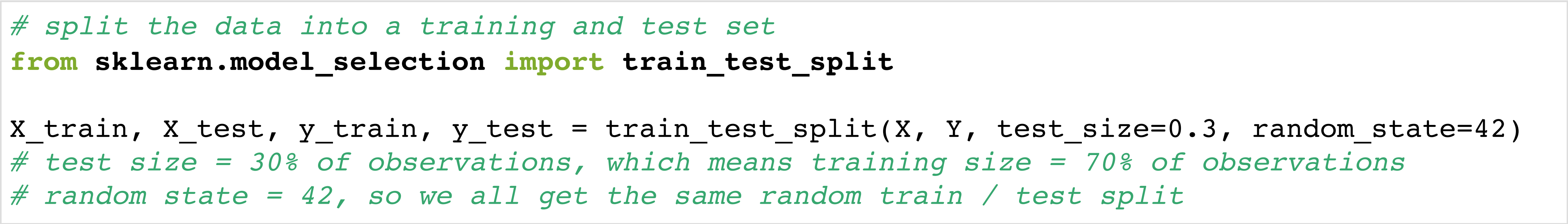
	label	congrats	eat	tonight	winner	chicken	dinner	wings
0	ham	0	1	0	0	0	0	0
1	ham	0	1	1	0	0	0	0
2	spam	0	0	0	1	0	0	0
3	spam	1	0	0	0	0	0	0
4	ham	0	0	0	0	0	1	0
5	ham	0	0	1	0	0	0	0
6	ham	0	0	0	0	0	0	0
7	spam	0	0	0	0	0	0	0
8	ham	0	0	0	0	0	1	0
9	ham	0	0	0	0	1	1	0
10	spam	0	0	0	0	0	0	0
11	ham	0	0	0	0	0	0	1

Training Set (70-80%)

Test Set (20-30%)

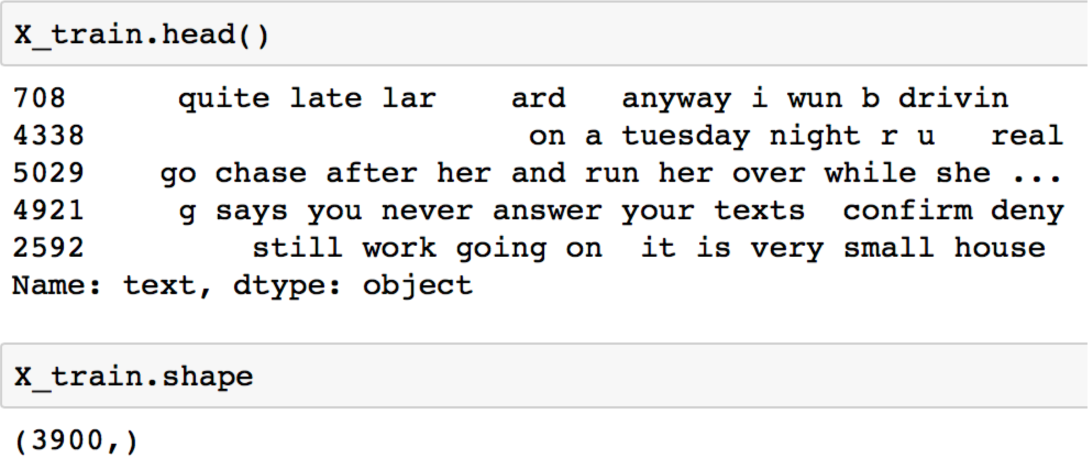
Step 2: Split the data [Code]

Input:

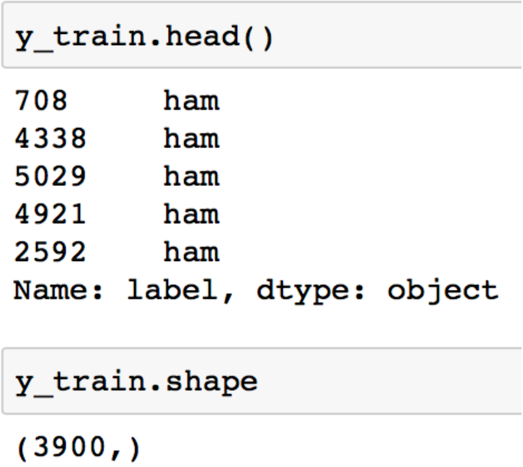


```
# split the data into a training and test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
# test size = 30% of observations, which means training size = 70% of observations
# random state = 42, so we all get the same random train / test split
```

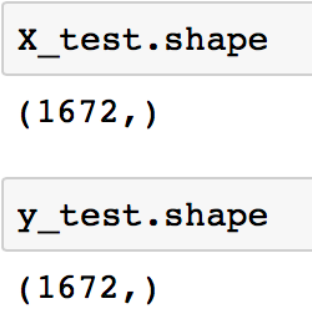
Output:



```
X_train.head()

708        quite late lar     ard    anyway i wun b drivin
4338                            on a tuesday night r u   real
5029     go chase after her and run her over while she ...
4921      g says you never answer your texts  confirm deny
2592           still work going on  it is very small house
Name: text, dtype: object

X_train.shape

(3900,)
```



```
y_train.head()

708       ham
4338      ham
5029      ham
4921      ham
2592      ham
Name: label, dtype: object

y_train.shape

(3900,)
```



```
X_test.shape

(1672,)

y_test.shape

(1672,)
```

Step 3: Numerically encode the input data [Code]

Input:

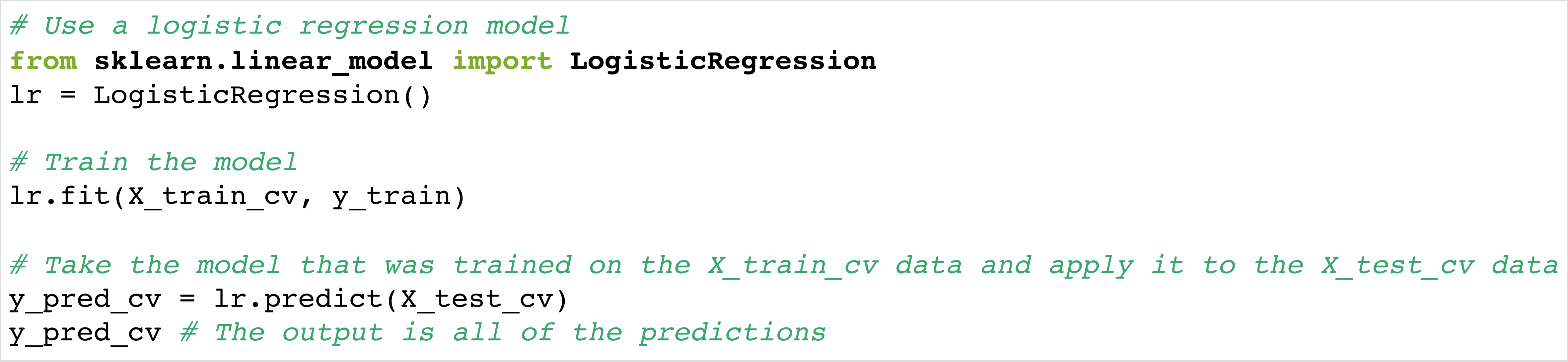
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(stop_words=‘english’)

X_train_cv = cv.fit_transform(X_train) # fit_transform learns the vocab and one-hot encodes
X_test_cv  = cv.transform(X_test) # transform uses the same vocab and one-hot encodes

# print the dimensions of the training set (text messages, terms)
print(X_train_cv.toarray().shape)
```

Output:

```
(3900, 6103)
```

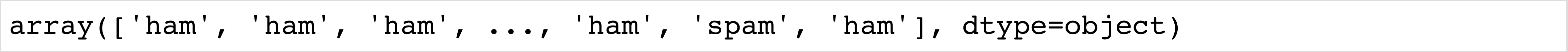
Step 4: Fit model and predict outcomes [Code]

Input:



```
# Use a logistic regression model
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()

# Train the model
lr.fit(X_train_cv, y_train)

# Take the model that was trained on the X_train_cv data and apply it to the X_test_cv data
y_pred_cv = lr.predict(X_test_cv)
y_pred_cv # The output is all of the predictions
```

Output:



```
array(['ham', 'ham', 'ham', ..., 'ham', 'spam', 'ham'], dtype=object)
```

Step 5: Evaluate the model

After fitting a model on the training data and predicting outcomes for the test data, how do you know if the model is a good fit?

		Predicted	
		FALSE	TRUE
Actual	FALSE	True Negative	False Positive
	TRUE	False Negative	True Positive

True Positive: Actual label is SPAM, the prediction is SPAM

True Negative: Actual label is NOT SPAM, prediction is NOT SPAM

False Positive: Actual label is NOT SPAM, predict is SPAM

False Negative: Actual label is SPAM, prediction is NOT SPAM

Step 5: Evaluate the model

#	Actual	Predicted	Result
1	ham	ham	true negative
2	ham	ham	true negative
3	spam	spam	true positive
4	spam	spam	true positive
5	ham	ham	true negative
6	ham	spam	false positive
7	ham	ham	true negative
8	ham	ham	true negative
9	ham	ham	true negative
10	spam	spam	true positive

Confusion Matrix

		Predicted	
		ham	spam
Actual	ham	True Negative (6)	False Positive (1)
	spam	False Negative (0)	True Positive (3)

Step 5: Evaluate the model

Error Metrics

- Accuracy: Overall, how often is the classifier correct

  (TP + TN) / All

- Precision: When a positive value is predicted, how often is the prediction correct?

  TP / (TP + FP)

- Recall / Sensitivity: When the actual value is positive, how often is the prediction correct?

  TP / (TP + FN)

- F1 Score: Harmonic mean. if the F1 score is high, both precision and recall of the classifier indicate good results.

  2*(P*R)/(P+R)

Step 5: Evaluate the model

After fitting a model on the training data and predicting outcomes for the test data, how do you know if the model is a good fit?

Error Metrics

- Accuracy = (TP + TN) / All = 0.9
- Precision = TP / (TP + FP) = 0.75
- Recall = TP / (TP + FN) = 1
- F1 Score = 2*(P*R)/(P+R) = 0.86

Confusion Matrix

		Predicted	
		ham	spam
Actual	ham	True Negative (6)	False Positive (1)
	spam	False Negative (0)	True Positive (3)

Step 5: Evaluate the model

		Predicted	
		FALSE	TRUE
Actual	FALSE	True Negative	False Positive
	TRUE	False Negative	True Positive

Question :
Is False Positive (FP) or False Negative (FN) more important to reduce?

SPAM detector (positive class is "SPAM"). Is it better to:
(a) Flag a job offer email as "SPAM" and move to the SPAM INBOX? (FP)
(b) Flag a "get rich" marketing email as "NOT SPAM" and keep in the INBOX (FN)

Optimization strategy:

- If False Negative is costly, optimize Recall
- If False Positive is costly, optimize Precision

Step 5: Evaluate the model [Code]

Input:

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

cm = confusion_matrix(y_test, y_pred_cv)
sns.heatmap(cm, xticklabels=['predicted_ham', ‘predicted_spam'], yticklabels=['actual_ham', ‘actual_spam'],
annot=True, fmt='d', annot_kws={'fontsize':20}, cmap=“YlGnBu");

true_neg, false_pos = cm[0]
false_neg, true_pos = cm[1]

accuracy = round((true_pos + true_neg) / (true_pos + true_neg + false_pos + false_neg),3)
precision = round((true_pos) / (true_pos + false_pos),3)
recall = round((true_pos) / (true_pos + false_neg),3)
f1 = round(2 * (precision * recall) / (precision + recall),3)

print('Accuracy: {}'.format(accuracy))
print('Precision: {}'.format(precision))
print('Recall: {}'.format(recall))
print(‘F1 Score: {}’.format(f1))
```

Step 6: Save the Model [Code]

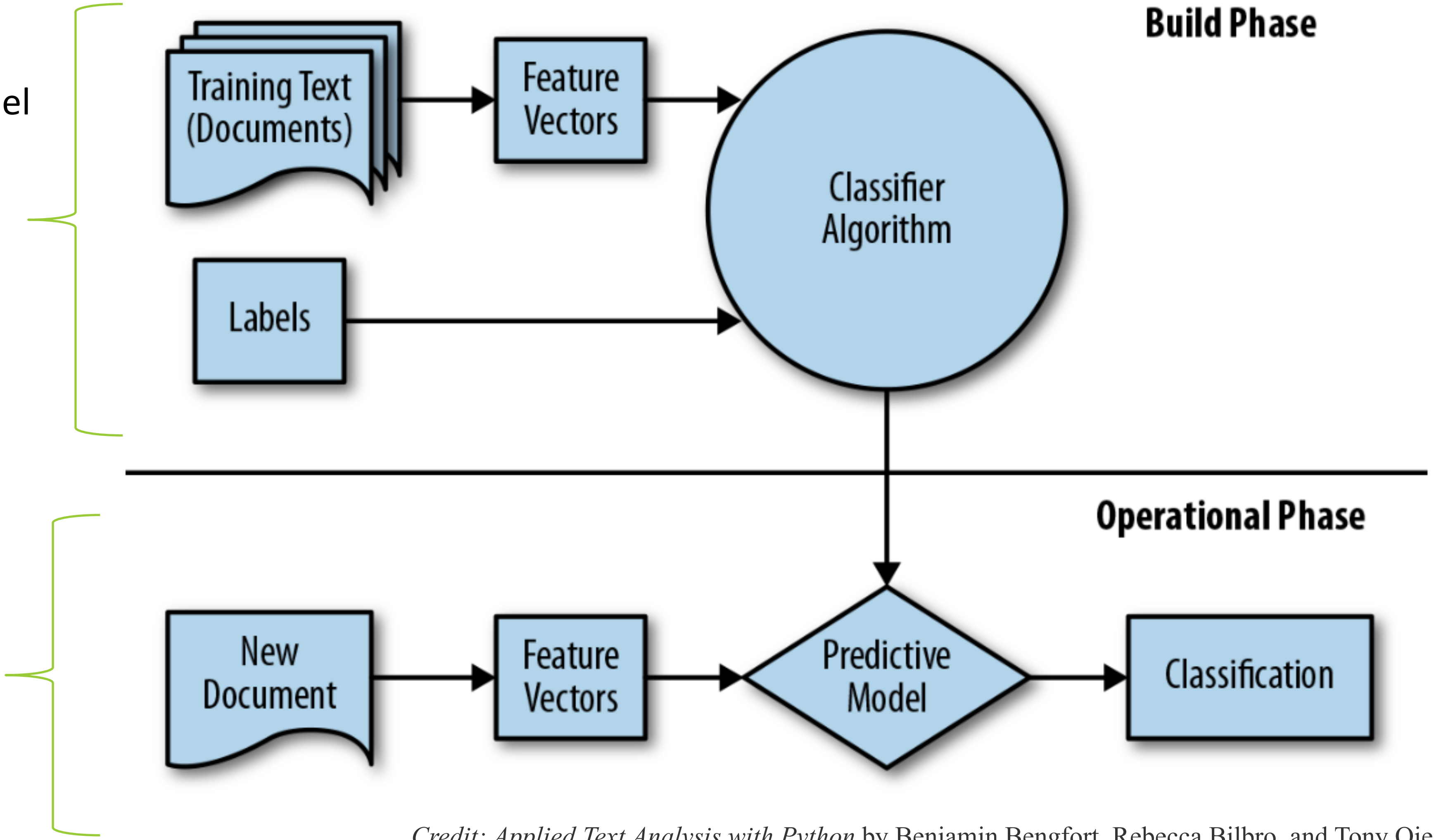
Input:

```
import pickle
from datetime import datetime

time = datetime.now().strftime("%Y-%m-%d")
path1 = 'classifier-{}.pkl'.format(time)

path2 = 'countvectoriser-{}.pkl'.format(time)
with open(path1, 'wb') as f1:
    pickle.dump(lr, f1)

with open(path2, 'wb') as f2:
    pickle.dump(cv, f2)
```

Workflow

Data with Label

Data without Label



Credit: *Applied Text Analysis with Python* by Benjamin Bengfort, Rebecca Bilbro, and Tony Ojeda (O'Reilly). 978-1-491-96304-3.

Operational Phase

Steps for operation the model

1. Reload the model : Load in the regression model that was saved during the modelling stage
2. Reload the Vectorizer: Load in the vectorizer that was used to encode the training set
3. Preprocess the new text: Clean the input data in the SAME way as it was done during modelling
4. Numerically encode the input : Convert the text to vectors
5. Predict the label : Reuse the model to predict the label

Step 1: Reload the model [Code]

Input:

```
path1 = "classifier-2020-02-02.pkl"
with open(path1, 'rb') as f:
    model = pickle.load(f)
```

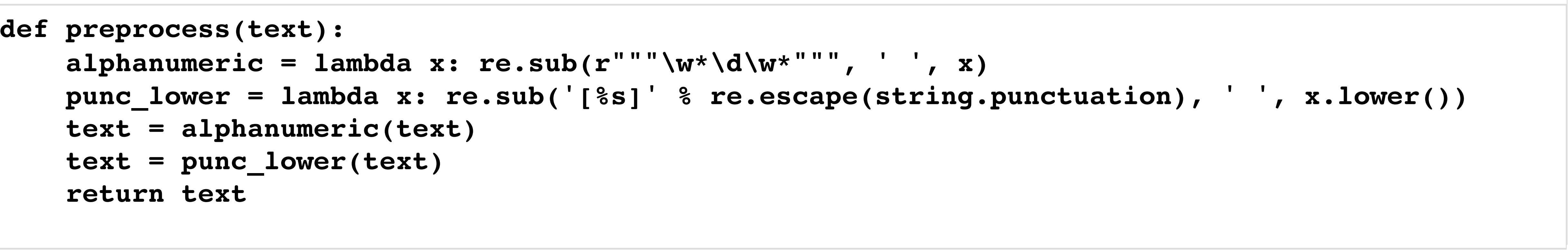
Step 2: Reload the Vectorizer [Code]

Input:

```
path2 = "countvectoriser-2020-02-02.pkl"
with open(path2, 'rb') as f:
    trained_cv = pickle.load(f)
```

Step 3: Pre-process the new text [Code]

Input:



```
def preprocess(text):
    alphanumeric = lambda x: re.sub(r"""\w*\d\w*""", ' ', x)
    punc_lower = lambda x: re.sub('[%s]' % re.escape(string.punctuation), ' ', x.lower())
    text = alphanumeric(text)
    text = punc_lower(text)
    return text
```

It is important to preprocess the input text using the same set of pre-processing tasks used during the training stage. Otherwise, the list of vocabulary may be different.

Step 4: Encode the text [Code]

Input:

```
def encode_text_to_vector(cv, text):
       new_cv = CountVectorizer(stop_words='english', vocabulary=trained_cv.vocabulary_)
       text_vector = new_cv.fit_transform( [text ] )
       return text_vector
```

From the trained vectorizer, we need to retrieve the vocabulary from it and pass the list as a parameter to the new vectorizer.

Be careful to use the same set of parameters as well

The new vectorizer is then use to encode new text and provide a new vector with same vocabulary

Step 5: Predict the label [Code]

Input:

```
new_text = input("Enter the new text > ")
new_text = preprocess(new_text)
new_text_vector = encode_text_to_vector(trained_cv, new_text)
predicted_label = (model.predict(new_text_vector))
print ("The text is  a " + predicted_label)
```

Exercise 1 - Classification of Text with Navie Bayes

What was our original goal

- To build a text classification model using Logistics Regression
- To create an application that uses the model to classify a user input text

Exercise: Can you replicate the earlier example using Naive Bayes?

Refer to Jupyter Notebook:
ex1A-build-phase-text-classification-navie-bayes.ipynb

ex1B-op-phase-text-classification-navie-bayes.ipynb

Naive Bayes: Fit model [Code]

Input:

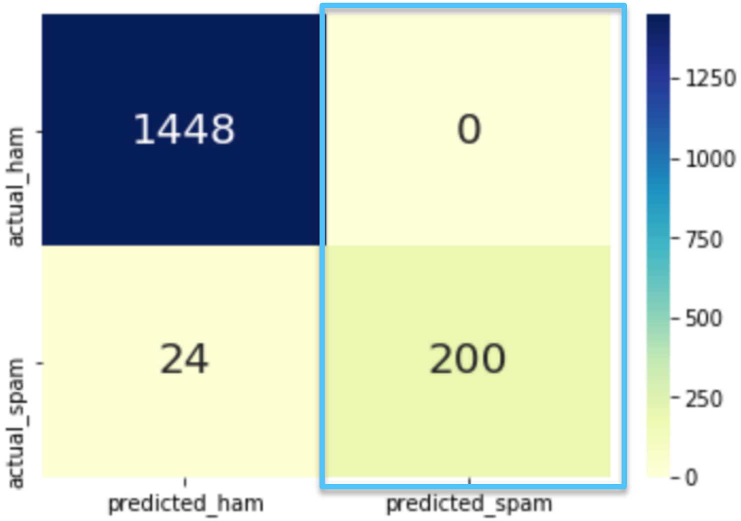
```
# Use a Naive Bayes model
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()

# Train the model
nb.fit(X_train_cv, y_train)

# Take the model that was trained on the X_train_cv data and apply it to the X_test_cv data
y_pred_cv_nb = nb.predict(X_test_cv)
y_pred_cv_nb # The output is all of the predictions
```

Output:

```
array(['ham', 'ham', 'ham', ..., 'ham', 'spam', ‘ham'], dtype='<U4')
```

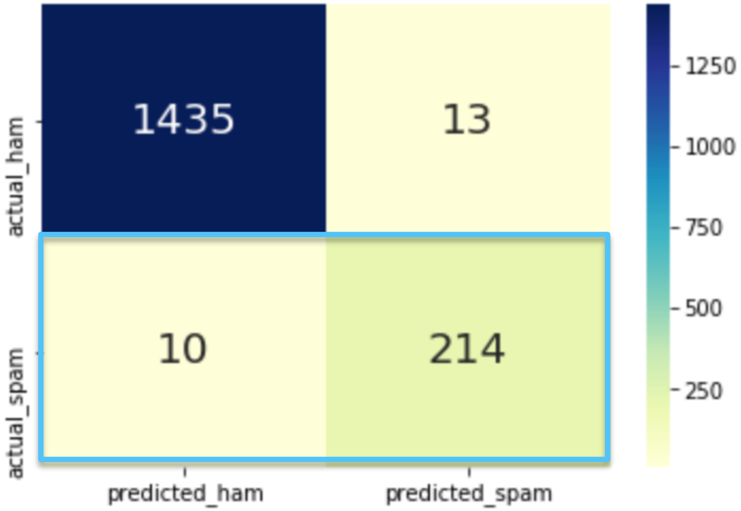
Comparing Models

Logistic Regression



Accuracy: 0.986
Precision: 1.0
Recall: 0.893
F1 Score: 0.943

Naive Bayes



Accuracy: 0.986
Precision: 0.939
Recall: 0.952
F1 Score: 0.945

Which Works Well

Benchmark Dataset: SMS Spam Message
Features: Bag of words with TF-IDF scoring

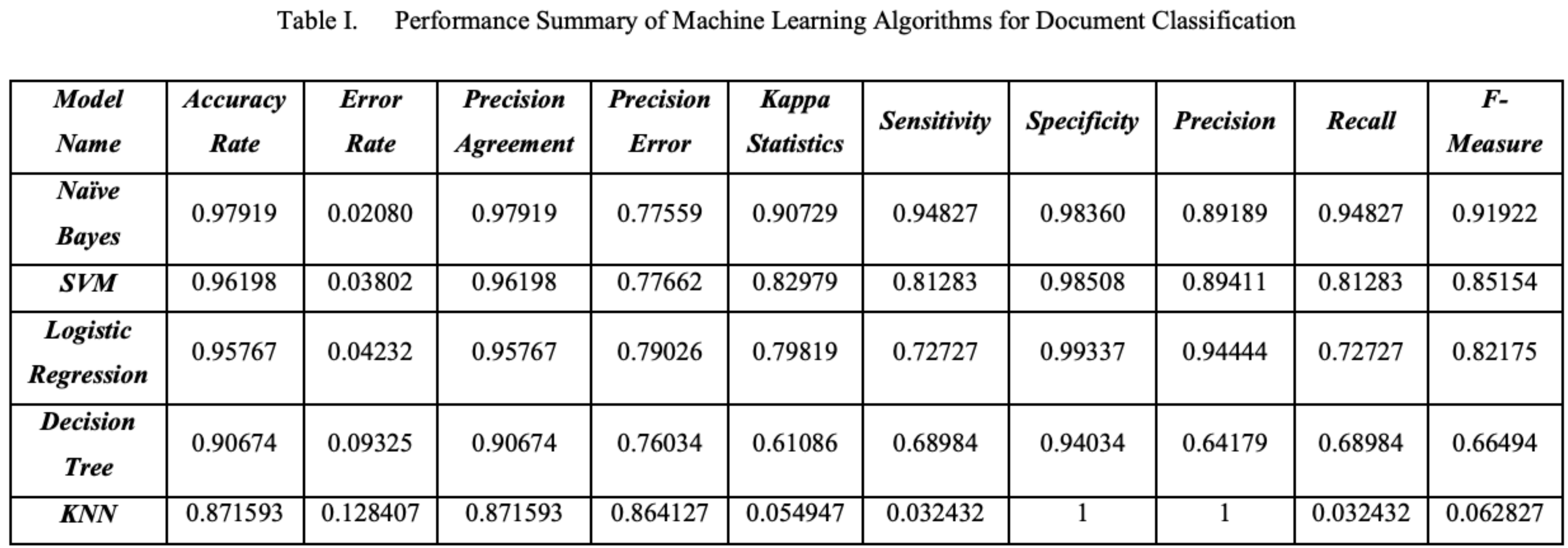


Table I. Performance Summary of Machine Learning Algorithms for Document Classification

Model Name	*Accuracy Rate*	*Error Rate*	*Precision Agreement*	*Precision Error*	*Kappa Statistics*	*Sensitivity*	*Specificity*	*Precision*	*Recall*	*F-Measure*
Naïve Bayes	0.97919	0.02080	0.97919	0.77559	0.90729	0.94827	0.98360	0.89189	0.94827	0.91922
SVM	0.96198	0.03802	0.96198	0.77662	0.82979	0.81283	0.98508	0.89411	0.81283	0.85154
Logistic Regression	0.95767	0.04232	0.95767	0.79026	0.79819	0.72727	0.99337	0.94444	0.72727	0.82175
Decision Tree	0.90674	0.09325	0.90674	0.76034	0.61086	0.68984	0.94034	0.64179	0.68984	0.66494
KNN	0.871593	0.128407	0.871593	0.864127	0.054947	0.032432	1	1	0.032432	0.062827

Source: https://www.ijarcs.info/index.php/Ijarcs/article/view/4699/4173

Exercise 2 - Comparing Models

Now that you know the basics of creating classifications models, you can evaluate what is the most optimal model for a unique dataset

Exercise:

To train different possible models based on combinations of:

- Unigram and bigram tokens
- Count and TF-IDF vectorisation
- Logistic regression and Naive Bayes

Refer to Jupyter Notebook:
ex2-build-and-evalute-classification-models.ipynb

References

- Machine Learning for Natural Language Processing, https://www.lexalytics.com/lexablog/machine-learning-natural-language-processing

- StatQuest: Logistic Regression https://youtu.be/yIYKR4sgzI8

- Machine Learning Cheatsheet: https://github.com/afshinea/stanford-cs-229-machine-learning/blob/master/en/super-cheatsheet-machine-learning.pdf

- Classification of Phishing Email Using Random Forest Machine Learning Technique , https://www.hindawi.com/journals/jam/2014/425731/

- Empirical Evaluation Of Machine Learning Algorithms For Automatic Document Classification, https://www.ijarcs.info/index.php/Ijarcs/article/view/4699/4173

- Intel AI Developer Program, https://software.intel.com/en-us/ai