

# Week 5: Text Classification

Edwin Simpson

Department of Computer Science,  
University of Bristol, UK.

# Lab 5:

How It Relates to the Theory

# CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
from nltk import word_tokenize

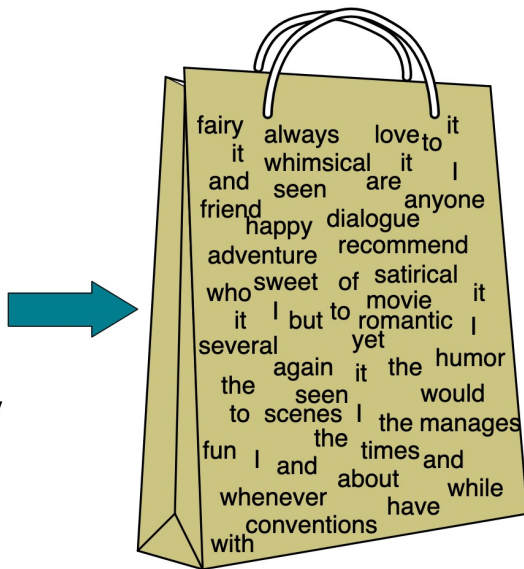
# CountVectorizer can do its own tokenization, but for consistency we want to
# carry on using WordNetTokenizer. We write a small wrapper class to enable this:
class Tokenizer(object):
    def __call__(self, tweets):
        return word_tokenize(tweets)

vectorizer = CountVectorizer(tokenizer=Tokenizer()) # construct the vectorizer

vectorizer.fit(train_tweets) # Learn the vocabulary
X_train = vectorizer.transform(train_tweets) # extract training set bags of words
X_test = vectorizer.transform(test_tweets) # extract test set bags of words
```

# CountVectorizer: Extracts a Bag of Words

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1

```
vectorizer = CountVectorizer(tokenizer=Tokenizer()) # construct the vectorizer

vectorizer.fit(train_tweets) # Learn the vocabulary
X_train = vectorizer.transform(train_tweets) # extract training set bags of words
X_test = vectorizer.transform(test_tweets) # extract test set bags of words
```

# CountVectorizer

- With lemmatisation as well as tokenisation:

```
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer

class LemmaTokenizer(object):

    def __init__(self):
        self.wnl = WordNetLemmatizer()

    def __call__(self, tweets):
        return [self.wnl.lemmatize(self.wnl.lemmatize(self.wnl.lemmatize(tok, pos='n'), pos='v'), pos='a') for tok in word_tokenize(tweets)]

vectorizer = CountVectorizer(tokenizer=LemmaTokenizer())

vectorizer.fit(train_tweets)
X_train = vectorizer.transform(train_tweets)
X_test = vectorizer.transform(test_tweets)

# Print out some of the features in the vocabulary
```

# Naïve Bayes: How does the fit method work?

```
classifier = MultinomialNB()  
classifier.fit(X_train, train_labels)
```

- Answers please:

<https://uob.padlet.org/edwinsimpson/9kef5dkl87ztndbr>

# How does the fit method work? It performs training via maximum likelihood estimation.

```
classifier = MultinomialNB()  
classifier.fit(X_train, train_labels)
```

$$P(y = c) = \frac{\text{num\_docs\_in\_class\_c}}{\text{total\_num\_docs}}$$

$$P(x_i = w | y = c) = \frac{\text{count}(w \text{ in } c) + 1}{\sum_{w' \in V} (\text{count}(w' \text{ in } c) + 1)}$$

# Naïve Bayes: How does predict() work?

```
y_test_pred = classifier.predict(X_test)
```

- Answers please:

<https://uob.padlet.org/edwinsimpson/9kef5dkl87ztndbr>



# How does predict() work? By applying Bayes' rule.

```
y_test_pred = classifier.predict(X_test)
```

$$P(y|\mathbf{x}) = \frac{\prod_{i=1}^N P(x_i|y)P(y)}{\sum_{y'} \prod_{i=1}^N P(x_i|y') P(y')}$$

# How does predict() work? By applying Bayes' rule.

```
y_test_pred = classifier.predict(X_test)
```

$$P(y|\mathbf{x}) = \frac{\prod_{i=1}^N P(x_i|y)P(y)}{\sum_{y'} \prod_{i=1}^N P(x_i|y') P(y')}$$

```
feat_likelihoods = np.exp(classifier.feature_log_prob_)  
###
```

```
# WRITE YOUR CODE HERE
```

```
print(feat_likelihoods[:, vocabulary['happy']])  
print(feat_likelihoods[:, vocabulary['hate']])
```

```
[0.00016529 0.00010482 0.00175175]  
[5.09280976e-04 8.95763610e-05 5.33420957e-05]
```

# Evaluation

```
# WRITE YOUR CODE HERE
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

acc = accuracy_score(test_labels, y_test_pred)
print(f'Accuracy = {acc}')

prec = precision_score(test_labels, y_test_pred, average='macro')
print(f'Precision (macro average) = {prec}')

rec = recall_score(test_labels, y_test_pred, average='macro')
print(f'Recall (macro average) = {rec}')

f1 = f1_score(test_labels, y_test_pred, average='macro')
print(f'F1 score (macro average) = {f1}')

# We can get all of these with a per-class breakdown using classification_report
print(classification_report(test_labels, y_test_pred))
```

```
Accuracy = 0.5891403451644416
Precision (macro average) = 0.5856910631417499
Recall (macro average) = 0.5819831225339609
F1 score (macro average) = 0.5717257637198551
```

	precision	recall	f1-score	support
0	0.66	0.43	0.52	3972
1	0.61	0.68	0.64	5937
2	0.49	0.64	0.56	2375
accuracy			0.59	12284
macro avg	0.59	0.58	0.57	12284
weighted avg	0.60	0.59	0.58	12284

# Logistic Regression: What happens when we call fit()?

```
from sklearn.linear_model import LogisticRegression  
  
classifier = LogisticRegression()  
classifier.fit(X_train, train_labels)
```

- Answers please:

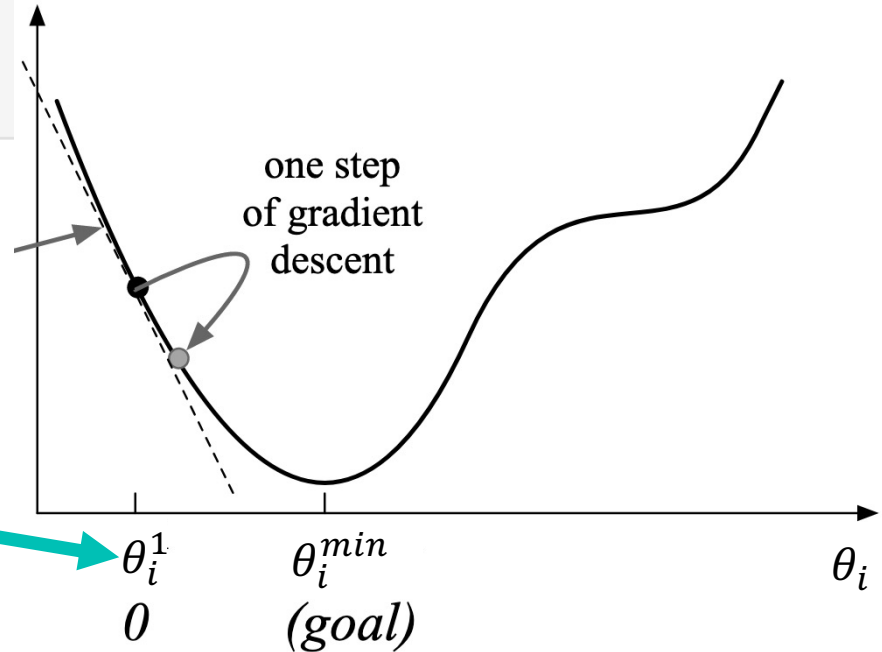
<https://uob.padlet.org/edwinsimpson/9kef5dkl87ztndbr>

# Logistic Regression: What happens when we call fit()?

```
from sklearn.linear_model import LogisticRegression
```


```
classifier = LogisticRegression()  
classifier.fit(X_train, train_labels)
```

$$\frac{1}{1 + e^{-\sum_{i=1}^N \theta_i \cdot x_i}}$$



# Logistic Regression: What happens when we call fit()?

```
from sklearn.linear_model import LogisticRegression  
  
classifier = LogisticRegression()  
classifier.fit(X_train, train_labels)
```

$$\frac{1}{1 + e^{-\sum_{i=1}^N \theta_i x_i}}$$


```
print(classifier.coef_[0, vocabulary['happy']])  
print(classifier.coef_[0, vocabulary['hate']])  
  
[-0.77738419 -1.37907454  2.15645872]  
[ 1.93643371 -0.18230344 -1.75413027]
```

# Logistic Regression: Predict()

```
y_test_pred = classifier.predict(X_test)
```

# Logistic Regression: Predict()

```
y_test_pred = classifier.predict(X_test)
```

$$P(y|\mathbf{x}) = \frac{1}{1 + e^{-\sum_{i=1}^N \theta_i \cdot x_i}}$$



# Logistic Regression Weights

$$P(y|\mathbf{x}) = \frac{1}{1 + e^{-\sum_{i=1}^N \theta_i \cdot x_i}}$$

Weights for class 0:

fuck with weight 2.2746936670362468  
ruined with weight 2.285901145651962  
bullshit with weight 2.294779431095451  
worse with weight 2.306632219294751  
horrible with weight 2.365448035218213  
disappointed with weight 2.461803150472878  
terrible with weight 2.549632390159403  
stupid with weight 2.6889778179393637  
worst with weight 3.0342095726396345  
sucks with weight 3.1526321346127872

Weights for class 1:

compare with weight 1.0880637334189303  
load with weight 1.0923449104160423  
capital with weight 1.0997919926130835  
clemson with weight 1.1399402906077198  
gucci with weight 1.1711297920743988  
saturday\u002c with weight 1.2514631288868407  
bama with weight 1.2597113294061348  
yakub with weight 1.27282481391156  
paterno with weight 1.278296534704651  
rush with weight 1.3409116343549219

Weights for class 2:

loved with weight 2.1554406394933  
happy with weight 2.1564587219609868  
proud with weight 2.2422940424189997  
congratulations with weight 2.2569560577857093  
fantastic with weight 2.290266356674143  
brilliant with weight 2.4569745211237803  
awesome with weight 2.553376158873567  
exciting with weight 2.6281831636826163  
congrats with weight 2.774124946022347  
amazing with weight 3.0595739864995113

# Quiz

# Quiz

---

## QUESTION 1

Consider a binary classification task for an imaginary language with only two words. The words have the following likelihoods:  $p(x_i = \text{'ikkk'} \mid y=0) = 0.2$ ,  $p(x_i = \text{'ikkk'} \mid y=1) = 0.8$ ,  $p(x_i = \text{'ooo'} \mid y=0) = 0.8$ ,  $p(x_i = \text{'ooo'} \mid y=1) = 0.2$ . The class priors are  $p(y=0) = 0.8$  and  $p(y=1) = 0.2$ . If we observe a document with just one word, 'ikkk', what class is the document most likely belong to?

- ☐  $y=1$
  - ☐  $y=0$
  - ☐ Equal probability.
-

# Quiz

## QUESTION 1

Consider a binary classification task for an imaginary language with only two words. The words have the following likelihoods:  $p(x_i = \text{'ikkk'} \mid y=0) = 0.2$ ,  $p(x_i = \text{'ikkk'} \mid y=1) = 0.8$ ,  $p(x_i = \text{'ooo'} \mid y=0) = 0.8$ ,  $p(x_i = \text{'ooo'} \mid y=1) = 0.2$ . The class priors are  $p(y=0) = 0.8$  and  $p(y=1) = 0.2$ . If we observe a document with just one word, 'ikkk', what class is the document most likely belong to?

- ☐  $y=1$
- ☐  $y=0$
- ☒ Equal probability.

Bayes' rule

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}$$

$P(\mathbf{x})$  is constant

# Quiz

---

## QUESTION 2

What are some limitations of the bag of words representation?

- ☐ It is expensive to obtain from a document.
- ☐ It loses information from the original documents.
- ☐ It does not contain enough information for tasks like spam filtering.
- ☐ It cannot deal with negated words without some tweaks.
- ☐ The vocabulary of test documents may not match that of the training set.
- ☐ The large size of the vocabulary makes the classifier quite slow.

# Quiz

## QUESTION 2

What are some limitations of the bag of words representation?

- ☐ It is expensive to obtain from a document.
- ☐ It loses information from the original documents. Ignores word order
- ☐ It does not contain enough information for tasks like spam filtering.
- ☐ It cannot deal with negated words without some tweaks. “not”, “happy” → positive
- ☐ The vocabulary of test documents may not match that of the training set. Can't use words that are only in the test set
- ☐ The large size of the vocabulary makes the classifier quite slow.

# Quiz

## QUESTION 3

How can we deal with negated words, e.g., the phrase “not happy” in a NB classifier?

- ☐ Include both ‘not’ and ‘happy’ in the bag of words (don’t remove ‘not’ as a stopwords).
- ☐ Before running NB classification, use regular expressions to detect negated phrases and append “NOT\_” to the start of negated words.
- ☐ Before running NB classifier, use regular expressions and lexicons to replace negated words with their antonyms, e.g, ‘not happy’ is replaced by ‘sad’.
- ☐ It doesn’t matter because the text will contain other negative terms so the classifier will still detect the negative sentiment.

# Quiz

## QUESTION 3

How can we deal with negated words, e.g., the phrase “not happy” in a NB classifier?

- ☐ Include both ‘not’ and ‘happy’ in the bag of words (don’t remove ‘not’ as a stopwords).
- ☒ Before running NB classification, use regular expressions to detect negated phrases and append “NOT\_” to the start of negated words.
- ☐ Before running NB classifier, use regular expressions and lexicons to replace negated words with their antonyms, e.g, ‘not happy’ is replaced by ‘sad’.
- ☐ It doesn’t matter because the text will contain other negative terms so the classifier will still detect the negative sentiment.

Put ‘NOT\_happy’ in vocabulary



# Quiz

---

## QUESTION 4

When is accuracy NOT sufficient to evaluate classifier performance?

- ☐ When comparing performance on different datasets with different numbers of examples in each class.
- ☐ When the cost of false positives and false negatives is not the same.
- ☐ When comparing classifiers on a balanced dataset.
- ☐ If you don't know the number of false positives.

# Quiz

	Gold label = +ve	Gold label = -ve
Prediction = +ve	<b>True positive (TP)</b>	<b>False positive (FP)</b>
Prediction = -ve	<b>False negative (FN)</b>	<b>True negative (TN)</b>

## QUESTION 4

When is accuracy NOT sufficient to evaluate classifier performance?

- ☐ When comparing performance on different datasets with different numbers of examples in each class.
- ☐ When the cost of false positives and false negatives is not the same.
- ☐ When comparing classifiers on a balanced dataset.
- ☐ If you don't know the number of false positives.

Avoid accuracy if the classes are very imbalanced or you cannot compute all of the confusion matrix.

# Quiz

---

## QUESTION 5

What advantages does logistic regression have over naïve Bayes?

- ☐ It handles features with continuous values.
- ☐ It can model complex relationships between features (e.g., a XOR function where if one feature is present and not the other, it will predict a certain class).
- ☐ It can also be used for regression
- ☐ It does not make the conditional independence assumption.

# Quiz

Does not assume independence, but still uses a linear function, so cannot model XOR.

---

## QUESTION 5

What advantages does logistic regression have over naïve Bayes?

- ☐ It handles features with continuous values.
- ☐ It can model complex relationships between features (e.g., a XOR function where if one feature is present and not the other, it will predict a certain class).
- ☐ It can also be used for regression
- ☐ It does not make the conditional independence assumption.

# Coming up....

# Coming Up....

- Next week: Reading week
- No lab or live lecture, but there will be some new videos to watch ready for the following week.
- Quiz:
  - Thursday 10<sup>th</sup> March (the week after reading week)
  - Covers weeks 4-6 of this unit – the Text Analytics video content so far
  - Similar style to the unmarked Blackboard quizzes
  - 10% of the unit