Probability Recap

Formal Definition

- **Sample Space** Ω — the set of all possible elementary events.
- **σ-algebra** Ω — a set of subsets of Ω containing and closed under complementation and countable unions. We refer to the elements of $\mathcal{A}$ as events (not elementary) and denote them with uppercase letters A, B, C.
- **Probability Distribution** $\Pr : \mathcal{A} \to [0,1]$ — a function such that:
  - $\Pr[\Omega] = 1$
  - For any two disjoint events $A, B \in \mathcal{A}$, where $A \cap B = \emptyset$, it holds: $\Pr[A \cup B] = \Pr[A] + \Pr[B]$
- **Probability Space** is referred to as the triplet of sample space, σ-algebra, and probability distribution $(\Omega, \mathcal{A}, \Pr)$.
- If Ω is finite or countable and the singletons of Ω are events and therefore $\mathcal{A} = 2^\Omega$, we will call the distribution discrete.

Example

- **Fair die:**
  - Ω = {□1, □2, □3, □4, □5, □6}
  - $\mathcal{A} = 2^\Omega$
  - $\Pr[A] = |A| / 6$
- If we define event A as the number rolled being less than 5, then A has a probability of $\Pr[A] = 2/3$.

Conditional Probabilities and Independence

- **Conditional Probability** of event A given event B is defined as: $\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]}$ when $Pr[B] \neq 0$.
- Events A and B are called **independent** if: $\Pr[A \cap B] = \Pr[A] \Pr[B]$
- If events A and B are independent, then: $\Pr[A \mid B] = \Pr[A]$

Basic Properties

- $Pr[A \cup B] = Pr[A] + Pr[B] - Pr[A \cap B]$
  (Addition Rule)

- $Pr\left[\bigcup_{i=1}^{n} A_i\right] \le \sum_{i=1}^{n} Pr\left[A_i\right]$
  (Union Bound)

- $Pr\left[A \mid B\right] = \dfrac{Pr\left[B \mid A\right] Pr\left[A\right]}{Pr\left[B\right]}$
  (Bayes' Formula)

- $Pr\left[\bigcap_{i=1}^{n} A_i\right] = Pr\left[A_1\right] Pr\left[A_2 \mid A_1\right] Pr\left[A_3 \mid A_1 \cap A_2\right] \ldots Pr\left[A_n \mid \bigcap_{i=1}^{n-1} A_i\right]$
  (Chain Rule)

- $\Omega = A_1 \cup A_2 \cup \ldots \cup A_n$, where $A_i \cap A_j = \varnothing$ for $i \neq j \Longrightarrow$
  $Pr\left[B\right] = \sum_{i=1}^{n} Pr\left[B \mid A_i\right] Pr\left[A_i\right]$
  (Total Probability Theorem)

Random Variables

- A **random variable** (random function) is a function $X: \Omega \to R$ such that for every real interval $I \subset R$, the preimage is an event, i.e., $X^{-1}(I) \in F$ (always true for discrete probability spaces).

- **Consequence:** If (Ω, F, Pr) is a discrete probability space, $X: \Omega \to R$ is a random variable, and $f: R \to R$ is a function, then $f(X): \Omega \to R$ is a random variable.

- The **distribution function** of a discrete random variable is the function:
  $x \mapsto Pr\left[X = x\right] = Pr\left[X^{-1}(x)\right]$.

- The **joint distribution function** of discrete random variables X, Y is the function:
  $(x, y) \mapsto Pr\left[X = x, Y = y\right] = Pr\left[X^{-1}(x) \cap Y^{-1}(y)\right]$.

- Random variables $X, Y: \Omega \to R$ are **independent** if:
  $Pr\left[X = x, Y = y\right] = Pr\left[X = x\right] Pr\left[Y = y\right]$ for every $x, y \in R$.

- A sequence of random variables is called **independent and identically distributed** if they are mutually independent and have the same distribution function.

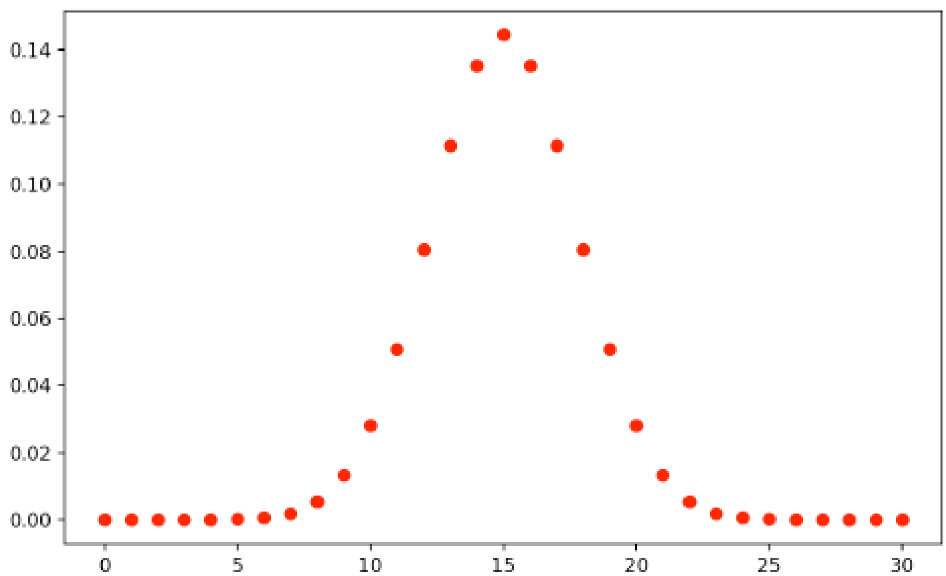
- A **multidimensional random variable** is a function $X: \Omega \to R^n$, for which its projections $X_i: \Omega \to R$, $X_i(\omega) := \mathrm{Proj}_i X(\omega)$, $i = 1, 2, \ldots, n$ are random variables.

- The **distribution function** of a discrete multidimensional random variable is the function:
  $(x_1, x_2, \ldots, x_n) \mapsto Pr\left[X = (x_1, x_2, \ldots, x_n)\right] = Pr\left[X^{-1}\left((x_1, x_2, \ldots, x_n)\right)\right] = Pr\left[X_1^{-1}(x_1) \cap X_2^{-1}(x_2) \cap \ldots \cap X_n^{-1}(x_n)\right] = P$
  .

- **Property:** Let $X:\Omega \rightarrow R^n$ be a random variable. Then $(X(\Omega), X(F), Pr)$ is a probability space.

Example of a Discrete Random Variable Distribution

- **Probability space:** All possible outcomes when flipping n coins.
- **Random variable X:** Number of heads obtained.
- **Binomial distribution:** $B(n,p)$
$Pr[X=k]=\binom{n}{k} p^k (1-p)^{n-k}$
- **Special case:** For $n=1$, we get the **Bernoulli distribution**.



Principle of Maximum Likelihood

- Given a sequence of m independent and identically distributed random variables $X_1, X_2, \ldots, X_m$ with a distribution function $Pr[X=x \mid \theta]$, which depends on the parameter θ.
- We have observed (measured) the corresponding values $x_1, x_2, \ldots, x_m$ for the sequence of random variables $X_1, X_2, \ldots, X_m$.

- The likelihood of having made the corresponding observation is:
  $L(\theta)=Pr\left[X_1=x_1, X_2=x_2, \ldots, X_m=x_m \mid \theta\right)=\prod_{i=1}^{m} Pr\left[X_i=x_i \mid \theta\right)$
- We obtain the maximum likelihood by finding the value of the parameter θ for which the likelihood $L(\theta)$ is maximal. That is,
  $\hat{\theta}=arg\,max_\theta L(\theta)$

Maximizing Likelihood for Binomial Distribution

- Assume a binomial distribution function $B(n,p)$ for m independent and identically distributed random variables $X_1, X_2, \ldots, X_m$ with observations $x_1, x_2, \ldots, x_m$. That is, $Pr[X=x \mid p)=\binom{n}{x} p^x (1-p)^{n-x}$.
- Let f_k be the frequency of value k among $x_1, x_2, \ldots, x_m$. We seek:

$$\hat{p}=arg\,\max_p L(p)=arg\,\max_p \log L(p)=arg\,\max_p \sum_{k=1}^{n} f_k \left(\log \binom{n}{k} + k \log p + (n-k) \log(1-p)\right)$$

- The derivative:

$$\frac{\partial \log L(p)}{\partial p}=\sum_{k=1}^{n}\left(\frac{k f_k}{p}-\frac{(n-k) f_k}{1-p}\right)=0$$

- Solving gives:

$$(1-p)\sum_{k=1}^{n} k f_k = p \sum_{k=1}^{n} (n-k) f_k$$

- Therefore:

$$\hat{p}=\frac{1}{nm}\sum_{i=1}^{m} x_i$$

To derive $\hat{p}=\frac{1}{nm}\sum_{i=1}^{m} x_i$ from the equation

$$(1-p)\sum_{k=1}^{n} k f_k = p \sum_{k=1}^{n} (n-k) f_k,$$

follow these steps:

1. **Expand the Right Side:**

   The right side can be expanded as:

$$p\sum_{k=1}^{n}(n-k)f_k=p\left(n\sum_{k=1}^{n}f_k-\sum_{k=1}^{n}kf_k\right).$$

2. **Simplify the Equation:**

   Substitute the expanded form into the equation:

$$(1-p)\sum_{k=1}^{n}kf_k=p\left(n\sum_{k=1}^{n}f_k-\sum_{k=1}^{n}kf_k\right).$$

3. **Combine Terms:**

   Rearrange the terms to isolate $\sum_{k=1}^{n}kf_k$:

$$\sum_{k=1}^{n}kf_k=\frac{pn\sum_{k=1}^{n}f_k}{1-p+p}=pn\sum_{k=1}^{n}f_k.$$

4. **Relate to Observations:**

   Recognize that $\sum_{k=1}^{n}kf_k$ is the total number of successes across all trials, which is equivalent to $\sum_{i=1}^{m}x_i$.

5. **Solve for $\hat{p}$:**

   Substitute back to find $\hat{p}$:

$$\hat{p}=\frac{\sum_{i=1}^{m}x_i}{n\sum_{k=1}^{n}f_k}.$$

6. **Normalize by Total Trials:**

   Since $\sum_{k=1}^{n}f_k=m$, the total number of trials, we have:

$$\hat{p}=\frac{1}{nm}\sum_{i=1}^{m}x_i.$$

This shows how the maximum likelihood estimate $\hat{p}$ is derived as the average number of successes per trial.

$$(1-p)\sum_{k=1}^{n} k f_k = p\left(n\sum_{k=1}^{n} f_k - \sum_{k=1}^{n} k f_k\right).$$

1. **Distribute p on the Right Side:**

$$p\left(n\sum_{k=1}^{n} f_k - \sum_{k=1}^{n} k f_k\right) = p n \sum_{k=1}^{n} f_k - p\sum_{k=1}^{n} k f_k.$$

2. **Combine Like Terms:**

   Move $p\sum_{k=1}^{n} k f_k$ to the left side:

$$(1-p)\sum_{k=1}^{n} k f_k + p\sum_{k=1}^{n} k f_k = p n \sum_{k=1}^{n} f_k.$$

3. **Factor Out $\sum_{k=1}^{n} k f_k$:**

$$\sum_{k=1}^{n} k f_k = p n \sum_{k=1}^{n} f_k.$$

4. **Solve for $\sum_{k=1}^{n} k f_k$:**

   Since $\sum_{k=1}^{n} f_k = m$, the total number of trials, we have:

$$\sum_{k=1}^{n} k f_k = \sum_{i=1}^{m} x_i,$$

   where $\sum_{i=1}^{m} x_i$ is the total number of successes.

5. **Substitute Back to Find $\hat{p}$:**

$$\hat{p} = \frac{\sum_{i=1}^{m} x_i}{n m}.$$

Bernoulli Document Model

- Let a vocabulary $V = \{t_1, t_2, \ldots, t_M\}$ be given.

- For each document, we associate an M-dimensional vector of zeros and ones $d=(e_1, e_2, \ldots, e_M)$, where $e_i=1$ if the term t_i appears in the document and $e_i=0$ otherwise. That is, $X=\{0,1\}^M$.
- We assume $U_i: X \to \{0,1\}$ are mutually independent random variables with Bernoulli distribution, such that U_i gives us the i-th projection of the elements of X.
- In this case:

$$\Pr(d)=\Pr[(e_1, e_2, \ldots, e_M)]=\Pr[U_1=e_1, U_2=e_2, \ldots, U_M=e_M]=\prod_{i=1}^{M} \Pr[U_i=e_i]$$

- We seek the most probable class c given that we have a document d. That is, we seek $c_{MAP}=arg\,max_{c \in C} \Pr[c \mid d]$
- MAP = maximum a posteriori
- $\Pr[c \mid d]=\frac{\Pr[d \mid c]\Pr[c]}{\Pr[d]}$
- $\Pr[d \mid c]=\Pr[(e_1, e_2, \ldots, e_M) \mid c]=\prod_{i=1}^{M} \Pr[U_i=e_i \mid c]$
- $c_{MAP}=arg\,max_{c \in C}\left(\Pr[c]\prod_{i=1}^{M} \Pr[U_i=e_i \mid c]\right)$
- $c_{MAP}=arg\,max_{c \in C}\left(\log \Pr[c]+\sum_{i=1}^{M} \log \Pr[U_i=e_i \mid c]\right)$

Estimating Parameters Using the Maximum Likelihood Principle

- N - number of documents in D
- N_c - number of documents in D of class c
- $N_{c,t}$ - number of documents in D of class c in which the term t appears
- $\Pr[c] \approx \frac{N_c}{N}$
- $\Pr[U_i=1 \mid c] \approx \frac{N_{c,t_i}}{N_c} \approx \frac{N_{c,t_i}+1}{N_c+2}$
- $\Pr[U_i=0 \mid c]=1-\Pr[U_i=1 \mid c]$

Algorithms for Naive Bayes Classifier Using Bernoulli Document Model

```
TrainBernoulliNB(C, D)
    V <- EXTRACT_VOCABULARY(D)
    N <- COUNT_DOCS(D)
    for each c in C do
        Nc <- COUNT_DOCS_IN_CLASS(D, c)
        prior[c] <- Nc/N
        for each t in V do
            Nct <- COUNT_DOCS_IN_CLASS_CONTAINING_TERM(D, c, t)
            condprob[t][c] <- (Nct + 1)/(Nc + 2)
    return V, prior, condprob

ApplyBernoulliNB(C, V, prior, condprob, d)
    Vd <- EXTRACT_TERMS_FROM_DOC(V, d)
    for each c in C do
        score[c] <- log prior[c]
        for each t in V do
            if t in Vd then
                score[c] += log(condprob[t][c])
            else
                score[c] += log(1-condprob[t][c])
    return argmax(c in C, score[c])
```

Example of Distribution of a Discrete Random Variable

- Sample space: all possible outcomes when tossing n coins.
- Random variable X: number of heads.
- **Binomial distribution**: $B(n, p)$

$$\Pr[X=k] = \binom{n}{k} p^k (1-p)^{n-k}$$

- **Generalization**: Multinomial distribution $M(n, l, p_1, p_2, \ldots, p_l)$
  - Toss n dice with l sides.
  - Random variables: $X_1, X_2, \ldots, X_l$ - number of times side “i” appears.

$$\Pr[X_1=k_1, X_2=k_2, \ldots, X_l=k_l] = \frac{n!}{k_1!k_2!\ldots k_l!} p_1^{k_1} p_2^{k_2} \ldots p_l^{k_l}$$

when

$$\sum_{i=1}^{l} k_i = n, \sum_{i=1}^{l} p_i = 1.$$

Maximizing Likelihood in Multinomial Distribution

- Assume a binomial function for the distribution $M(n, l, p_1, p_2, \ldots, p_l)$ of m i.i.d. joint random variables $X^{(1)}, X^{(2)}, \ldots, X^{(m)}$, where $X^{(i)} = \left(X_1^{(i)}, X_2^{(i)}, \ldots, X_l^{(i)}\right)$ with observations $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$, where $x^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_l^{(i)}\right)$. That is,

$$\Pr\left[X_1^{(i)} = x_1^{(i)}, X_2^{(i)} = x_2^{(i)}, \ldots, X_l^{(i)} = x_l^{(i)}\right) = \frac{n!}{x_1^{(i)}!\, x_2^{(i)}! \ldots x_l^{(i)}!} p_1^{x_1^{(i)}} p_2^{x_2^{(i)}} \ldots p_l^{x_l^{(i)}}$$

- We seek:

$$\hat{p} = \underset{p}{arg\,max}\, L(p) = \underset{p}{arg\,max} \log L(p) = \text{¿}$$

$$\text{¿}\, \underset{p}{arg\,max} \sum_{i=1}^{m} \log \Pr\left[X_1^{(i)} = x_1^{(i)}, X_2^{(i)} = x_2^{(i)}, \ldots, X_l^{(i)} = x_l^{(i)}\right) = \text{¿}$$

$$\text{¿}\, \underset{p}{arg\,max} \sum_{i=1}^{m} \left(\log \frac{n!}{x_1^{(i)}!\, x_2^{(i)}! \ldots x_l^{(i)}!} + x_1^{(i)} \log p_1 + x_2^{(i)} \log p_2 + \ldots + x_l^{(i)} \log p_l\right)$$

- **Problem**: If $p_i \rightarrow \infty$, then $L(p) \rightarrow \infty$.

- **Solution**: We need to find $\hat{p} = arg\,max_p \log L(p)$ with the constraint $\sum_{i=1}^{l} p_i = 1$.

$$\hat{p}_i = \frac{\sum_{j=1}^{m} x_i^{(j)}}{n\,m}$$

Multinomial Document Model

- Let a vocabulary $V = \{t_1, t_2, \ldots, t_M\}$ be given.

- For each document, we assign an M-dimensional vector of natural numbers: $d = (f_1, f_2, \ldots, f_M)$, where f_i represents the count of occurrences of term t_i in the document.
  That is, $X = N^M$.

- This representation of documents is called **Bag of Words**.
- Assuming that the document's length is $n = f_1 + \ldots + f_M$,
  we have a multinomial distribution. Specifically:

$$\Pr(d) = \Pr\big((f_1, f_2, \ldots, f_M)\big) = K_d \prod_{i=1}^{M} \Pr(t_i)^{f_i},$$

  where $K_d = \dfrac{n!}{f_1! f_2! \ldots f_M!}$.

- For an arbitrary (variable) document length, the coefficient K_d is multiplied by the probability $\Pr(|d| = n)$ — the probability of the given document d having a length n.
- We are looking for the most likely class c given that we have document d. That is, we are searching for:

$$c_{MAP} = argmax_{c \in C} \Pr(c \mid d)$$

- By Bayes' theorem:

$$\Pr(c \mid d) = \frac{\Pr(d \mid c) \cdot \Pr(c)}{\Pr(d)}$$

- The probability $\Pr(d \mid c)$ can be expanded as:

$$\Pr(d \mid c) = \Pr\big((f_1, f_2, \ldots, f_M) \mid c\big) = K_d \prod_{i=1}^{M} \Pr(t_i \mid c)^{f_i}$$

- Therefore, the maximum a posteriori class is:

$$c_{MAP} = argmax_{c \in C} \Pr(c) \prod_{i=1}^{M} \Pr(t_i \mid c)^{f_i}$$

- Taking the logarithm to simplify computation:

$$c_{MAP} = argmax_{c \in C} \log \Pr(c) + \sum_{i=1}^{M} f_i \log \Pr(t_i \mid c)$$

- Further simplified to:

$$c_{MAP} = argmax_{c \in C} \log \Pr(c) + \sum_{k=1}^{n} \log \Pr(t_{d_k} \mid c)$$

Estimation of Parameters Using the Principle of Maximum Likelihood

- N - the total number of documents in D.
- N_c - the number of documents in D that belong to class c.
- $T_{c,t}$ - the total occurrences of term t in the documents in D belonging to class c.

$$\Pr[c] \approx \frac{N_c}{N}$$

$$\Pr[t_i \mid c] \approx \frac{T_{c,t_i}}{\sum_{t' \in V} T_{c,t'}} \approx \frac{T_{c,t_i} + 1}{\sum_{t' \in V} T_{c,t'} + |V|}$$

Algorithms for Naive Bayes Classifier Using Multinomial Document Model

TrainMultinomialNB(C, D)

1. $V \leftarrow \text{EXTRACTVOCABULARY}(D)$
2. $N \leftarrow \text{COUNTDOCS}(D)$
3. **for each** c **in** C **do**
4. $\quad N_c \leftarrow \text{COUNTDOCSINCLASS}(D, c)$
5. $\quad \text{prior}[c] \leftarrow N_c / N$
6. $\quad \text{text}_c \leftarrow \text{CONCATENATETEXTOFALLDOCSINCLASS}(D, c)$
7. $\quad$ **for each** t **in** V **do**
8. $\quad\quad T\,c[t] \leftarrow \text{COUNTTOKENSOFTERM}(\text{text}_c, t)$
9. $\quad\quad$ **for each** t **in** V **do**
10. $\quad\quad\quad \text{condprob}[t][c] \leftarrow (\text{Tc}[t]+1) / (\sum (\text{Tc}[t']+1) \text{ for } t' \in V)$
11. **return** V, prior, condprob

ApplyMultinomialNB(C, V, prior, condprob, d)

1. $W \leftarrow \text{EXTRACTTOKENSFROMDOC}(V, d)$
2. **for each** c **in** C **do**
3. $\quad \text{score}[c] \leftarrow \log(\text{prior}[c])$
4. $\quad$ **for each** t **in** W **do**
5. $\quad\quad \text{score}[c] + ¿ \log(\text{condprob}[t][c])$
6. **return** $\text{argmax}(c \, \text{in} \, C, \text{score}[c])$

Gaussian Naïve Bayes

The probability density function for a feature $x=v$ given class C_k is defined as:

$$p(x=v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Where:

- μ_k is the mean of the feature values for class C_k.
- σ_k^2 is the variance of the feature values for class C_k.

Example of Naive Bayes

Naive Bayes

Tutorial - How to build a Spam Classifier in python and sklearn - milindsoorya.site

```
train_spam = ['send us your password', 'review our website', 'send your password', 'send us your account']
train_ham = ['Your activity report','benefits physical activity', 'the importance vows']
test_spam = ['renew your password', 'renew your vows']
test_ham = ['benefits of our account', 'the importance of physical activity']

import pandas as pd

data = pd.DataFrame({
    'text': train_spam + test_spam + train_ham + test_ham,
```

```
    'label': [1] * (len(train_spam) + len(test_spam)) + [0] *
(len(train_ham) + len(test_ham))
})
data
```

	text	label
0	send us your password	1
1	review our website	1
2	send your password	1
3	send us your account	1
4	renew your password	1
5	renew your vows	1
6	Your activity report	0
7	benefits physical activity	0
8	the importance vows	0
9	benefits of our account	0
10	the importance of physical activity	0

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
#remove the punctuations and stopwords
import string

def text_process(text):

    text = text.translate(str.maketrans('', '', string.punctuation))
    text = [word for word in text.split() if word.lower() not in
stopwords.words('english')]

    return " ".join(text)

text = data['text'].apply(text_process)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\MSI\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
text
```

```
0                 send us password
1                   review website
2                    send password
3                  send us account
4                   renew password
5                       renew vows
6                  activity report
7       benefits physical activity
8                  importance vows
9                 benefits account
```

```
10    importance physical activity
Name: text, dtype: object
```

```python
from collections import Counter

total_counts = Counter()
for i in range(len(text)):
    for word in text.values[i].split(' '):
        total_counts[word] += 1

print("Total words in data set: ", len(total_counts))
```

```
Total words in data set:  13
```

```python
vocab = sorted(total_counts, key=total_counts.get, reverse=True)
print(vocab)
```

```
['send', 'password', 'activity', 'us', 'account', 'renew', 'vows',
'benefits', 'physical', 'importance', 'review', 'website', 'report']
```

```python
vocab_size = len(vocab)
word2idx = {}
#print vocab_size
for i, word in enumerate(vocab):
    word2idx[word] = i

word2idx
```

```
{'send': 0,
 'password': 1,
 'activity': 2,
 'us': 3,
 'account': 4,
 'renew': 5,
 'vows': 6,
 'benefits': 7,
 'physical': 8,
 'importance': 9,
 'review': 10,
 'website': 11,
 'report': 12}
```

```python
def text_to_vector(text):
    word_vector = np.zeros(vocab_size)
    for word in text.split(" "):
        if word2idx.get(word) is None:
            continue
        else:
            word_vector[word2idx.get(word)] += 1
    return np.array(word_vector)
```

```
import numpy as np

word_vectors = np.zeros((len(text), len(vocab)), dtype=np.int_)
for i, t in enumerate(text):
    word_vectors[i] = text_to_vector(t)

word_vectors, word_vectors.shape
```

```
(array([[1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
        [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
        [0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0]]),
 (11, 13))
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(word_vectors,
data['label'], test_size=0.5, random_state=111)

from sklearn.naive_bayes import MultinomialNB

mnb = MultinomialNB(alpha=0.2)

pred_scores_word_vectors = []

mnb.fit(X_train, y_train)
mnb.predict(X_test)
```

```
array([1, 1, 0, 0, 1, 1])
```

```
y_test
```

```
5    1
0    1
7    0
8    0
2    1
1    1
Name: label, dtype: int64
```

```
X_test[-1]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0])
```

```
mnb.predict_proba(X_test)
```

```
array([[0.13545966, 0.86454034],
       [0.00343377, 0.99656623],
       [0.99661845, 0.00338155],
       [0.84941176, 0.15058824],
       [0.02544942, 0.97455058],
       [0.48456376, 0.51543624]])
```

TF-IDF - Term Frequency- Inverted Document Frequency

- View documents as **Bags Of Words**
- Mary lent John some money. = John lent Mary some money.
- Formula:

$$TF*IDF(word, document) = ¿$$

- n - total number of documents

Term Frequency

- **Frequency of word in a document (here, raw count)**
- **0 if the term is not met in the document!!!**
- Relevance does not increase proportionally with frequency -> **log (base of 10)**
- Makes TF-IDF **increase with the number of occurrences** within a doc

Document Frequency

- **Number of documents containing the word** - an inversed measure of significance
- Logarithm with base 10 dampens the effect of IDF
- Affects ranking of queries with **at least 2 terms**
- Makes TFIDF **increase with the rarity of the term in the collection**

```python
documents = [
    "Ross Edgley, at 33 - first man to swim around Britain",
    "Ross Edgley to Circumnavigate Britain Spent 5 Months at Sea",
    "Get Set 4 Swimming - H2OMG! Can this man swim around Britain?",
    "Welcome to the world of strongman swimming | British GQ",
]
query = "Who was the first man ever to swim around Britain?"

# ! pip3 install sklearn
from sklearn.feature_extraction.text import CountVectorizer

count_vectorizer = CountVectorizer()
count_vectorizer.fit(documents)
print(count_vectorizer.vocabulary_)  # word to id
```

```
{'ross': 15, 'edgley': 7, 'at': 2, '33': 0, 'first': 8, 'man': 12,
'to': 24, 'swim': 20, 'around': 1, 'britain': 3, 'circumnavigate': 6,
'spent': 18, 'months': 13, 'sea': 16, 'get': 9, 'set': 17, 'swimming':
21, 'h2omg': 11, 'can': 5, 'this': 23, 'welcome': 25, 'the': 22,
'world': 26, 'of': 14, 'strongman': 19, 'british': 4, 'gq': 10}
```

```
# transform produces a sparse representations of documents - only
values != 0
# we need toarray() to preview the whole lists
count_vectorizer.transform(documents).toarray()
```

```
array([[1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
0,
        0, 0, 1, 0, 0],
       [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,
0,
        0, 0, 1, 0, 0],
       [0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1,
1,
        0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
1,
        1, 0, 1, 1, 1]])
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(min_df=2)
tfidf_vectorizer.fit_transform(documents).toarray()
tfidf_vectorizer.vocabulary_
```

```
{'ross': 5,
 'edgley': 3,
 'at': 1,
 'man': 4,
 'to': 8,
 'swim': 6,
 'around': 0,
 'britain': 2,
 'swimming': 7}
```

Naive Base

The Data

Source: https://www.kaggle.com/crowdflower/twitter-airline-sentiment?select=Tweets.csv

This data originally came from Crowdflower's Data for Everyone library.

As the original source says,

A sentiment analysis job about the problems of each major U.S. airline. Twitter data was scraped from February of 2015 and contributors were asked to first classify positive, negative, and neutral tweets, followed by categorizing negative reasons (such as "late flight" or "rude service").

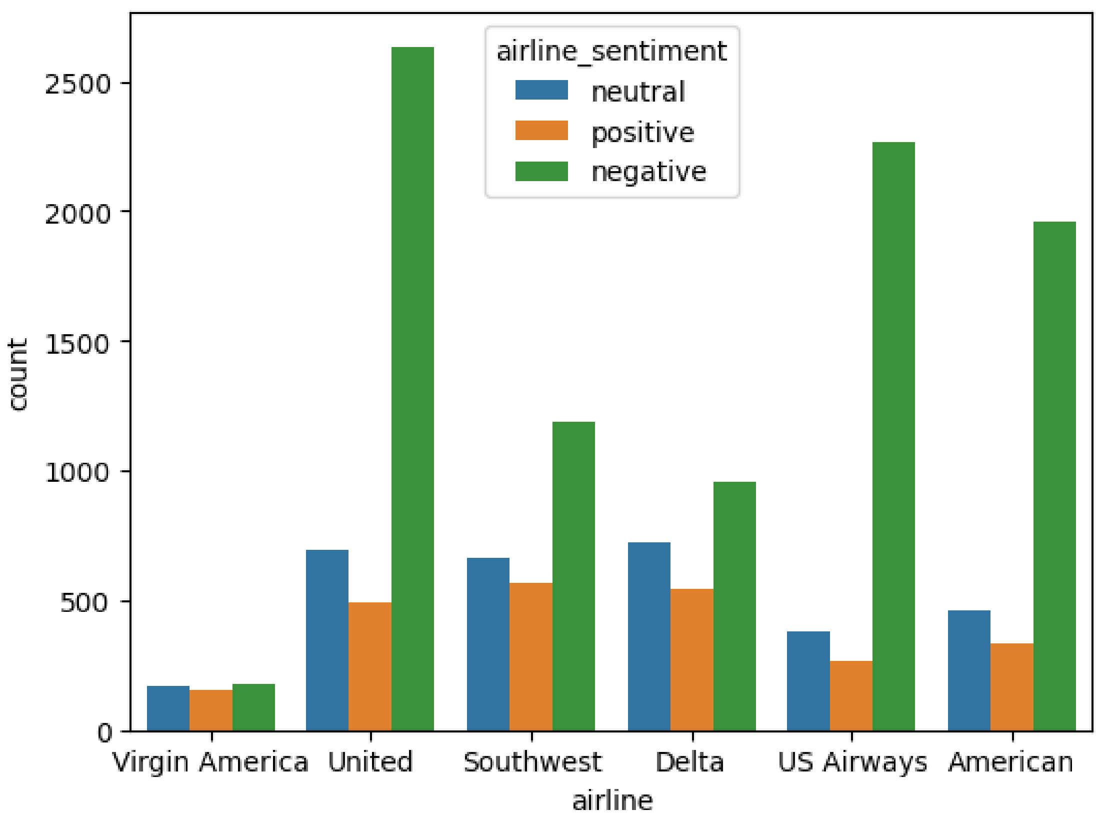
The Goal: Create a Machine Learning Algorithm that can predict if a tweet is positive, neutral, or negative. In the future we could use such an algorithm to automatically read and flag tweets for an airline for a customer service agent to reach out to contact.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("Data/airline_tweets.csv")

df.head()

             tweet_id airline_sentiment  airline_sentiment_confidence
\
0  570306133677760513           neutral                        1.0000

1  570301130888122368          positive                        0.3486

2  570301083672813571           neutral                        0.6837

3  570301031407624196          negative                        1.0000

4  570300817074462722          negative                        1.0000


  negativereason  negativereason_confidence         airline  \
0            NaN                        NaN  Virgin America
1            NaN                     0.0000  Virgin America
2            NaN                        NaN  Virgin America
3    Bad Flight                      0.7033  Virgin America
4    Can't Tell                      1.0000  Virgin America

  airline_sentiment_gold        name negativereason_gold
retweet_count  \
0                    NaN     cairdin                 NaN
0
1                    NaN    jnardino                 NaN
0
2                    NaN  yvonnalynn                 NaN
0
3                    NaN    jnardino                 NaN
0
4                    NaN    jnardino                 NaN
0

                                                 text tweet_coord  \
0                 @VirginAmerica What @dhepburn said.         NaN
1  @VirginAmerica plus you've added commercials t...         NaN
2  @VirginAmerica I didn't today... Must mean I n...         NaN
3  @VirginAmerica it's really aggressive to blast...         NaN
```

```
4  @VirginAmerica and it's a really big bad thing...         NaN

                   tweet_created tweet_location
user_timezone
0  2015-02-24 11:35:52 -0800            NaN  Eastern Time (US &
Canada)
1  2015-02-24 11:15:59 -0800            NaN  Pacific Time (US &
Canada)
2  2015-02-24 11:15:48 -0800      Lets Play  Central Time (US &
Canada)
3  2015-02-24 11:15:36 -0800            NaN  Pacific Time (US &
Canada)
4  2015-02-24 11:14:45 -0800            NaN  Pacific Time (US &
Canada)
```

```
sns.countplot(data=df, x="airline", hue="airline_sentiment")
```

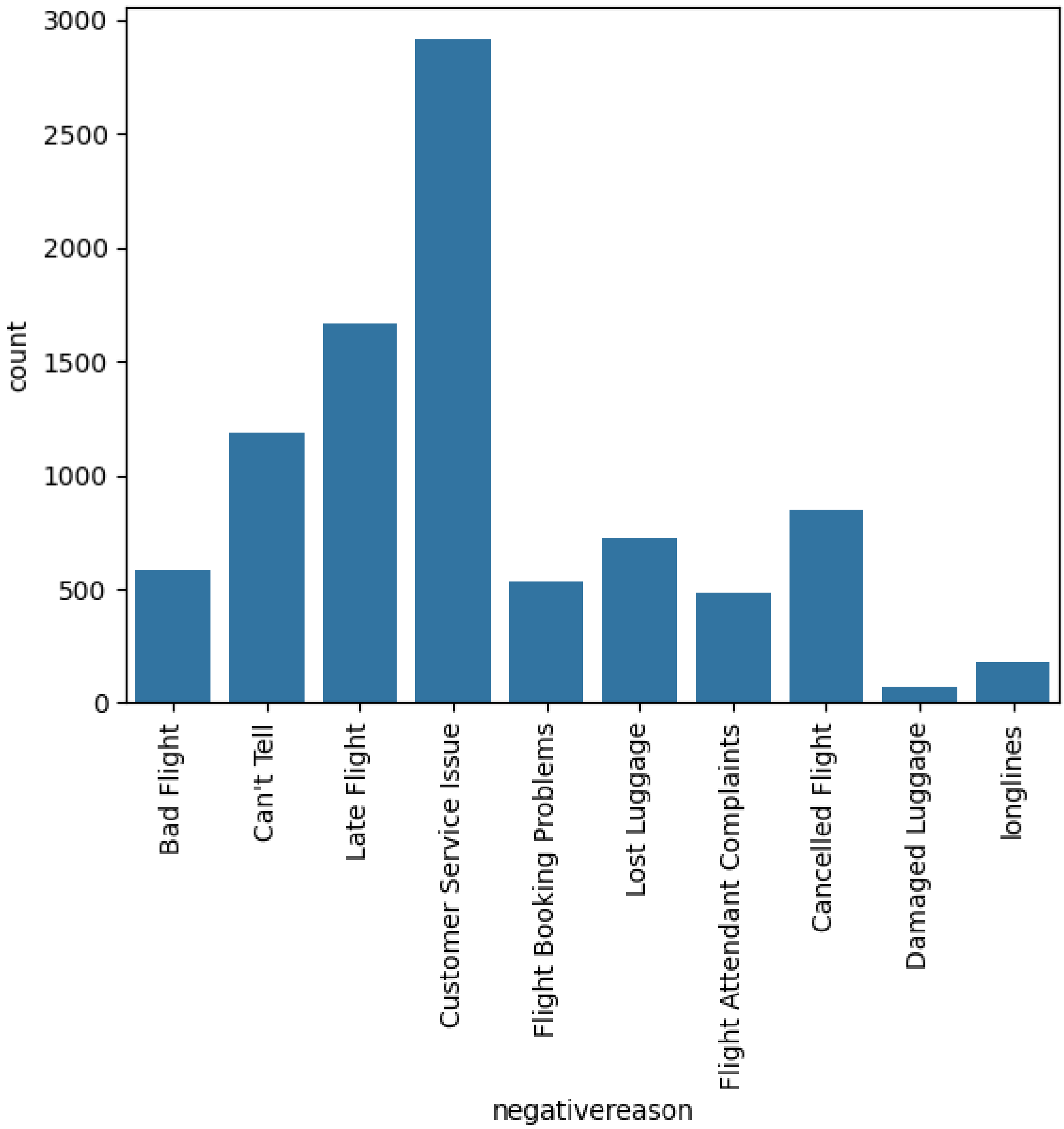
```
<Axes: xlabel='airline', ylabel='count'>
```




```
sns.countplot(data=df, x="negativereason")
plt.xticks(rotation=90)
```

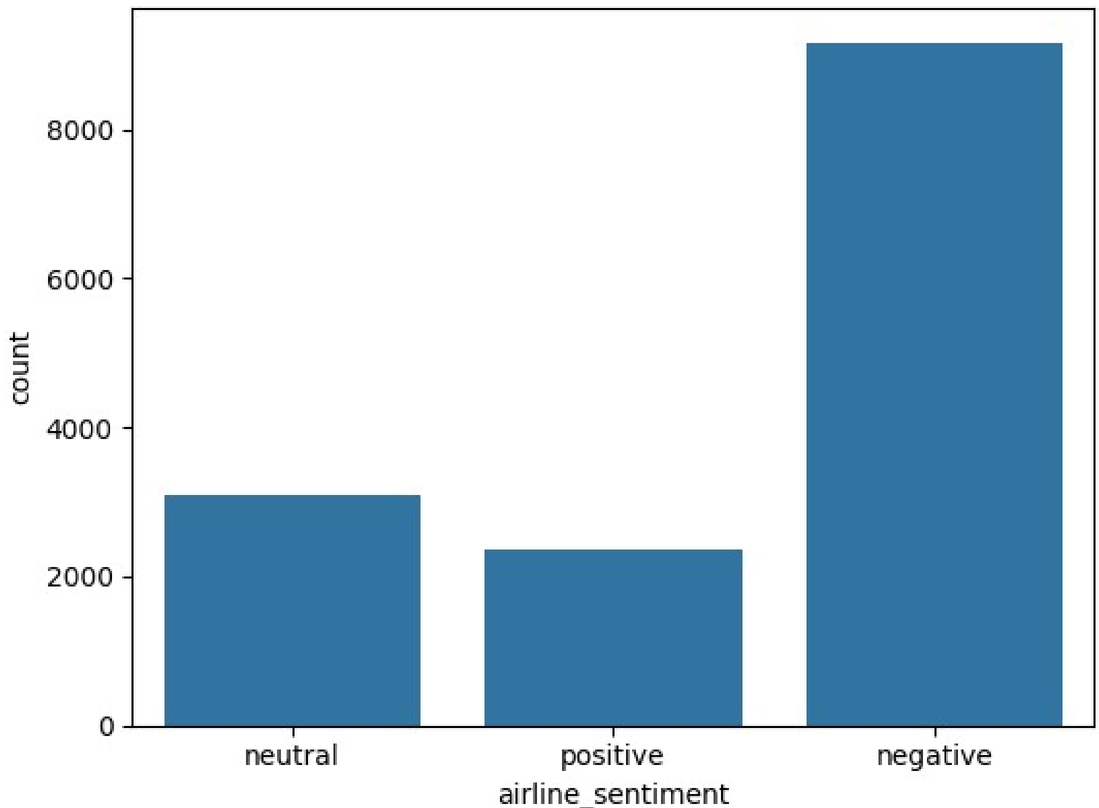
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [Text(0, 0, 'Bad Flight'),
```

```
 Text(1, 0, "Can't Tell"),
 Text(2, 0, 'Late Flight'),
 Text(3, 0, 'Customer Service Issue'),
 Text(4, 0, 'Flight Booking Problems'),
 Text(5, 0, 'Lost Luggage'),
 Text(6, 0, 'Flight Attendant Complaints'),
 Text(7, 0, 'Cancelled Flight'),
 Text(8, 0, 'Damaged Luggage'),
 Text(9, 0, 'longlines')])
```




```
sns.countplot(data=df, x="airline_sentiment")
```

```
<Axes: xlabel='airline_sentiment', ylabel='count'>
```




```
df["airline_sentiment"].value_counts()
```

```
airline_sentiment
negative    9178
neutral     3099
positive    2363
Name: count, dtype: int64
```

Features and Label

```
data = df[["airline_sentiment", "text"]]

data.head()
```

```
  airline_sentiment                                               text
0           neutral                @VirginAmerica What @dhepburn said.
1          positive  @VirginAmerica plus you've added commercials t...
2           neutral  @VirginAmerica I didn't today... Must mean I n...
3          negative  @VirginAmerica it's really aggressive to blast...
4          negative  @VirginAmerica and it's a really big bad thing...
```

```
y = df["airline_sentiment"]
X = df["text"]
```

Train Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=101
)
```

Vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer

countVec = CountVectorizer(stop_words="english")

countVec.fit(X_train)

CountVectorizer(stop_words='english')

X_train_countVec = countVec.transform(X_train)
X_test_countVec = countVec.transform(X_test)

X_train_countVec

<Compressed Sparse Row sparse matrix of dtype 'int64'
	with 107073 stored elements and shape (11712, 12971)>

tfidf = TfidfVectorizer(stop_words="english")

tfidf.fit(X_train)

TfidfVectorizer(stop_words='english')

X_train_tfidf = tfidf.transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

X_train_tfidf

<Compressed Sparse Row sparse matrix of dtype 'float64'
	with 107073 stored elements and shape (11712, 12971)>
```

DO NOT USE .todense() for such a large sparse matrix!!!

Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB

nb_cv = MultinomialNB()
nb_cv.fit(X_train_countVec, y_train)

MultinomialNB()
```

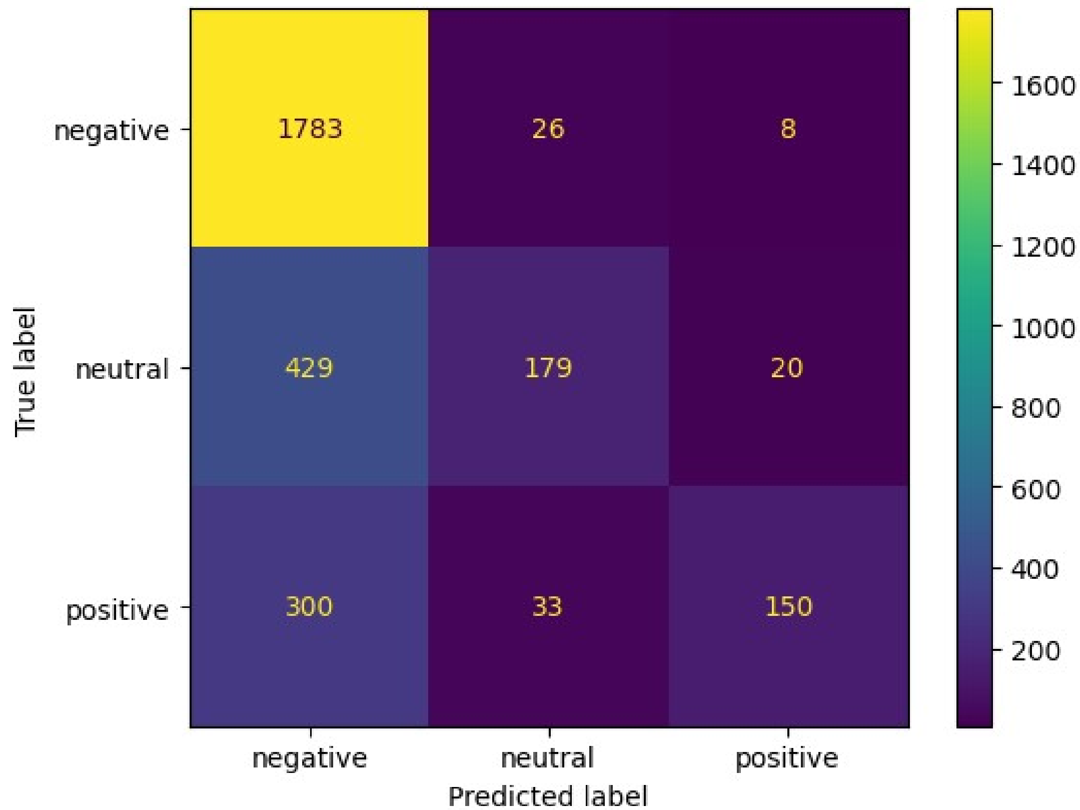
```python
nb_tfidf = MultinomialNB()
nb_tfidf.fit(X_train_tfidf, y_train)
```

```
MultinomialNB()
```

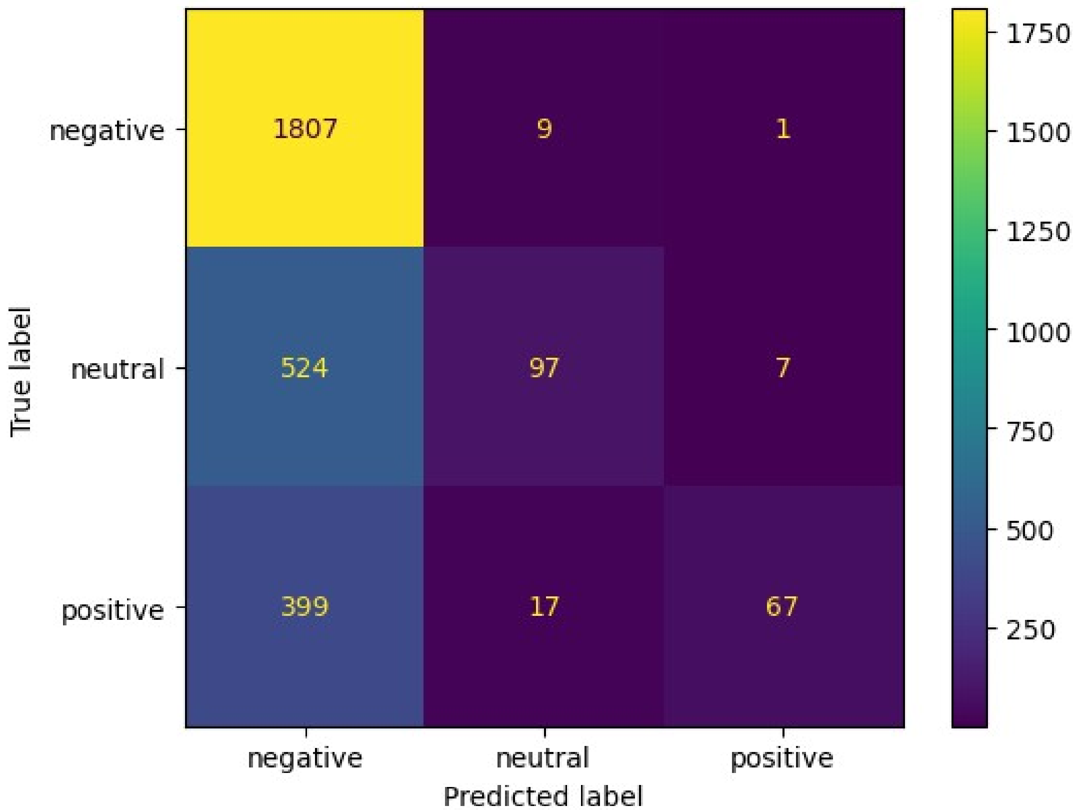
Performance Evaluation

```python
from sklearn.metrics import (
    ConfusionMatrixDisplay,
    classification_report,
    confusion_matrix,
)

def report(model):
    preds = model.predict(X_test_tfidf)
    print(classification_report(y_test, preds))
    cm = confusion_matrix(y_test, preds, labels=model.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=model.classes_)
    disp.plot()

print("Count vectorization model")
report(nb_cv)
```

```
Count vectorization model
              precision    recall  f1-score   support

    negative       0.71      0.98      0.82      1817
     neutral       0.75      0.29      0.41       628
    positive       0.84      0.31      0.45       483

    accuracy                           0.72      2928
   macro avg       0.77      0.53      0.56      2928
weighted avg       0.74      0.72      0.67      2928
```




```
print("Tf-idf model")
report(nb_tfidf)

Tf-idf model
              precision    recall  f1-score   support

    negative       0.66      0.99      0.79      1817
     neutral       0.79      0.15      0.26       628
    positive       0.89      0.14      0.24       483

    accuracy                           0.67      2928
   macro avg       0.78      0.43      0.43      2928
weighted avg       0.73      0.67      0.59      2928
```



Finalizing a PipeLine for Deployment on New Tweets

If we were satisfied with a model's performance, we should set up a pipeline that can take in a tweet directly.

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([("tfidf", TfidfVectorizer()), ("naiveBase",
MultinomialNB())])

pipe.fit(df["text"], df["airline_sentiment"])

Pipeline(steps=[('tfidf', TfidfVectorizer()), ('naiveBase',
MultinomialNB())])

new_tweet = ["I really liked the flight"]
pipe.predict(new_tweet)

array(['negative'], dtype='<U8')

new_tweet = ["bad flight"]
pipe.predict(new_tweet)

array(['negative'], dtype='<U8')

pipe.predict(X_test)
```

```
array(['positive', 'negative', 'negative', ..., 'negative',
'negative',
       'negative'], dtype='<U8')
```

```
X_test.iloc[0]
```

```
'@SouthwestAir thanks! Very excited to see it :D'
```

```
new_tweet = ["ok flight"]
pipe.predict(new_tweet)
```

```
array(['negative'], dtype='<U8')
```