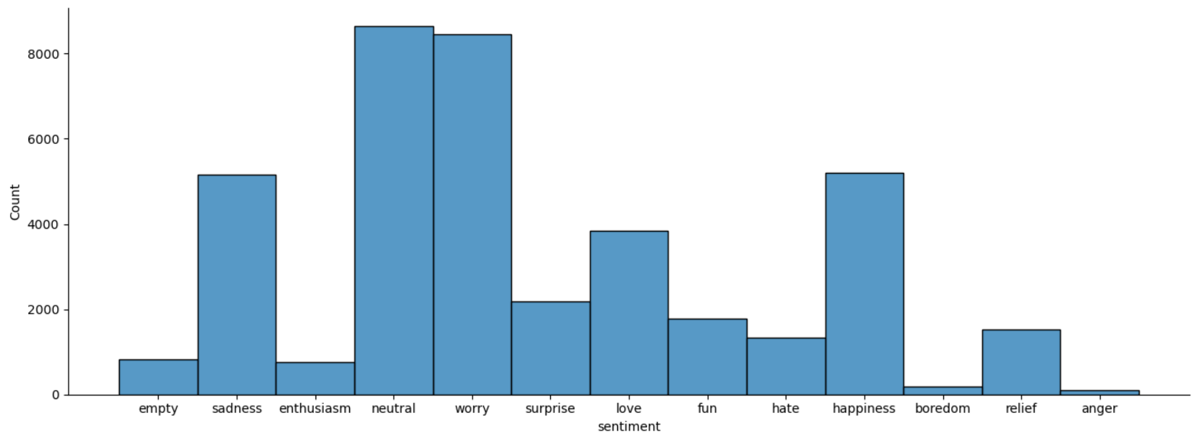
In [75]:
```
import pandas as pd

df = pd.read_csv('tweet_emotions.csv', index_col='tweet_id')
```

In [76]:
```
import seaborn as sb

sb.displot(df.sentiment, aspect=16/6)
```

Out[76]: <seaborn.axisgrid.FacetGrid at 0x7f91ff8732d0>



The dataset is a list of texts from emotions from tweets. The attempt here is to make a happy/sad sentiment classifier.

In [77]:
```
# filter "neutral"
df = df[df.sentiment.isin(['empty','sadness','enthusiasm','worry','surprise'
df.content.replace('[\d][\d]+', '', regex=True, inplace=True)
df.content.transform(lambda x: x.lower())

X = df.content
y = df.sentiment.isin(['enthusiasm','surprise','love','fun','happiness','rel

from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split as tts

X, X_t, y, y_t = tts(X,y,test_size=0.2,train_size=0.8,random_state=1)

vectorizer = TfidfVectorizer(stop_words=set(stopwords.words('english')), bin
X = vectorizer.fit_transform(X)
X_t = vectorizer.transform(X_t)
```

In [78]:
```
from sklearn.naive_bayes import BernoulliNB as NaBa
from math import log

nb_model = NaBa()
nb_model.fit(X, y)

# check if model learned anything

happy = sum(y == True)/len(y)
print(f'happiness: {happy}, log: {log(happy)}, model: {nb_model.class_log_pr
```

```
happiness: 0.488740085296345, log: -0.7159244537270022, model: -0.7159244537
27002
```

In [79]:
```
# Something, yes. Metrics:
from sklearn.metrics import accuracy_score as acc, precision_score as prec,

preds = nb_model.predict(X_t)
print(cm(y_t, preds))
print(f'acc: {acc(y_t, preds)}')
print(f'prec (sad, happy): ({prec(y_t, preds, pos_label=False)},{prec(y_t, p
print(f'rec (sad, happy): ({rec(y_t, preds, pos_label=False)}, {rec(y_t, pre
print(f'f1: {f1(y_t, preds)}')
# but not very well
```

```
[[2577  659]
 [1068 1969]]
acc: 0.724693129284234
prec (sad, happy): (0.7069958847736626,0.7492389649923896)
rec (sad, happy): (0.796353522867738, 0.6483371748435957)
f1: 0.6951456310679611
```

In [80]:
```
# now, logreg
from sklearn.linear_model import LogisticRegression as LR

lr_model = LR(multi_class='ovr', solver='saga', random_state=1)
lr_model.fit(X, y)

preds = lr_model.predict(X_t)
print(cm(y_t, preds))
print(f'acc: {acc(y_t, preds)}')
print(f'prec (sad, happy): ({prec(y_t, preds, pos_label=False)},{prec(y_t, p
print(f'rec (sad, happy): ({rec(y_t, preds, pos_label=False)}, {rec(y_t, pre
print(f'f1: {f1(y_t, preds)}')
```

```
[[2490  746]
 [ 917 2120]]
acc: 0.7348955842499602
prec (sad, happy): (0.7308482535955386,0.7397069085833915)
rec (sad, happy): (0.76946847960445, 0.6980572933816266)
f1: 0.7182788412671524
```

In [81]:

```
# now, nn

from sklearn.neural_network import MLPClassifier as MLP # heh
nn_model = MLP(
    solver = 'adam',
    alpha = 1e6,
    activation = 'logistic',
    hidden_layer_sizes = (50, 13),
    random_state = 1
)
nn_model.fit(X, y)
```

Out[81]:

MLPClassifier

```
MLPClassifier(activation='logistic', alpha=1000000.0,
              hidden_layer_sizes=(50, 13), random_state=1)
```

In [82]:

```
preds = lr_model.predict(X_t)
print(cm(y_t, preds))
print(f'acc: {acc(y_t, preds)}')
print(f'prec (sad, happy): ({prec(y_t, preds, pos_label=False)},{prec(y_t, p
print(f'rec (sad, happy): ({rec(y_t, preds, pos_label=False)}, {rec(y_t, pre
print(f'f1: {f1(y_t, preds)}')
```

```
[[2490  746]
 [ 917 2120]]
acc: 0.7348955842499602
prec (sad, happy): (0.7308482535955386,0.7397069085833915)
rec (sad, happy): (0.76946847960445, 0.6980572933816266)
f1: 0.7182788412671524
```

All three seem to have equal accuracy in determining the happiness level of a tweet. However, the neural network seems to take a lot more processing power to make in the first place. It's significantly more math. It also took up about 500mb of ram in order to fit.

In [ ]: