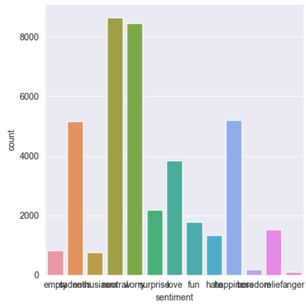
In [ ]:

In [26]:
```
import pandas
import seaborn
import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.linear_model.logistic import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn import metrics as metrics
import numpy as np
from nltk.corpus import stopwords
from sklearn.neural_network import MLPClassifier



# Open and read the files into pandas dataframes
testdf = pandas.read_csv("tweet_emotions.csv")
# print(testdf['sentiment'])

seaborn.catplot(x="sentiment" ,kind="count", data=testdf)

# Split the data
X = testdf.content
y = testdf.sentiment

# Train test/split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

# Use default vectorizer
vectorizer = TfidfVectorizer()

# vectorize
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

# Train the classifier
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

# Make predictions and print confusion matrix
pred = naive_bayes.predict(X_test)
print(confusion_matrix(y_test, pred))
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
[[   0    0    0    0    0    0    0    9    0    0    0   17]
 [   0    0    0    0    0    0    0    7    0    0    0   33]
 [   0    0    0    0    0    2    0    0   65    0    1    0  102]
 [   0    0    0    0    0    5    0    1   63    0    0    0   91]
 [   0    0    0    0    0   15    0    1  148    0    0    0  229]
 [   0    0    0    0    0  107    0   18  407    0    0    0  545]
 [   0    0    0    0    0    0    0    0   44    0    1    0  193]
 [   0    0    0    0    0   53    0  104  233    0    1    0  409]
 [   0    0    0    0    0   32    0    8  712    0    9    0  911]
 [   0    0    0    0    0    8    0    3  102    0    0    0  192]
 [   0    0    0    0    0    0    0    1  143    0    4    0  824]
 [   0    0    0    0    0    9    0    4  151    0    1    0  288]
 [   0    0    0    0    0   11    0    5  285    0    3    0 1390]]
accuracy score:  0.289625
```



Naive Bayes

In [22]:
```
# Train test/split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

# Use default vectorizer
vectorizer = TfidfVectorizer()

# vectorize
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

# Train the classifier
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

# Make predictions and print confusion matrix
pred = naive_bayes.predict(X_test)
print(confusion_matrix(y_test, pred))
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
[[   0    0    0    0    0    0    0    9    0    0    0   17]
 [   0    0    0    0    0    0    0    7    0    0    0   33]
 [   0    0    0    0    0    2    0    0   65    0    1    0  102]
 [   0    0    0    0    0    5    0    1   63    0    0    0   91]
 [   0    0    0    0    0   15    0    1  148    0    0    0  229]
 [   0    0    0    0    0  107    0   18  407    0    0    0  545]
 [   0    0    0    0    0    0    0    0   44    0    1    0  193]
 [   0    0    0    0    0   53    0  104  233    0    1    0  409]
 [   0    0    0    0    0   32    0    8  712    0    9    0  911]
 [   0    0    0    0    0    8    0    3  102    0    0    0  192]
 [   0    0    0    0    0    0    0    1  143    0    4    0  824]
 [   0    0    0    0    0    9    0    4  151    0    1    0  288]
 [   0    0    0    0    0   11    0    5  285    0    3    0 1390]]
accuracy score:  0.289625
```

As we can see, the performance of NB on this dataset has much to be desired, likely due to the larger amount of categories it has to work with. With each category getting less data, its predictions become less accurate.

Linear Regression

In [31]:
```
# Create and fit pipeline

pipe1 = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('logreg', LogisticRegression(multi_class='multinomial', solver='lbfgs',class_weight='balanced')),
])

pipe1.fit(testdf.content, testdf.sentiment)

# Test and evaluate
pred = pipe1.predict(testdf.content)

print("Confusion matrix:\n", metrics.confusion_matrix(testdf.sentiment, pred))

print("\nOverall accuracy: ", np.mean(pred==testdf.sentiment))
```

```
/Users/noahwhitworth/Desktop/HLTPython/TextClassify/venv/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Confusion matrix:
 [[ 110    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0  179    0    0    0    0    0    0    0    0    0    0    0]
 [   1   11  748   11    5    4    9    6    4   10    8    1    9]
 [   0    4    1  716    5    2    1    5    6    9    5    1    4]
 [   2    5   23   55 1379   56   22   54   30   48   31   43   28]
 [  14   31  120  262  426 2447   66  471  421  394  145  263  149]
 [   1   13   15   11   15    8 1182    9    4    9   29   13   14]
 [   5   16   70  118  198  330   66 2274  208  189  142  144   82]
 [  29  100  414  446  517  442  284  349 3884  587  492  552  542]
 [   4   13   23   32   41   42   19   61   25 1178   41   22   25]
 [  18   91  206  182  203  133  420  159  325  242 2483  269  434]
 [   5   11   32   67   68  108   56   95   68   78   98 1424   77]
 [  42  135  357  366  422  319  573  275  855  493 1110  609 2903]]

Overall accuracy:  0.522675
```

Linear Regression does notably better, however there is still much to be desired. This overall poor performance is likely due to a combination of multiple categories, biases in the data, and the nature of the task it is asked to do (categorize the sentiments of tweets).

Neural Networks

In [35]:
```
from nltk.corpus import stopwords
from sklearn.neural_network import MLPClassifier

# Text preprocessing

stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=stopwords, binary=True)

# Set up pipe, just like the Linear Regression

pipe1 = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('neuralnet', MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(15, 7), random_state=1)),
         ])

pipe1.fit(testdf.content, testdf.sentiment)

# Predict and test

pred = pipe1.predict(testdf.content)

print("Confusion matrix:\n", metrics.confusion_matrix(testdf.sentiment, pred))

print("\nOverall accuracy: ", np.mean(pred==testdf.sentiment))
```

```
/Users/noahwhitworth/Desktop/HLTPython/TextClassify/venv/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:541: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
Confusion matrix:
 [[   0    0    0    0    0    0    0    0   19    0    4    1   86]
 [   0    0    0    0    0    0    0    0   42    0    9    2  126]
 [   0    0    0    0    0    6    0    3  577    0   13    5  223]
 [   0    0    0    0    0    0   25    0   15  599    0   15   15   90]
 [   0    0    0    0    0 1206    0   83  401    0    4   16   66]
 [   0    0    0    0    0 3761    0  886  420    0   32   40   70]
 [   0    0    0    0    0    0    0    0   25    0  209    0 1089]
 [   0    0    0    0    0 1071    0 2478  108    0   64   54   67]
 [   0    0    0    0    0  181    0   59 7263    0   70  111  954]
 [   0    0    0    0    0  520    0   74  754    0   16   35  127]
 [   0    0    0    0    0    2    0    4  138    0 1744   10 3267]
 [   0    0    0    0    0   98    0   42 1589    0   58   73  327]
 [   0    0    0    0    0   12    0   31  506    0 1247   17 6646]]

Overall accuracy:  0.549125
```

Finally, we can see that the Neural Network did the best out of all the methods, doing slightly better than the Linear Regression and over twice as good as the Naive Bayes method. However, as said before, the results still leave much to be desired to the the biases in the data.