Answer 1

I first tried the .split() method of python. This gave an accuracy of 44% which is the same as majority prediction. The error was that the .split() method only separates at spaces, thereby considering “good” and “good!” as two separate words. This leads to the individual terms to be very low, thereby causing a majority prediction since the sum is not outweighing the difference introduced by p(y).

Then I used word_tokenize from nltk but that ran into underflow issues (the net size of the emails became very large as it counts multiple punctuations as different words)

Finally I used CountVectorizer from SKLearn.

Random prediction: 20%
Majority prediction: 44%
My Implementation: 60.035% (test set)
63.05% (training set)

There is a significant improvement over both the random prediction and majority prediction

Confusion matrix:

```
[[ 14335.   2785.   1360.   1080.   3136.]
 [  3845.   3290.   1675.    727.    323.]
 [  1187.   3335.   5190.   2543.    574.]
 [   449.   1081.   5410.  18027.  15353.]
 [   353.    347.    896.   6981.  39436.]]
```

Highest value for diagonal entry: 5
The classification is not very good as the non-diagonal entries are quite high. This is to be expected since the classes themselves have some dependence in reality.

47% accuracy using the function provided (test set) after both stemming and stopword removal

Feature engineering

Bigrams:63.28 % on test set

```
[[  1.66810000e+04   4.11900000e+03   1.76700000e+03   8.37000000e+02
    1.42700000e+03]
 [  9.12000000e+02   7.90000000e+02   2.14000000e+02   2.10000000e+01
    2.00000000e+00]
 [  7.57000000e+02   1.84100000e+03   1.69300000e+03   2.40000000e+02
    4.00000000e+01]
 [  1.30900000e+03   3.49800000e+03   9.53100000e+03   1.86250000e+04
    1.05240000e+04]
 [  5.10000000e+02   5.90000000e+02   1.32600000e+03   9.63500000e+03
    4.68290000e+04]]
```

POS+Bigrams=63.19%

```
[[  1.68110000e+04   4.22800000e+03   1.75000000e+03   7.79000000e+02
    1.23700000e+03]
 [  5.80000000e+02   5.05000000e+02   1.32000000e+02   9.00000000e+00
    1.00000000e+00]
 [  5.74000000e+02   1.35800000e+03   1.17100000e+03   1.58000000e+02
    2.90000000e+01]
 [  1.55800000e+03   3.99200000e+03   9.87600000e+03   1.77070000e+04
    9.24700000e+03]
 [  6.46000000e+02   7.55000000e+02   1.60200000e+03   1.07050000e+04
    4.83080000e+04]]
```

F1 score:

1:0.65
2:0.34
3:0.36
4:0.54
5:0.74

Macro avg:0.6
Micro avg:0.52
Weighted avg:0.61

Good metric to see the problem with majority prediction

Full training set:

accuracy:72.021%
F1 value:0.67