### 1.1.2 1b) Parameters of the vectorizer

""" Trial Run 1
Settings: Default --> binary=False, ngram_range=(1,1)
Result Accuracy: 0.795"""

```
print(clf.score(dev_test_vectors, dev_test_target)) #0.795
```

""" Trial Run 2
Settings: binary=True, ngram_range=(1,1)
Result Accuracy: 0.815"""

```
print(clf.score(dev_test_vectors, dev_test_target)) #0.815
```

""" Trial Run 3
Settings: binary=False, ngram_range=(1,2)
Result Accuracy: 0.795"""

```
print(clf.score(dev_test_vectors, dev_test_target)) #0.795
```

""" Trial Run 4
Settings: binary=True, ngram_range=(1,2)
Result Accuracy: 0.84"""

```
print(clf.score(dev_test_vectors, dev_test_target)) #0.84
```

""" Trial Run 5
# Settings: binary=False, ngram_range=(1,3)
# Result Accuracy: 0.81"""

```
print(clf.score(dev_test_vectors, dev_test_target)) #0.81
```

""" Trial Run 6
Settings: binary=True, ngram_range=(1,3)
Result Accuracy: 0.86"""

```
print(clf.score(dev_test_vectors, dev_test_target)) #0.86
```

| **Mean Accuracy** | | **CountVectorizer: Binary** | |
|---|---|---|---|
| | | **True** | **False** |
| **CountVectorizer: Ngram_range** | **[1,1]** | 0.815 | 0.795 |
| | **[1,2]** | 0.84 | 0.795 |
| | **[1,3]** | 0.86 | 0.81 |

The best result is obtained by using the settings binary=True and ngram_range=[1,3]. By using these parameters, the CountVectorizer

1) Takes into account unigrams, bigrams and trigrams to classify the texts and
2) No longer observes how often a token occurs in the texts but only whether it occurs or not. If the word occurs in any of the texts, it will get a '1', if not it gets a '0'.

The worst results are achieved when using the settings binary=False and ngram_range=[1,1] as well as ngram_range=[1,2]. In general, it is observable that the runs with binary=True achieve better results than with binary=False. Therefore, one could conclude that for this specific task, it is not of importance how often a word occurs in the text but whether it occurs at all. Furthermore, the CountVectorizer achieves higher when taking into account larger consecutive units such as trigrams or bigrams rather than only unigrams. With regard to this, one may conclude that it is easier to decide whether a review is positive or negative on the basis of larger consecutive units in comparison to only one or two words.

# 2. N-fold cross-validation

## 2a) 9-fold cross-validation

##### Results of the run #####

"""Settings: binary=True, ngram_range=(1,3)"""

```
print("True + (1,3) ", "\n", scorecard) #[0.855, 0.85, 0.855, 0.82, 0.83,
0.86, 0.895, 0.84, 0.88]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.85
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.023
```

## 2b) 9-fold cross-validation with different settings

##### Results of the runs #####

"""Settings: binary=True, ngram_range=(1,3)"""

```
print("True + (1,3) ", "\n", scorecard) #[0.855, 0.85, 0.855, 0.82, 0.83,
0.86, 0.895, 0.84, 0.88]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.85
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.023
```

"""Settings: binary=False, ngram_range=(1,3)"""

```
print("False + (1,3) ", "\n", scorecard) #[0.83, 0.835, 0.825, 0.815, 0.76, 0.81, 0.9, 0.825, 0.835]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.83
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.04
```

"""Settings: binary=False, ngram_range=(1,2)"""

```
print("False + (1,2) ", "\n", scorecard) #[0.82, 0.82, 0.825, 0.81, 0.795, 0.825, 0.895, 0.865, 0.84]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.83
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.03
```

"""Settings: binary=True, ngram_range=(1,2)"""

```
print("True + (1,2) ", "\n", scorecard) #[0.84, 0.845, 0.835, 0.835, 0.845, 0.87, 0.885, 0.86, 0.86]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.85
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.017
```

"""Settings: binary=False, ngram_range=(1,1)"""

```
print("False + (1,1) ", "\n", scorecard) #[0.795, 0.78, 0.835, 0.775, 0.83, 0.8, 0.86, 0.85, 0.82]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.82
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.03
```

"""Settings: binary=True, ngram_range=(1,1)"""

```
print("True + (1,1) ", "\n", scorecard) #[0.8, 0.805, 0.825, 0.805, 0.845, 0.835, 0.855, 0.84, 0.84]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.83
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.02
```

Mean accuracy for 9-fold cross-validation with different settings for the CountVectorizer

| **Mean Accuracy** | **CountVectorizer: Binary** |
|---|---|

| | | True | False |
|---|---|---|---|
| **CountVectorizer: Ngram_range** | **[1,1]** | 0.83 | 0.82 |
| | **[1,2]** | 0.85 (std: 0.017) | 0.83 |
| | **[1,3]** | 0.85 (std: 0.023) | 0.83 |

In this 9-fold cross-validation, the settings binary=True and ngram_range=(1,2) achieve the best results in comparison to the other five possible settings. However, the difference between these settings and the best settings from exercise 1 (binary=True and ngram_range=(1,3)) is only small and decided by the lower standard deviation for ngram_range=(1,2).

Here again, it is observable that the runs with binary=True achieve higher accuracies that those with binary=False. Furthermore, and also adding to the results from exercise 1, runs that take bigrams and trigrams into account achieve better than those only considering monograms.

## 3. Linear Regression

##### Results of the runs #####

"""Settings: Naïve Bayes, binary=False, ngram_range=(1,1)"""

```
print("False + (1,1) ", "\n", scorecard) #[0.795, 0.78, 0.835, 0.775, 0.83,
0.8, 0.86, 0.85, 0.82]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.82
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.03
```

"""Settings: binary=True, ngram_range=(1,2)"""

```
print("True + (1,2) ", "\n", scorecard) #[0.84, 0.845, 0.835, 0.835, 0.845,
0.87, 0.885, 0.86, 0.86]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.85
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.017
```

"Settings: Logistic Regression, binary=False, ngram_range=(1,1)"""

```
print("False + (1,1) + Logistic Regression", "\n", scorecard) #[0.845,
0.81, 0.85, 0.83, 0.845, 0.815, 0.91, 0.815, 0.85]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.84
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.03
```

"""Settings: Logistic Regression, binary=True, ngram_range=(1,2)"""

```
print("True + (1,2) + Logistic Regression", "\n", scorecard) #[0.875,
0.865, 0.89, 0.85, 0.885, 0.87, 0.89, 0.855, 0.885]
mean = sum(scorecard) / len(scorecard)
print("Mean: ", mean) #0.87
print("Standard Deviation: ", statistics.stdev(scorecard)) #0.01
```

| **Mean Accuracy** | | **Settings CountVectorizer** | |
|---|---|---|---|
| | | **Default (False, (1,1))** | **Best settings from ex.2 (True, (1,2))** |
| **Classifier** | **Naive Bayes** | 0.82 | 0.85 |
| | **Logistic Regression** | 0.84 | 0.87 |

What can be seen from the table is that independent of the classifier, the runs with the adjusted CountVectorizer settings binary=True and ngram_range=(1,2) were better than the ones with the default settings binary=False and ngram_range=(1,1). Therefore, one can conclude that the CountVectorizers taking into account bigrams (and also trigrams, see ex.2) achieve higher than the Vectorizers only working on the basis of monograms.

Moreover, it is perceptible from the results that the Logistic Regression obtained better results than the Naïve Bayes classifier with the same CountVectorizer settings. This could result from Naïve Bayes' simplifying assumption that the events being looked at are independent from each other. Therefore, it is based on the calculation of likelihood, rather than the actual probability as in a Logistic Regression.