DAT410: Group 60 Assignment 4

Cecilia Nyberg
Personal number: 990106
Program: MPDSC
Hours spent: 20
`cecnyb@chalmers.se`

Elin Stiebe
Personal number: 000210
Program: MPDSC
Hours spent: 20
`elinsti@chalmers.se`

March 10, 2024

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part in other solutions.

1 Implementation N-gram Algorithms

The purpose of this project is to create a way to generate motivation for Nobel prize winners. The dataset used was found on kaggle.com. The set contains Nobel prize winners from all years when the prize has been dealt out (1901-2023). The idea is to use N-gram models on the existing Nobel prize motivations to generate new motivations.

Each of the implemented algorithms comes in two flavors. In version 1 the 15 most probable bigrams/trigrams are extracted from the entire set. Then each of the new words in the bigram or trigram is added to the so-far generated sentence. The word that will be added to the sentence will be the word that creates the sentence with the highest probability to exist. In version 2, 15 existing bigrams/trigrams are chosen at random. After that, each new word in the n-grams is added to the generated sentence and evaluated with respect to the probability of that sentence. Once again, the word creating the most probable sentence is added. The algorithm's behavior is decided by a boolean argument value *sorted*. If sorted is set to true the flow displayed in figure 1 will occur. The top 15 most common bigrams are selected, compared with their respective probability to be in the sentence built this far, then the most probable bigram is selected and the new word is used as the following word of the sentence.

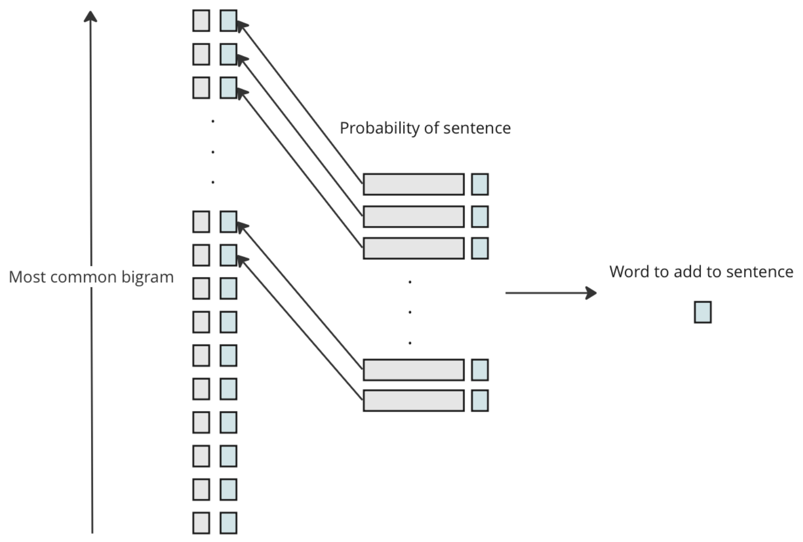

Figure 1: Logic when sorted is set to true.

If sorting is set to false, the logic in figure 2 will occur instead. This can lead to the 15 bigrams/trigrams considered being very different every time and thus these implementations have a larger inherent randomness. Instead of selecting the overall most common bigrams, it selects 15 random bigrams which it then compares with their probability to be in the sentence before the most probable of those are outputted and appended to the sentence.

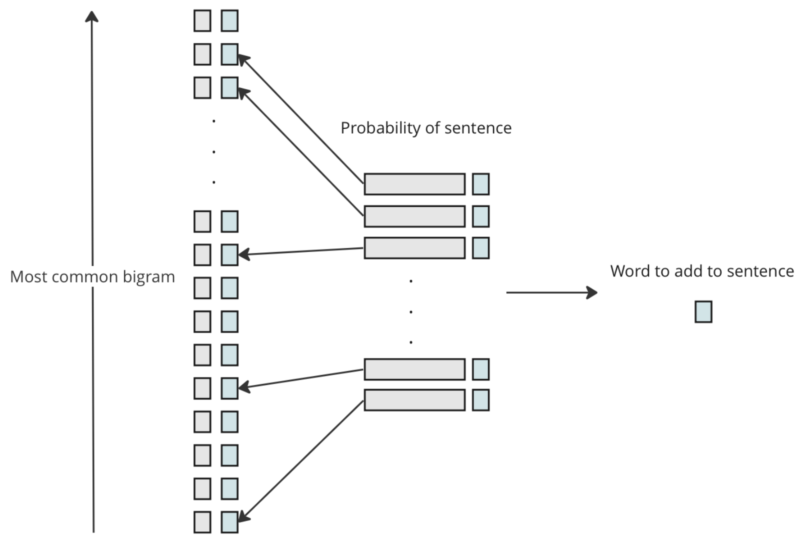

Figure 2: Logic when sorted is set to false.

The very first strategy, shown in subsection 1.1, only comes in one variant since it behaves purely randomly.

1.1 Randomly generated motivation

This randomly generated motivation was used as a baseline to compare all other algorithms. The baseline is used to see whether any of the models would produce meaningful motivations. The implementation of the random generation is shown below in listing 1.

```
def random_motivation(df):
    bigram_count = bigram_counter(df)
    words = []
    word = '#START#'
    while word != '#END#':
        words.append(word)
        next_word = random.choices(list(bigram_count.keys()),
                    list(bigram_count.values()))
        word = next_word[0][1]
    return ' '.join(words[1:-1])
```

Listing 1: Code for randomly generated motivation

Below is an example output for the randomly generated motivation. It can be seen that this motivation does not follow the structure or grammar of a sentence, nor does it make sense on a semantic level. It is simply words from the data set put together in a random order.

its and in by extremely on the their

Table 1: Random motivation

1.2 The most Probable Motivation using Bigram

Next, the implementation of an algorithm that derives the most probable motivation with bigrams given the underlying data. The full implementation is noted in listing 2. To repeat, the *sorted* argument specifies whether the bigrams should be sorted before the top 15 words are selected. It is the 15 bigrams that are compared and decided between, to determine which should be the word to follow. If sorted = false it is not the overall most common bigram that should be selected but the most probable to be in the sentence of a random selection of bigrams.

```python
def most_probable_motivation(df, sorted = True):
    bigram_count = bigram_counter(df)
    if sorted:
        bigram_count = sort_dictionary(bigram_count)
    sentence = ''
    word = '#START#'
    sentence += (word)
    while word != '#END#' :#and len(sentence.split(' ')) < 10:
        all_bigrams_with_word = [bigram for bigram in bigram_count.keys() if
                                        bigram[0] == word]
        if not sorted:
            # take 15 random bigrams from possible_15_words
            length = len(all_bigrams_with_word)
            if length >= 15:
                possible_15_words = random.sample(all_bigrams_with_word, 15)
            else:
                possible_15_words = random.sample(all_bigrams_with_word, length)
        else:
            possible_15_words = all_bigrams_with_word[:15]

        if len(possible_15_words) == 0:
            most_common_words = sort_dictionary(dict(islice(
                            get_word_count_dict(motivations).items(), 15)))
            for common in most_common_words:
                possible_15_words.append((word, common))

        possible_sentences = [sentence + ' ' + word[1] for word in
                                possible_15_words]
        probabilities = [probability_of_sentence_bigram(sentence, df) for
                                    sentence in possible_sentences]

        word = possible_15_words[probabilities.index(max(probabilities))][1]

        sentence += ' ' + (word)
    return sentence
```

Listing 2: The algorithm generates the most probable motivation.

Now, let's look at its outputs noted in table2. First, sort is set to True, the result that follows is deterministic. One interesting thing to note is the used pronoun "his". The very simple algorithm seems to be biased toward men. This is of course due to the data, it seems more men than women have won the Nobel prize this far. Now sort is set to False. In this case, there is a possibility to get several different outputs, three of which are noted in table 2.

Value Sort	Output
True	#START# for his discovery of the development #END#
False	#START# for their discoveries concerning the fields of neutron scattering techniques for his services he has opened a tribute to important reagents in connection with special reference to our understanding and human values #END#
False	#START# for their discovery of nuclear magnetic resonance states, made therewith in our understanding and methods within dna-based chemistry; for having developed independently, on the structure determination of importance to his analysis #END#
False	#START# for pioneering contributions to the discovery of his services he has made therewith in particular #END#

Table 2: Most probable motivation outputs.

1.3 Introducing more Randomness

In this implementation, one of the top three most probable words is selected randomly to be the next word in the sentence. The full implementation is in listing 3. This implementation has the same argument *sorted* with the same behavior as noted in section 1.2. A random word is selected out of the top 3 most probable words to follow in the sentence among the 15 extracted words that were extracted using sorted=True or False.

```python
def most_probable_motivation_with_random(df, sorted = True):
    bigram_count = bigram_counter(df)
    if sorted:
        bigram_count = sort_dictionary(bigram_count)
    sentence = ''
    word = '#START#'
    sentence += (word)
    while word != '#END#':
        all_bigrams_with_word = [bigram for bigram in bigram_count.keys() if
                                        bigram[0] == word]
        if not sorted:
            # take 15 random bigrams from possible_15_words
            length = len(all_bigrams_with_word)
            if length >= 15:
                possible_15_words = random.sample(all_bigrams_with_word, 15)
            else:
                possible_15_words = random.sample(all_bigrams_with_word, length)
        else:
            possible_15_words = all_bigrams_with_word[:15]

        if len(possible_15_words) == 0:
            most_common_words = sort_dictionary(dict(islice(
                         get_word_count_dict(motivations).items(), 15)))
            for common in most_common_words:
                possible_15_words.append((word, common))

        word_and_prob = {}
        for select in possible_15_words:
            word_and_prob[select[1]] = probability_of_sentence_bigram(sentence
                         + ' ' + select[1], df)

        word_and_prob = sort_dictionary(word_and_prob)
        # Some words don't have three options for their next word
        lenght = len(word_and_prob)
        lenght = (lenght if lenght < 3 else 3)
        top_three_words = list(word_and_prob.keys())[:lenght]
        word = np.random.choice(top_three_words)
        sentence += ' ' + (word)
    return sentence
```

Listing 3: Code for randomly generated motivation

Three example outputs are shown below in table 3. Each of the outputs were generated when sorted was set to True or False.

Value Sort	Output
True	#START# for his discovery and their discoveries of a large hadron collider #END#
True	#START# in the development and for their contributions to important literary achievements, characterized as well as they appeal to find a large number of his discovery relating to combat hunger as some fundamental symmetry which with special recognition as an epic descriptions of economic fluctuations and especially lupus vulgaris, with particular regard to create peace prize laureates represent civil war crimes human rights in connection #END#
True	#START# for his discovery of a tribute to the theory #END#
False	#START# for having made his development #END#
False	#START# for integrating climate change #END#
False	#START# who emulates the domain of molecular mechanisms in cell membranes; for their discovery relating to resolve international conflicts, to a weapon of radioactive substances, and deepened insight into long-run macroeconomic policy #END#

Table 3: Result of taking a random word of top three bigrams.

1.4 Trigrams

It is also possible to use trigrams instead of bigrams, to capture a broader relationship between words in a sentence. The full implementation is in listing 4.

```python
def most_probable_motivation_trigram(df, sorted = True):
    sorted_val = sorted
    trigram_count = trigram_counter(df)
    if sorted:
        trigram_count = sort_dictionary(trigram_count)
    sentence = ''
    last_word = '#START#'
    sentence += (last_word)
    while last_word != '#END#' :
        word_to_add = ''
        if len(sentence.split(' ')) == 1 :
            word_to_add = use_bigram_instead(last_word, sentence, sorted_val, df)
        else:
            second_to_last_word = sentence.split(' ')[-2]
            all_trigrams_with_word = [trigram for trigram in trigram_count.keys()
                if trigram[1] == last_word and trigram[0] == second_to_last_word]

            if not sorted:
                length = len(all_trigrams_with_word)
                if length >= 15:
                    possible_15_words = random.sample(all_trigrams_with_word, 15)
                else:
                    possible_15_words = random.sample(all_trigrams_with_word,
                                                      length)
            else:
                possible_15_words = all_trigrams_with_word[:15]

            if len(possible_15_words) == 0:
                word_to_add = use_bigram_instead(last_word, sentence, sorted_val,
                                                 df)
            else:
                possible_sentences = [sentence + ' ' + word[2] for word in
                                      possible_15_words]
                probabilities = [probability_of_sentence_trigram(sentence, df)
                                 for sentence in possible_sentences]
                word_to_add = possible_15_words[probabilities.index(
                                              max(probabilities))][2]
        sentence += ' ' + (word_to_add)
        last_word = word_to_add
    return sentence
```

Listing 4: Code for generation of motivations using trigrams.

The outputs are noted in table 4. The first one is deterministic as expected, but the second version where sorted is set to false has a range of outputs. Three of the outputs are noted in table 4.

Value Sort	Output
True	#START# for his discovery of the extraordinary services he has rendered by their researches into the constitution of haemin and chlorophyll and especially for his work on information and communication technology; for developing semiconductor heterostructures used in high-speed- and opto-electronics #END#
False	#START# for his contributions to the development of neutron scattering techniques for studies of condensed matter; for the rights of women and children #END#
False	#START# for their discovery of the human condition #END#
False	#START# for having developed and applied dynamic models for complex chemical systems #END#

Table 4: Result using trigram to calculate probabilities.

1.5 Accounting for category

The data consisted of noble prizes for six different categories. This far, the generated motivation does not account for the subject but looks the same regardless of category. To account for the subject, words important for a specific category were extracted. This was done by calculating the occurrence of every word in motivations within a specific subject and dividing this by the occurrences of that word in all subjects. The code can be seen in listing 5.

```python
#calculate most common words for each category that are not in general
def get_top_15_words(word_count, word_count_general):
    top_15_words = {}
    for word in word_count:
        top_15_words[word] = probability_of_word_in_category(word,
                                word_count, word_count_general)
    top_15_words = sort_dictionary(top_15_words)
    return list(top_15_words.keys())[:15]

def probability_of_word_in_category(word, word_count, word_count_general):
    if word not in word_count:
        return 0.01
    return max(word_count[word] / word_count_general[word], 0.01)
```

Listing 5: 15 most distinct words for category

This was integrated with the method for the most probable motivation using bigram. The 15 most distinct words for a category were extracted and combined with the 15 most probable bigrams. In total 30 different words were investigated as candidates for the next one in the sentence. The new probability for each word was calculated as a combination of the probability of the sentence and the probability of the word in the category. If a word was not occurring in the category dictionary or the probability was very small, 0.01 was returned. This factor can be increased or decreased depending on how much influence the categorical aspect should have. If it is increased, the difference between common and uncommon words in a category becomes smaller so that the categorical aspect has less impact. The implementation can be seen in listing 6.

```python
def most_probable_motivation_for_category(df, df_cat, sorted = True):
    word_count_general = get_word_count_dict(df)
    word_count_cat = get_word_count_dict(df_cat)
    common_words_for_cat = get_top_15_words(word_count_cat, word_count_general)
    bigram_count = bigram_counter(df)
    if sorted:
        bigram_count = sort_dictionary(bigram_count)
    sentence = ''
    word = '#START#'
    sentence += (word)
    while word != '#END#' :
        possible_words = []
        all_bigrams_with_word = [bigram for bigram in bigram_count.keys() if
                                bigram[0] == word]
        if not sorted:
            length = len(all_bigrams_with_word)
            if length >= 15:
                possible_15_words = random.sample(all_bigrams_with_word, 15)
            else:
                possible_15_words = random.sample(all_bigrams_with_word, length)
        else:
            possible_15_words = all_bigrams_with_word[:15]
        possible_words = possible_15_words

        for word in common_words_for_cat:
            possible_words.append((possible_15_words[0][0], word))
        possible_sentences = [sentence + ' ' + bigram[1] for
                    bigram in possible_words]

        probabilities = []
        for sent in possible_sentences:
            prob = probability_of_sentence_bigram(sent, df) *
            probability_of_word_in_category(sent.split()[-1], word_count_cat,
                                word_count_general)
            probabilities.append(prob)

        word = possible_words[probabilities.index(max(probabilities))][1]
        sentence += ' ' + (word)

    return sentence
```

Listing 6: Generated motivation with category

The difference was notable and some output can be seen in table 5. The outputs are much more context-specific, but the sentence structure is slightly worse than before. Currently, the categorical aspect is only combined with the bigram approach, but it can be combined with the other versions as well.

Category	**Motivation**
Chemistry	#START# for his researches into the discovery of organic chemistry #END#
Literature	#START# for an epic writer #END#
Peace	#START# for their efforts to the border conflict in Guatemala on several continents #END#
Physics	#START# for theoretical physics, in particular those phenomena discovered by crystals #END#
Medicine	#START# for their discoveries concerning the chromosome in certain psychoses #END#
Economics	#START# for analyzing economic theory of econometric models and applied dynamic macroeconomics: the scientific work on our understanding static actively contributed to alleviating global poverty #END#
General	#START# for his discovery of the development #END#

Table 5: Motivations for Various Categories

2 Evaluation

Now to the evaluation of the outputs. Were any of the models successful in generating Nobel Prize motivations? Evaluation of generated text is difficult since there are many different ways to phrase the same thing. A good evaluation should reasonably consider both the grammar and structure and the meaning of the generated text. Section 2.1 evaluates the motivations using the BLEU score while section 2.2 takes a semantic approach to evaluation.

2.1 BLEU

BLEU is one of the most common evaluation metrics according to Dayal, 2020. BLEU evaluates the effectiveness of the generated text by comparing n-gram overlaps with some optimal output according to some set weights. In this case, the model was trained on all motivations except for one year *1990*. The motivations from this year were kept as a means for evaluation. 1990 contains motivations for each type of award and is in between the older and newer data (1901 - 2023) meaning they will hopefully be representative motivations. Each of the models' outputs was compared to every given motivation in the evaluation set. The motivations from the evaluation set are displayed in table 6

Price	**Motivation**
Chemistry	for his development of the theory and methodology of organic synthesis
Economics	for their pioneering work in the theory of financial economics
Literature	for impassioned writing with wide horizons, characterized by sensuous intelligence and humanistic integrity
Medicine	for their discoveries concerning organ and cell transplantation in the treatment of human disease
Peace	for his leading role in the peace process which today characterizes important parts of the international community
Physics	for their pioneering investigations concerning deep inelastic scattering of electrons on protons and bound neutrons, which have been of essential importance for the development of the quark model in particle physics

Table 6: Evaluation data.

The generated sentences are not displayed since each of the BLEU-scores displayed in table 7 are means taken by generating 10 different sentences and comparing each sentence to the 6 evaluation sentences in table 6. A BLEU score is the best if it is equal to 1 and the worst if it is equal to 0. There is no single model close to a score of 1 in table 7, they all seem to have quite significant discrepancies. The minimum value for all of the models was very close to 0 and were thus rounded down to 0, meaning at least one of their produced sentences had almost no n-gram overlaps with the price motivations within all categories for 1990. The sorted bigram had a max of about 0.41. This might have been explained by it using very common bi-grams such as "for their" or "for his" which certainly has an overlap in the nominations of the evaluation set.

Algorithm	**Mean**	**Median**	**Min**	**Max**
Random	0.0759	0	0	0.5969
Most Probable (Bigram, sorted)	0.1906	0.1685	0	0.4061
Most Probable (Bigram, unsorted)	0.1166	0.1177	0	0.3698
Most Probable with Random (Bigram, sorted)	0.1003	0.1005	0	0.3767
Most Probable with Random (Bigram, unsorted)	0.0884	0	0	0.7071
Most Probable (Trigram, sorted)	0.0560	0.0579	0	0.0998
Most Probable (Trigram, unsorted)	0.0913	0.0907	0	0.3648
Most Probable (Category and bigram, unsorted)	0.0651	0.0251	0	0.3615

Table 7: Results of BLEU evaluation.

The weights used in BLEU were all 0.25, this means that the model will set equal importance to unigrams, bigrams, trigrams, and those above. This distribution of weights was decided on since the models are all built using bigrams or trigrams, meaning putting a higher weight on bigrams would give an advantage to some of the models. Table 9 displays the change in BLEU score of the model from section 1.2 "Most probable bigram sorted" when the weights are set to favor different values of n n-grams. This model was chosen since it had the highest mean and median from table 7. The model is deterministic and has an output like in table 8.

Value Sort	Output
True	#START# for his discovery of the development #END#

Table 8: Most probable motivation output.

Unigrams are favored in the first line and these results are also the highest. This indicates that the generated sentence uses a lot of similar words as is used in the evaluation data set. Bigram is lower, yet still higher than when using the standardized weights (0.25, 0.25, 0.25, 0.25) which indicates that the model uses relevant bigrams. This is also reasonable due to how the model is built, using just bigrams. Row three in table 9 shows when trigrams are favored, this score is close to the score displayed in table 7 and is significantly lower than uni- and bigram. The last row shows then BLEU only considers higher grams.

Weights (unigram, bigram, trigram, higher-gram)	Mean	Median	Min	Max
(1, 0, 0, 0)	0.5694	0.5833	0.4722	0.6390
(0, 1, 0, 0)	0.3857	0.3857	0.2286	0.5429
(0, 0, 1, 0)	0.1912	0.1912	0.0294	0.3529
(0, 0, 0, 1)	0.1111	0.0909	0	0.2424

Table 9: Results of BLEU evaluation weights are set to favor bigrams.

The fourth row in table 9 displays low scores. It is not as common to have high scores for n-gram overlaps when n becomes large. This has logical explanations in that every meaning can be written in vastly different ways. This brings us to a con with BLEU, that it only covers the similarities for the words in the text and leaves out any semantic similarities. For example, if synonyms are used this can become a problem since a generated text with words meaning the same thing as words in the text it is compared to will get a low BLEU score, which might be unfair. On the other hand, since Nobel Prize motivations are expected to follow a certain language style (e.g. formal language), there is reason for comparing at a word level. Another evaluation approach follows in section 2.2.

2.2 Semantic similarity score

The semantic similarity score was used as a complement to the BLEU score since it captures the semantic meaning of a motivation instead of comparing the words. This was calculated the same way as the BLEU score, but first, embedded the sentences using a sentence transformer and then using the semantic similarity instead of BLEU. The scores are displayed in table 10. Several of the max scores reach quite high levels where the category-specific algorithm reaches the highest max of 0.56. The deterministic sentence of table 8 still has the highest mean and median just as in section 2.1 with BLEU. It seems "for his discovery of the development" is quite a good motivation.

Algorithm	Mean	Median	Min	Max
Random	0.1802	0.1739	-0.0342	0.4471
Most Probable (Bigram, Sorted)	0.3063	0.3047	0.1810	0.4585
Most Probable (Bigram, Unsorted)	0.1925	0.1861	-0.0092	0.4328
Most Probable with Random (Bigram, Sorted)	0.2144	0.2090	0.0177	0.4470
Most Probable with Random (Bigram, Unsorted)	0.2036	0.1968	-0.0416	0.5702
Most Probable (Trigram, Sorted)	0.2003	0.1787	0.0983	0.3128
Most Probable (Trigram, Unsorted)	0.1857	0.1414	-0.0413	0.5066
Most Probable (Category and Bigram, Unsorted)	0.1566	0.1188	-0.0935	0.5624

Table 10: Results

3 Conclusion and Discussion

This paper focused on n-gram models for predicting Nobel prize motivations. The deterministic model using bigram was able to create the best motivation in accordance to evaluations performed with BLEU and a semantic approach. Its motivation was "for his discovery of the development". This motivation is very general and lacks any personal touch. The evaluation set was diverse in the fact that it contained a motivation from each type of prize, though it would have been preferable to have additional motivation sets to compare to. Further, this specific sentence might have scored high specifically because it was general and had similarities with many of the evaluation sentences.

The trigram or category-specific model seemed to perform the best when evaluating it as a human reader. The one containing the category could give an understanding that the prize the person was getting was related to science and not just any prize. The trigram model's motivations were readable and interesting to read. One issue with them was that they switched categories quite drastically. For example "for his contributions to the development of neutron scattering techniques for studies of condensed matter; for the rights of women and children", this prize winner seems to have won both the physics and peace prize. It is interesting to see just how good motivations the models were able to achieve with very little data. This is probably due to the small scope the data and generation of motivations operate within. A more general model would not be able to get such good results with that little data.

Some aspects that would have been nice to implement if time was not a limit would be; gender neutrality, and the possibility to insert and choose names or specific scientific achievements in a motivation. An important aspect to consider is that the reason behind someone getting a noble prize is very important for the motivation. This was not considered in this project. For example, using a transformer could be a solution to this. Information regarding what accomplishment the prize was for could be sent to the model and a motivation could be generated.

References

Dayal, D. (2020). *How to evaluate text generation models: Metrics for automatic evaluation of nlp models*. Medium. https://towardsdatascience.com/how-to-evaluate-text-generation-models-metrics-for-automatic-evaluation-of-nlp-models-e1c251b04ec1