FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Micha de Rijk

Codenames: a practical application for modelling word association

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	RNDr. David Mareček, PhD. Dr. Gosse Bouma
Study programme:	Computer Science
Study branch:	Computational Linguistics

Prague 2020

Here I want to thank everyone who helped me on the road to completing this thesis. I would not have managed it without your help, support and encouragement.

First, I would like to thank my local supervisor David Mareček for his patience and excitement for this thesis and for all the invaluable feedback and ideas he gave me throughout. I would also like to thank Gosse Bouma, my supervisor from Groningen, for his interesting suggestions.

I am grateful to all my friends who supported me throughout my time in Prague. I would like to thank Mei for the adventurous hikes, studying together for difficult exams and keeping me sane when times were tough. Thank you Nastya for your much needed advice and skepticism, you taught me a lot about myself. I would also like to thank Akshay for sharing his experiences with me and the many games of pool, you made me feel much better about myself. I am thankful to Rupal for the many walks, always having time to talk and dragging me to new places. Lastly, I want to thank my dear friend Taico for believing in me and providing the work space in which most of this thesis was written.

Title: Codenames: a practical application for modelling word association

Author: Micha de Rijk

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. David Mareček, PhD., Institute of Formal and Applied Linguistics, Dr. Gosse Bouma, Faculty of Arts

Abstract: Word association is an important part of human language. Many techniques for capturing semantic relations between words exist, but their ability to model word associations is rarely tested. We introduce the game of Codenames with one human player as a word association task to evaluate how well a language model captures this information. We establish the baseline f-score of 0.362 and explore the performance of several collocations and word embedding models on this task. Our best model uses fastText word embeddings and achieves an f-score of 0.789 for Czech and 0.751 for English.

Keywords: computational word association, codenames, pointwise mutual information, word embeddings

Contents

1. Introduction

In recent years a lot of progress has been made in the field of distributional semantics. Since the introduction of word embeddings many improvements and variations have been made to try to capture as much information as possible. While word embeddings are great at capturing semantic similarity, human word association is not limited to this type of relation alone. The words *ice* and *cream* in a collocate such as *ice cream* and co-hyponyms like *knife* and *fork* are just as well associated with one another as synonyms such as *autumn* and *fall*.

It would be useful to know whether our current language models such as word embeddings and collocations can adequately capture these associations. For this purpose we introduce a new word association task called Codenames, based on a popular word association game of the same name.

We are aware of one earlier attempt in this area by Obrtlík [2018]. We improve upon their work by simplifying the task to a game with only one human player and a word association model. To show whether such a task is a good test of a language model, we build several models that try to capture these associations using different techniques and evaluate their performance on the task.

Our aim is to 1) turn the game of Codenames into a word association task suitable for the evaluation of computational models of word association and 2) evaluate the performance of several baseline models.

We cover the different types of word associations more extensively in Section 1.1. The basics of the game are discussed in Section 1.2 along with the simplification to one human player. The structure of this thesis is explained in Section 1.3.

1.1 Word association

Word association is the mental network through which we associate words words with one another. Words can be associated in several ways:

- **Synonym**: A synonym is a word that expresses the same concept as another word. For example *boat* and *ship*, which are both used to denote a vessel for traveling on water.

- **Hypernym**: We say that A is a hypernym of B if A describes a set of concepts that B belongs to. For example, *fruit* is a hypernym of *apple* because *apple* belongs to the set of objects described by the word *fruit*.

- **Hyponym**: The opposite of a hypernym. A is a hyponym of B if A belongs to the set of concepts expressed by B. The word *apple* is a hyponym of *fruit*, because *apple* is a type of *fruit*.

- **Co-hyponym**: The co-hyponym relation refers to words that have a hypernym in common such as *knife* and *fork* which are both hyponyms of the word *utensil*.

- **Meronym**: A meronym is a word that is a part or member of another. For example, *sentence* is a meronym of *text*, because a sentence is usually part of a text.
- **Holonym**: A holonym is the opposite of a meronym. *Text* is a holonym of the word *sentence*.
- **Collocate**: A collocation is a set of words that co-occur together more often than would be expected by random chance. The individual memebers of such a set are called collocates. For example, the individual words *code* and *source* are collocates because of their frequent co-occurrence in the compound word *source code*.

Word associations can vary in strength based on the direction of association. For example, the word *Eiffel* would be very strongly related to the word *tower*: when someone says *Eiffel*, *tower* immediately springs to mind. However, this relation does not hold as strongly when inverted. If someone says *tower*, words like *building* and *tall* spring to mind much more quickly than *Eiffel*. Similarly, *brick* is related to *tower*, but not to *Eiffel*. As such, word association cannot be treated as a symmetric relation.

Modelling these different types of word associations computationally is very challenging. There are many ways in which words can be associated. Gathering all of these associations for each individual word in a language is an immense task. In fact, we argue that it is infeasible to encode all such relations in manually constructed ontologies and databases. (see Section 2.3 for more discussion on this topic)

As a result, we will have to use a different source for our relations and find a way to extract them that does not require manual annotation. There exist several unsupervised methods which solve this problem using large amounts of text as a source. We explore two of these methods in this thesis, namely word embeddings and collocations.

1.2 Codenames

The task of word association is the retrieval of associated lexical items in response to a word prompt. In order to make this task more appealing to participants, we test word association in the context of a word association game called Codenames.

Codenames is a word association board game created by Vlaada Chvátil. [1] It is played in two teams of 2 or more players, each team has one spymaster and one or more agents. The game board consists of 25 cards with a word written on it. There are nine cards that belong to the team that starts the game, eight that belong to the opposing team, seven neutral cards and one assassin card, which loses the game for the team that selects it. Both spymasters get to see which cards belong to which team, but the agents do not. Each turn one of the spymasters gives a hint[2] to their agents for one or more cards that belong to their team. The spymaster also gives a number that signals to how many of their

[1] `https://czechgames.com/en/codenames/`

[2] Hints are referred to as *clues* in the board game

own cards the hint is related. The agents then proceeds to guess cards until they select one that does not belong to their team or they voluntarily end their turn because they do not see any more cards that are related to the hints that their spymaster has given.

The goal of the game is to turn over all of the cards that belong to your team. As a spymaster you help achieve this goal by giving hints to your agents that are as associated with your own cards and unambiguous as possible. As an agent you will try to turn over as many cards of your own team using the hints given by your spymaster, without selecting any cards that do not belong to your team and avoiding the assassin at all cost. The game ends when either team has turned over all of their cards, in which case that team wins, or one of the teams has selected the assassin in which case that team loses.

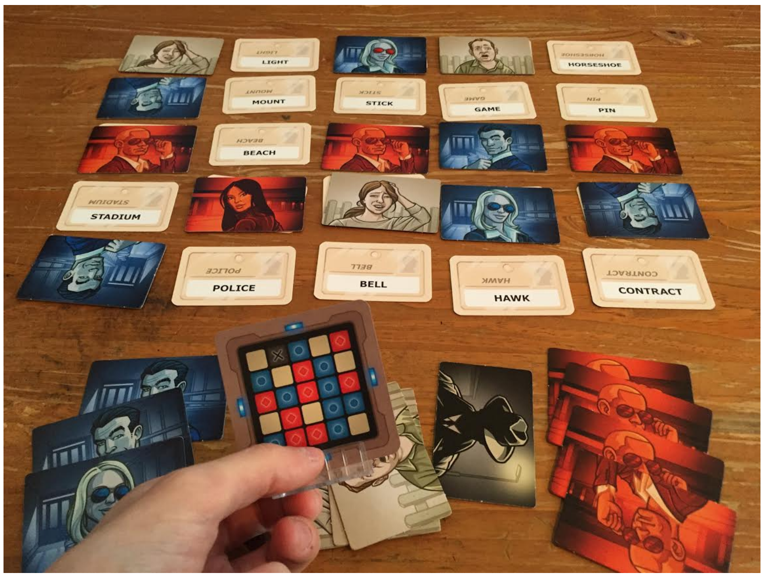


Figure 1.1: A regular game of Codenames[3]

A regular game of Codenames along with the board pieces used to play the game can be seen in Figure 1.1. The blue and red spy cards are used to cover words that have been selected during the game. In the picture the spymaster is holding the card that shows which cards belong to which team. The blue lights on the side of the card indicate that the blue team starts the game and therefore has 9 cards to turn over while the red team has only 8 cards they need to turn over in order to win the game.

There are some restrictions to the hint that the spymaster can give:

- The hint has to be one word. In other words, any word with a space or hyphen is invalid. The compound word *checkpoint* is a valid hint, even though it consists of two separate morphemes *check* and *point*. The words *part-time* and *lottery ticket* are not.

- The hint cannot contain part of a compound word that is on the board. If one of the cards on the table that has not been selected yet is *snowman*, it disallows hints such as *snow*, *man* and *snowball*.

- The hint cannot be any morphologically related form of a visible word on the board. If one of the on the table is *fly* and it has not been selected yet, it disallows hints such as *fly*, *flown*, *flight* and *butterfly*.

The game has some additional rules that we will not explain here because we do not use them in our implementation of the game. The rules that are excluded either complicate the analysis of the results without good reason or make the game less enjoyable to play, which contradicts the purpose of using a game to increase the willingness of participants to take part in the experiment.

Now for the part of the game that is most relevant for this thesis: the hints. The aim of the spymaster is to provide hints that are related to the cards belonging to their team. When playing the game with other people, it can already be exciting to give a good hint for multiple words, say *mozart, 3*, and even more so to correctly guess *symphony*, *concert* and *piano* when you get this hint as an agent. What if we could build a computational model that gives such hints?

The game is available in many languages, but we focus our efforts on English and Czech because these languages are well represented in our group of participants.

1.2.1 A single human player

There also exists a two player variant, which is detailed in the rule book of the board game.[4] We adapt this two player version into a version for one human player and one AI by replacing the player who plays as spymaster by an AI who selects hints using a word association model.

The game is made more regular by putting the AI and the player on the same team and introducing a dummy team that opposes them. The dummy team turns over one of their own cards during their turn and then passes the game back to the player.

In this variant, the player is always on the starting team. This way they start with 9 cards instead of 8, which means they will have to guess more cards so we get more data. This also makes the game more regular by not randomly letting either the dummy or the player's team start, which would introduce another unnecessary factor that we would have to take into account during the analysis.

1.3 Structure of the thesis

We first discuss the theoretical background necessary for the Codenames word association task in Chapter 2, such as the rules of the game and different approaches to computational models of word association. We then discuss the structure of our models, including data used and relevant techniques such as aggregation in Chapter 3. In Chapter 4 we give a brief overview of the application and the implications that some implementation details have for our experiments. In Chapter 5 we provide a detailed analysis of the performance of several collocation and word embedding models, a comparison between collocation methods based on sentence-level and dependency-level bigrams and two attempts at a good ensemble model. In Chapter 6 we provide a concise overview of our findings and mention several promising directions for future research.

[4]`https://czechgames.com/files/rules/codenames-rules-en.pdf`

2. Background & Literature

In this section we introduce the theoretical background on which our models are based. We first introduce the notions of word embeddings in Section 2.1 and pointwise mutual information in Section 2.2. We then discuss two manual approaches of encoding word association in Section 2.3 and conclude with a summary of a similar work by Obrtlík [2018] who explored a word embedding model for Codenames with two human players in Section 2.4.

2.1 Word embeddings

We start with a fixed vocabulary V, with $n = |V|$. For every word in V we can create a one-hot encoding, which is an n-dimensional vector with many 0s and one 1 representing the word. These one-hot encodings do not have any overlapping 1s amongst each other in any of the dimensions. However, many words are semantically related, like *dog* and *cat*, since they are both animals. The idea is to condense these one-hot vectors into a shorter d-dimensional vector. These vectors are called word embeddings.

To create such word embeddings efficiently, Mikolov et al. [2013] introduce the skip-gram model with negative sampling. Since then many additions to this technique have been proposed, such as adding topic information [Liu et al., 2015] or deriving the embeddings from dependency-based contexts [Levy and Goldberg, 2014]. For our word association model we use word embeddings enriched with subword information as described by Bojanowski et al. [2017]. This method is called fastText and adapts the skip-gram model with negative sampling to represent a word as a combination of the character n-grams it contains.

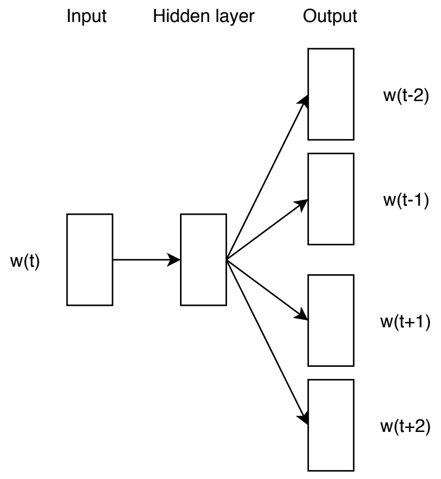


Figure 2.1: Schematic of a skip-gram model, adapted from Mikolov et al. [2013]

The original skip-gram model works by training a network with one hidden layer and a softmax layer to perform a multi-class classification task. For a given input word w, the task is to predict for each word in the vocabulary the probability that this word occurs in the context of w. Figure 2.1 provides a general picture of a skip-gram model. The input is represented by $w(t)$, the word at position t in the training corpus. The hidden layer is the d-dimensional vector that we want to train and $w(t-2)$, $w(t-1)$, $w(t+1)$ and $w(t+2)$ represent the surrounding words that the model has to predict.

Negative sampling proposed by Mikolov et al. [2013] speeds up this process significantly by simplifying the task to a binary classification task. We take a set of positive words from the context, given a certain window size and we take a subset of negative words that do not occur in the context, sampled randomly from the vocabulary. These samples provide the supervised data for the binary decision task that we use to train the model. The model gets an input word w and has to predict for each word in the sample whether it occurs in the context of w or not. Based on this task it learns a representation for each word, encoded in the hidden layer of the network.

Bojanowski et al. [2017] expand on this by representing a word not as a single vector, but through the sum of the vectors of its character n-grams. To obtain these n-gram vectors, we start by adding symbols $<$ and $>$ to mark word boundaries to distinguish suffixes and prefixes from characters sequences that occur in the middle of a word. We then extract all n-grams for $3 >= n <= 6$ and learn the vector representations for each character $n - gram$. After this we can calculate a word vector by summing the vector representations of its individual character n-grams.

The benefit of this approach is that the representations of character n-grams are global and shared between all words, so we obtain more accurate representations for rare words.

2.2 Information theory measures

Pointwise mutual information (PMI) is a measure of assciation used to find collocations, i.e. groups of two or more words that co-occur more often in a text than would be expected from random chance. Given two words a, b, we define $p(a,b)$ as the probability that b occurs after a, we calculate this using $p(a,b) = \frac{count(a,b)}{\sum_{x,y \in T} count(x,y)}$, where T is the set of words seen in the text and $count(x,y)$ is the number of times y occurs after x. $p(a)$ is the chance of seeing a word x in the text, we calculate this probability using $p(a) = \frac{count(a)}{\sum_{x \in T} count(x)}$. We then define the measure of pointwise mutual information as follows:

$$PMI(a,b) = \log_2 \frac{p(a,b)}{p(a)p(b)} \tag{2.1}$$

Church and Hanks [1990] lay the ground work for computational word association as a replacement for extracting word associations from participants in psycholinguistic experiments. They use pointwise mutual information to extract related words from the COBUILD corpus. They provide a thorough analysis of the relations encoded by their model, but they lack an evaluation of their model in regards to word associations made by humans.

Information theory offers several other measures of association that capture the distributional characteristics of words in a text. Wettler and Rapp [1993] explore several of these association measures and also evaluate their results by comparing the predictions of their system to responses elicited from a group of students. They do so for both English and German and their system achieves performance comparable to that of the average individual in the group of students.

More recently, Enguix et al. [2014] have used a simple graph-based algorithm to extract simple bigram counts. They filter the corpus to include only lemmatized adjectives, nouns and verbs and construct a graph with the words as nodes and weighted edges. The weight of an edge is determined by the number of times the two words (nodes) co-occur in the text. They compare the predictions made by their system to responses elicited from a group of 100 students and achieve impressive results: the associations made by the system overlap with associations made by the group of students slightly more often than the associations made by the average student of the group. The number of subjects that answered with the prediction made by the system was 6.2 on average, while the number of other subjects that come up with the answer of an average test person is 5.8. Associations made by their system include *frustration* given *anger* and *taste* for *bitter*.

In this work we will continue the trend of using information theoretic measures to model word association. We will use pointwise mutual information to extract collocates from a large corpus of text. The novelty of our approach compared to previous work is that we will evaluate the performance of these models on the word association game Codenames.

2.3 Association databases

One of the interesting approaches for computational word association that we considered is the use of ontologies and databases. We could rely completely on the word associations provided by manually entered data to build our models. We detail two of these approaches below.

2.3.1 WordNet

WordNet is a collection of synsets grouped into a semantic hierarchy. [University, 2010] A synset is a collection of one or more words with the same meaning, i.e. synonyms. The synsets, hypernyms and hyponyms that WordNet captures can provide good candidates for word association. Because of the information it encodes, it excels at strict relations such as hypernyms, hyponyms, meronyms and holonyms. This would be a great addition to our application and a fruitful area for future research. For example when considering countries, *continent* would be a useful hint when both *Africa* and are given *Australia*. However, it falls flat when considering more free associations such as *Frodo* and *ring*, which cannot be classified as either hypernym, hyponym, meronym or holonym and are thus not captured in WordNet. We therefore prefer several data-driven methods such as word embeddings and pointwise mutual information, which we explore in the rest of the thesis.

WordNet is not suitable for our purposes because it does not capture as many relations as we would like and is not as extensible as data-driven methods, which might capture even pop culture references such as the relation between *Frodo* and *ring*.

2.3.2 University of South Florida Free Association Norms

The University of South Florida Free Association Norms is a database of free associations containing 72,000 word pairs collected from almost 750,000 responses produced by over 6,000 participants. More than 5,000 stimulus words were tested. [Nelson et al., 2004] While this is a great resource for human responses on word association, it has too many gaps to be suitable for a computational model. Even when we look at all the responses in addition to the 5,000 stimulus words, words occur in the original Codenames board game such as *Amazon*, *Greece* and *horseshoe* do not occur in the database at all. These gaps can only be filled by repeating the experiment with these words as stimulus words. Moreover, this database exists only for English, limiting the applicability of this approach for other languages.

Although it is not suitable as a basis for a computational model, it is useful as a resource on human word associations. The database of word associations could be used to compare how similar the predictions made by a computational model are to human-level associations. While we do not perform this particular comparison in this thesis, it could serve as an interesting evaluation metric for future work.

We find that ontologies and association databases have too many blind spots and consistently fail to encode unorthodox or out-of-the-box relations that would nonetheless be considered valid associations by humans. For example, the word *Frodo* is present in neither WordNet nor the database of University of South Florida Free Association Norms.

2.4 Previous work on computational word association

We are aware of one other paper in the computational word association literature. Obrtlík [2018] explores a word embedding model for Codenames in a setting with two human players playing against each other with the same AI. Their model is based on fastText word embeddings trained on the CWC-2011 which contains more than 2.6 billion words. They use Gensim's most_similar method with several combinations of the player's own words as the positive words and the assassin as negative word. The combinations consist of all possible permutations of the set of words belonging to the player's team. This way they can theoretically generate a hint for up to 9 words. The highest number of target words they achieve is 6, which the model attempted 9 times with an average of 2.2 cards guessed and an accuracy of 60.74%. After using the most_similar method, they select the best possible hint by scoring each hint using a combination of weights for each type of card, the number of co-occurrences and the cosine similarity for the hint and each target word. They achieve a precision of 85.98% and for one word hints, a precision of 71% for two word hints and 66.8% for three word hints. The precision of their model across all hints was 72.9%.

The paper provides a thorough quantitative analysis of the model for Czech. We expand on their work by considering both English and Czech and exploring a wider range of models. We investigate a model based on dependency collocations alongside alterations of the fastText word embeddings model. We also adapt the game to a variant requiring only one human player by substituting the opposing team with a predictable, pre-programmed opponent in order to reduce the complexity of the game.

3. Methods

In this chapter we present several different methods for generating hints and discuss what is needed to generate good hints for the game using these models. We start with an explanation of the lexicon and hint filter in Section 3.1. We then focus on the extraction of similarity scores from word embeddings in Section 3.2 and collocations in Section 3.3. We expand on our collocations method by limiting the extraction of bigrams by words connected through dependency edges in Section 3.4.

The task of a word association model is to provide the set of most related words ordered by similarity for a given word. In the context of Codenames this notion is extended to providing the most (un)related hint given a set of target words (positive) and a set of words that the model must avoid hinting at (negative). First we use the model to calculate similarity scores for each positive word and negative word. Then we weight and aggregate them to arrive at a final score for each possible hint. This process is described in Section 3.5 and several aggregation methods are detailed. We conclude the chapter by providing an example game that shows the different kinds of hints generated by each technique in Section 3.6.

3.1 Game

The board consists of 25 words, chosen randomly each game from a list of approximately 400 words. This word list contains the 400 words used in the original Czech and English board game. We provide these lists in Appendix A.1 and Appendix A.2.

3.1.1 Hint filter

If we were allowed to give the hint *houses* for the word *house*, the problem of generating good hints would become as simple as generating plural forms. For this reason there exists a rule in the original board game to disallow hints that are morphologically related to one of the words on the board.

To adhere to the morphological rules set out for the hint in the original board game and avoid giving hints that violate the spirit of the game (such as plural forms and other morphologically related words), we introduce a hint filter which discards hints for which one or more of the following conditions is true:

- Length of hint is less than 3. (*be* or *no*)
- For every word on the board that has not been selected yet:
  - Hint is part of word (*horse* is part of *horseshoe*)
  - Word is part of hint (*study* is part of *studying*)
  - The relative Levenshtein distance is more than 50%. The relative Levenshtein distance for a word w and a hint h is defined in Equation 3.1.

$$Relative\_Levenshtein(h, w) = \frac{Levensthein(h, w)}{max(|h|, |w|)} \quad (3.1)$$

For example for the word *break* and the hint *broken*, the Levenshtein distance is 3, so the Relative Levenshtein distance will be $3/6 = 50\%$, which does not cross the 50% threshold, so the hint is rejected. Using this metric in combination with a minimum hint length of 3, we are able to detect almost all morphologically related words. The only ones we have seen slip through in practice are *mouse* and the hint *mice*. They go undetected because more than 50% of the letters are different. The only other words for which this is the case that we are aware of, are *louse* and *lice* and *opus* and *opera*. Of course this filter also catches some hints that are not morphologically related and would have been valid otherwise. Losing these hints is acceptable, because the heuristic has such a low error rate and our models still find many hints that do pass the filter.

3.2 Word embeddings

Word embeddings capture semantic similarity, words that have a similar meaning and occur in similar contexts are grouped together. It captures synonymy, which makes it a useful model for word association.

We make use of pre-trained word vectors that were trained on Wikipedia using the fastText method with default parameters as described in Bojanowski et al. [2017]. These embeddings are available in many languages, which makes it easier to build the same model for other languages.[1]

The pre-trained model provides over 2.5 million word embeddings for English and 600,000 for Czech. We cut down on the size of this collection considerably to limit the computation time needed to compare against all of these embeddings when scoring a word. The model is ordered by frequency of the word in the corpus that it was trained on (i.e. frequency of the word in Wikipedia). The first ten thousand words are character sequences that occur often such as *the*, *is* and *talk*. After this we start seeing nonwords like *np_*, words belonging to other languages like *segunda* and other obscure words that are no longer relevant for a general purpose word association model. To avoid clutter, speed up the model and to make sure that we do not include words that people might not know, we limit the number of word embeddings in our model to $10,000$.

To further combat clutter and avoid giving hints that are invalid according to the rules of the game, we filter out embeddings for words that do not adhere to the following conditions:

- The word contains only alphabetical characters.
- The word occurs 50 or more times as a lemma in the CzEng corpus. Lemmas are counted using the second column of the a-layer in CzEng.

We take the top 10,000 word embeddings after filtering, sorted according to the unigram frequency in Wikipedia. Some words that are in the lexicon are

[1]`https://github.com/facebookresearch/fastText/blob/master/docs/pretrained-vectors.md`

not in the top 10,000, for example the word *octopus* occurs at position 17,356, so in reality we select word embeddings for the top 9,600 most frequent words excluding the words in our lexicon and add to that the word embeddings of our 400 lexicon words.

Using this technique we generate one word embeddings file for Czech and one for English, which we use in all the word embedding models that we test.

$$cosine\_similarity(a, b) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.2}$$

For word embeddings we compute similarity scores using the cosine distance between two words a and b, represented by word embeddings A and B. The formula for this is given by Equation 3.2.

3.3 Collocations

A collocation is a group of two or more words that co-occur more often than random chance. Although less strong than the semantic relatedness that word embeddings capture, a collocation still captures a certain association between words. We can see this when we look at the collocation *Eiffel Tower*. When we see *Eiffel*, we quickly think *Tower*.

To find these collocations we need a large amount of text and a measure of association. The text is taken from the CzEng 1.7 corpus.[2] [Bojar et al., 2016] CzEng is a large parallel corpus for Czech and English, containing roughly 57 million sentence pairs and over 600 million words. The corpus bundles a large amount of data, including but not limited to text from subtitles, EU legislation, fiction and web pages.

Each sentence is annotated with large amounts of syntactic and semantic information. The annotation is separated into an analytical layer (a-layer) and a tectogrammatical layer (t-layer). These tectogrammatical trees were created automatically by Treex. `http://ufal.mff.cuni.cz/treex` The a-layer describes the surface-level syntactic tree, including word form, lemma and morphological tag. For the simple bigrams method that considers only words which occur one after the other and the sentence-level bigrams method we use the lemmas from the a-layer. The t-layer is a sparser tree which excludes function words so content words are related through dependency edges directly. This is very useful for our dependency-level bigrams method. The t-layer also includes for each node a t-lemma, a *functor* (semantic role) and a *formeme* (morphosyntactic label).

For the measure of association we use pointwise mutual information (PMI). To reiterate on Section 2.2, we define $p(a, b)$ as the probability that b occurs after a, we calculate this using $p(a, b) = \frac{count(a,b)}{\sum_{x,y \in T} count(x,y)}$, where T is the set of words seen in the text and $count(x, y)$ is the number of times y occurs after x. $p(a)$ is the chance of seeing a word x in the text, we calculate this probability using $p(a) = \frac{count(a)}{\sum_{x \in T} count(x)}$. Equation 3.3 then shows the formula used to calculate PMI scores for two words a and b.

[2] `http://ufal.mff.cuni.cz/czeng/`

$$PMI(a,b) = \log_2 \frac{p(a,b)}{p(a)p(b)} \tag{3.3}$$

For practical purposes we make a slight modification to the definition of $count(x, y)$ and say that $count(x, y)$ is the number of times y occurs **before or after** x. This way the direction of the relation doesn't matter for the association. *Eiffel* will be just as related to *tower* as *Arthur* is to *king*. Even though it might be feasible extract the direction of the relation from the syntactic makeup of the collocate or its syntactic context, this falls outside of the scope of this thesis. We instead choose to generalize and say that the co-occurrence of two words counts equally towards either direction regardless of context.

PMI scores are relative and depend on the corpus that they are based on. A PMI score of 10 based on data extracted from the Czech corpus does not imply the same degree of similarity as a PMI score of 10 in the English corpus. An example that demonstrates this point well is the fact that adding more "junk" data inflates the PMI scores. We take a corpus X and two words x and y, and a corpus X' which is the original corpus X plus a few more sentences that do not contain x or y. In this case the PMI of x and y for corpus X' will be higher than the PMI of x and y for corpus X, even though the counts of x and y remain the same. The reason for this is that the total count of unigrams and bigrams increases, so $p(x)$, $py(y)$ and $p(x, y)$ become smaller and the ratio between $p(x, y)$ and $p(x) * py(y)$ changes. As such, the PMI scores from two different corpora cannot be compared with one another and we will compare the PMI values for Czech and English separately.

3.3.1 Sentence-level bigrams

Experimentally we find that simple bigrams as described above do not form a good model. Finding a good collocate for one word is easily done, but finding good collocates for multiple words is rarely successful because there is too little overlap between collocates. That what is a frequent collocate for one word, is almost never a frequent collocate for another.

To solve this we propose a collocation model over sentence-level bigrams. We define $count(x, y)$ as the number of times y and x occur in the same sentence. This provides a much broader scope for co-occurrences, which increases the chance of overlapping collocates when trying to find a high scoring collocate for multiple words.

The upside of sentence-level collocations is that the model contains more bigrams which means we will discover bigrams we have not seen in the simple bigrams method and for the bigrams we have seen before, we get PMI values that are closer to the real distribution.

We first extract raw unigram and bigram frequencies of lemmas from the CzEng a-layer. During this process we strip away any information other than the raw alphabetic lemma. We consider only the unigrams and bigrams that contain at least one word from the lexicon. We introduce a frequency cutoff into our model, if a unigram has a lower frequency than this cutoff, we exclude it. In our experiments we use a frequency cutoff of 1,000, which means the model does

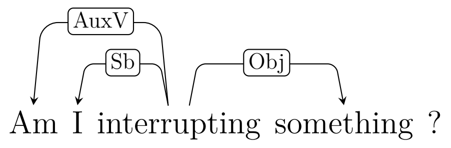


Figure 3.1: Example sentence with dependency annotations from CzEng

not consider hints that occur in less than one thousand times by themselves in the corpus. This translates to a model which takes roughly the top 16,000 most frequent words in CzEng.

The frequency cutoff is an important factor in the quality of the model. Setting the cutoff too high results in a model that is too general and cannot accurately target any particular word on the board. Setting the cutoff too low will result in very obscure words entering the model, which is problematic if these words fall outside the vocabulary of a player. A cutoff that is too low will also suffer from data sparsity. For example if a word occurs only once or twice in the data, it has a high PMI value for the words it co-occurs with, even though the real distribution might be much different. In this case, the PMI value is most likely not representative of the actual distribution.

After filtering we obtain a collocation model on which we can use pointwise mutual information (PMI) to find collocates for the words from our lexicon. We discuss the results of this method in Section 5.2.

3.4 Dependency-level collocations

A dependency is a syntactic relation between two words in a sentence. Each word in a sentence can have only one incoming arc. Each sentence contains a root node, usually the verb from which the dependency edges sprawl over the rest of the words. We show an example a sentence with a dependency analysis from CzEng in Figure 3.1.

For dependency-level bigrams we also consider the sentence level, but we restrict the bigrams to words between which there is a dependency relation. We define $count(x, y)$ as the number of times y and x occur in the same sentence and have a dependency relation.

For the dependency-level collocations we consider words from the t-layer and ignore special t-layer words starting with a hash sign (#). We apply the same filtering as described for the lemmas from the sentence-level bigrams. On top of this, we strip away information about reflexivity of verbs from the lemmas which is encoded with *_se* and *_si*.

The dependency-level collocations are bidirectional, the hint can be both the dependent as well as the head of the dependency relation. Whether the direction of the dependency relation plays a role in quality of the model, would be an interesting direction for future research.

3.4.1 Low frequency words

The sentence-level collocations contain more than 218 million bigrams for Czech and 516 million for English, while the dependency-level collocations contains only

Lexicon word	**Unigram frequency**	**Dependency-level hints**	**Sentence-level hints**
muchomůrka	155	137	690
koloběžka	141	69	963
karbaník	117	93	565
plastelína	104	63	562
vodník	103	57	297
ptakopysk	81	60	327
krápník	75	90	554
lochneska	65	42	319
sněženka	39	24	106
kamion	2	4	12

Table 3.1: List of 10 least frequent Czech lexicon words in CzEng. Unigram frequencies is the unigram count of the lexicon word in CzEng, sentence-level hints is the number of unique hints for this word in the sentence-level collocations model and dependency-level hints is the number of unique hints in the dependency-level collocations model.

18.2 million bigrams for Czech and 34.9 million for English in comparison. Although this might not be significant for words with a large number of bigrams, we expect it to have a large impact on words with low representation in the data.

In Table 3.1 we can see the number of possible hints that the dependency and sentence-level models can give for the least frequent words in CzEng. We can see that the dependency-level collocations have a much smaller hint space to work with than the sentence-level collocations. For example, for the word *lochneska*, there are only 42 unique hints to choose from in the dependency model while the sentence-level collocations model offers over 300 unique possibilities. In general, the sentence-level collocations model offers 5-10 times more options than the dependency-level collocations. When we inspect the number of edges counted for each bigram, the sentence-level model also has 5-10 times more data to work with. While this data sparsity might not have much influence on the words with large frequencies, it is likely that it will be hard for the model to target more than one word at a time if it only has 4 hints to choose from for the word *kamion*. As such, we expect the sentence-level collocations model to perform better than the dependency model for Czech.

In Table 3.2 we can see that the sparsity problem is much less bad for English than it is for Czech. The lowest ranking word *platypus* already has more possible hints in the dependency model (67) than 6 of the low frequency words from the Czech model. All other words have more than 100 possible hints that the model will consider when it occurs in a game, which should be enough to find an adequate hint to target a word on its own. However, we expect that the model will still have problems when looking for hints that are associated with more than one of these words, because it is unlikely that two spaces of 100-200 words will contain much overlap, if any at all.

Lexicon word	Unigram frequency	Dependency-level hints	Sentence-level hints
skyscraper	599	361	1770
snowman	591	229	935
roulette	582	179	1382
leprechaun	567	204	857
horseshoe	548	275	1670
bugle	444	210	1132
kiwi	367	148	1170
aztec	328	130	944
himalayas	322	144	883
platypus	98	67	351

Table 3.2: List of 10 least frequent English lexicon words in CzEng. Unigram frequencies is the unigram count of the lexicon word in CzEng, sentence-level hints is the number of unique hints for this word in the sentence-level collocations model and dependency-level hints is the number of unique hints in the dependency-level collocations model.

3.5 Aggregation

Now that we have a way to obtain the similarity scores for each word on the board in relation to each possible hint, we have to find a way to aggregate these similarity scores into one number which we will call the **aggregate score**. We also sometimes refer to aggregation as *weighting*, because of the weights that are applied to the similarity scores when they are fed to the aggregation method.

Our general strategy will be to split the similarity scores into four groups: own, enemy, neutral and assassin. Each containing the similarity scores for the hint and a word from the player's own cards, the enemy's cards, the neutral cards or the assassin cards respectively. Another categorization we will make is a more simple one. We divide the words on the board into positive and negative words, where the player's own words are the positive words and the negative words the combination of enemy, neutral and assassin words.

The simplest aggregation method is to sum the similarity scores for all the positive words. This works well, because the more related a word is to the hint, the more it contributes to the aggregated score.

The problem with this is the fact that we don't take the negative scores into account at all. For a hint with 3 positive words with a score of 10, there might also be a negative word with a score of 15. This is problematic, because a player will be very likely to choose the negative word over one of the positive words, thus making an incorrect decision and wasting a turn. Or worse yet, losing the game when the negative word is the assassin. This last point reveals an important point in the decision making process: selecting certain types of cards is worse than others. Thus, when choosing a hint, we should also factor the type of card into the equation.

For this purpose we introduce **weights**. These weights consist of four integers, one for player, enemy, neutral and assassin scores. The similarity scores for each category are multiplied by their category-specific weight before they are fed to the aggregation method. For example, if the model is considering the hint *apple*, and there is the positive word *pie* which has a PMI score of 14.051 for *apple*, and a bad word *tasty*, let's say the assassin, with a similarity score of 9.468. Now we apply the weights, say 1 for positive words, and 2 for negative words. The similarity score of the assassin in relation to the hint becomes 18.936 instead of 9.468, and the score for the positive word stays 14.051. The aggregation method can now decide that 18.936 > 14.051 and reject the hint, because the risk that the player will select the assassin is too high.

$$similarity\_score(hint, word) * weight \tag{3.4}$$

Equation 3.4 shows the formula for computing the scores that function as input to the aggregation methods, where *similarity_score* is a function producing similarity scores between two words and *weight* is the degree to which we would like to avoid cards of the category that the word falls into.

In the next sections we show several weighting schemes which can be applied to the relatedness scores of the models (PMI and cosine distance) to find the best hint in a game. The different weighting schemes are different functions for aggregating the similarity scores of the words in a game given a potential hint.

The procedure for finding good hints is straightforward. For all the words in a model we do the following:

- compute similarity scores between the hint and the words on the board,
- multiply each similarity score by the weight of the category that the word falls into,
- aggregate them using an aggregation function.

We use the same weights for all our models, the positive words are multiplied by 1, the negative words by 1.2, the neutral words by 1 and the assassin by 2. We would really like to avoid the assassin, because this ends the game immediately, hurting our recall considerably. We also like to avoid clicking enemy cards because it costs the player a turn. Clicking a neutral card is actually quite okay because it is nearly equivalent to getting a new hint by ending the turn and we also get to eliminate another card from play without penalty.

Different aggregation functions give different priority to these positive and negative scores. We discuss several aggregation functions and their pros and cons. Combined max score is discussed in Section 3.5.1, mean difference in Section 3.5.2, Gensim's most_similar method in Section 3.5.3 and top n in Section 3.5.4.

3.5.1 CombinedMax

$$CombinedMax(P, N) = \sum_{x}^{P} \begin{cases} x & \text{if } x \geq \max(N) \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

Equation 3.5 shows the formula for CombinedMax for arguments P and N where

P is the list of similarity scores for positive words and N is the list of similarity scores for negative words.

To calculate CombinedMax we first determine a threshold by taking the maximum similarity score from the list of negative words. We then sum the scores from the list of positive words that are above the threshold to get the aggregate score. This way a hint only scores high if it relates to many words that are more similar to the hint than the most_similar negative word. This implicit negative threshold is the most distinctive feature of this model.

This method is very sensitive to the weights we apply to the negative words. If we set the weights too high, this method is very good at finding the blind spots in a model. For example, for a collocations model it might find a hint for which there is one positive word with a high PMI score while the rest of the scores are zero. The reason why this happens is that when the weights are high, there are very few positive words that can cross the implicit negative threshold. Therefore the reward for the model to find a hint for which all but one positive word have a PMI value of zero, is very high.

3.5.2 MeanDiff

$$MeanDiff(P, N) = \frac{\sum_x^P x}{|P|} - \frac{\sum_y^N y}{|N|} \tag{3.6}$$

Equation 3.6 shows the formula for mean difference for arguments P and N where P is the list of similarity scores for positive words, N is the list of similarity scores for negative words.

The good part about this method is that it prioritizes . However, if there is a lot of variance with either the class of positive and negative words, mean difference does not account for situations where there is a one negative word that has a really high similarity score with the hint and overshadows the positive words leading the player to click an incorrect card. As such it is not so good at the start of the game when the mean can obscure negative words with high similarity if it is surrounded by many negative words with low similarity to the hint. Near the end of the game this method becomes a lot better, because each peak in similarity of individual words is reflected more strongly in the mean of either class.

3.5.3 Gensim - most_similar

This weighting method is the odd one out, because it does not exactly aggregate the similarity scores of the positive and negative words. Rather, the most_similar method in Gensim works by performing vector arithmetic, adding the embeddings of the positive words to each other and subtracting the negative vectors. The method then returns the words whose vectors are closest to the resulting point.

This method performs well at targeting positive words. However, because it subtracts negative vectors and quite literally tries to ”stay away” from the negative words, it can easily suffer from one simple mistake: including too many negative words. In other words, it assigns too much weight to negative words and starts generating hints that are specifically not referring to negative words, rather than providing hints that refer are similar to the positive words. To resolve this issue we let it take only the assassin word into account for the negative words.

3.5.4 Top-n

The top-n ($n \in 1, 2, 3$) methods are an adaptation of the CombinedMax function. The formula is the same, except for the fact that P is restricted to the n highest values in P. The distributional characteristics of these functions is very interesting, because you have some control over its behaviour by setting n. If we take the Top1 method, we will simply get the hint with the highest similarity score among all pairs of hint and target word. This results in hints with very large similarity scores which are usually highly associative. The Top2 method is generally more mixed, with one hint with a high similarity score and one hint with a moderate similarity score. And if we look at the Top3 method, we often get three words with moderately high similarity scores. Of course there is a lot of variation depending on the number of words still on the board.

The top-n methods are still similar to CombinedMax in the sense that they only have an upperbound n and no lower bound. A top-n is also allowed to give hints for less than n words, as long as $n >= 1$.

3.6 Example game

In this section we provide an example game in order to show the type of hints generated by each technique. We include examples from the sentence-level and dependency-level collocations method with CombinedMax aggregation as well as a word embeddings model with the CombinedMax aggregation method.

`ice`	`jam`	`fish`	`moon`	`america`
`pit`	`change`	`trunk`	`lion`	`shakespeare`
`chick`	`wind`	`marble`	`nail`	`casino`
`cricket`	`field`	`mount`	`snow`	`limousine`
`day`	`racket`	`buck`	`yard`	`witch`

	Dependency-level collocations		**Sentence-level collocations**		**Word embedding**	
Hint	cream		sweeten		frost	
Target 1	ice	15.501	jam	11.693	snow	0.547
Target 2	jam	7.575	snow	10.718	ice	0.376
Target 3	chick	6.942	ice	10.508	wind	0.358

Table 3.3: Board state at the start of turn 1 and hints and target words with their similarity scores for each model used in the example game. The player's cards are marked with blue, the cards of the dummy team are marked with red, the neutral cards with yellow and the assassin with black.

The game starts with the board shown in Table 3.3. In each turn in the simulation, we eliminate the positive word which the AI tried to hint at most for each method, removing one neutral card as if the player had mistakenly selected it and one enemy card to simulate the opposing dummy team taking a turn. Even though the player would play with only one of these models in a real game, we eliminate all of the most hinted cards to avoid situations where a model gives a hint targeting the same word again in the next turn, which would make the example much less informative.

On the first turn the word embedding model gives the hint *frost*, the sentence-level collocations model gives the hint *sweeten* and the dependency-level collocations model gives the hint *cream*. In Table 3.3 we can see that the first target of the dependency-level collocations model is *ice* which has a very high PMI value of 15.501 because *ice cream* is such a strong collocate. The second and third target word *jam* and *chick* are much less related according to the model with PMI values of 7.575 and 6.942.

For the sentence-level collocations model the PMI values calculated by the model are arguably less representative of the strength of the relation between the hint and the target word. The hint *sweeten* is clearly related to *jam*, because *jam* is usually sweet. The model ranks it as most related among the target words with a PMI score of 11.693. However, the second target word *snow* has a score of 10.718 and *ice* has a score of 10.508, while *snow* and *ice* are clearly not as related to *sweeten* as *jam* is, since neither of them are sweet. In fact, one might argue that they are not related to the word *sweeten* at all. *Ice* could be related to *sweeten* through the word *ice cream*, which is most likely the model assigned such a high similarity score to this word pair, because the model looks for words co-occurring in the same sentence. However, people have a much harder time associating words through indirect relations than direct relations, so this is not a good hint. This is one of the flaws inherent to the sentence-level collocations model, it often gives hints that are too indirect.

The word embeddings model tried to hint at the words *snow*, *ice* and *wind* using the hint *frost* with similarity scores of 0.547, 0.376 and 0.358. Although it is debatable whether wind can be frosty, *ice* and *snow* are undoubtedly related to *frost* and likely be easily recognized by the player in the context of a real game.

		`fish`	`moon`	`america`
`pit`	`change`	`trunk`	`lion`	`shakespeare`
`chick`	`wind`	`marble`		`casino`
`cricket`	`field`	`mount`		
`day`	`racket`	`buck`	`yard`	`witch`

	Dependency-level collocations		**Sentence-level collocations**		**Word embedding**	
Hint	tough		ghetto		quartet	
Target 1	racket	10.113	racket	7.784	chick	0.291
Target 2	chick	8.790	shakespeare	7.656	wind	0.280
Target 3	day	6.584	chick	7.098	shakespeare	0.275

Table 3.4: Board state at the start of turn 2 and hints and target words with their similarity scores for each model used in the example game.

The (simulated) player clicks the card *snow* based on the hint *frost*, *jam* based on the hint *sweeten* and *ice* based on the hint *cream*. The player then selects the neutral card *limousine* which ends the turn. The opposing team turns over the card *nail* and passes the game back to the player, resulting in the board state in Table 3.4.

In the following turn the models generate the hints *tough*, *ghetto* and *quartet* shown in Table 3.4. The hint *tough* generated by the dependency model is an interesting hint, because it is an adjective. When an adjective expresses a clear concept such as *sweet* or *green*, they form great hints for multiple target words such as *cake* and *candy* or *apple* and *grass*. However, when the adjectives are more vague, for example the word *tough*, it becomes much more difficult for the player to find good associations. The target words *racket*, *chick* and *day* are only tangentially related to the word *tough* and require individual consideration in order to be picked up by the player as potential targets. Only when putting the collocations together, *tough racket*, *tough chick*, *tough day*, does it become clear that the words might be related.

For the sentence-level hint *ghetto* we also see relations with the target words *racket*, *Shakespeare* and *chick* which are not immediately obvious, which makes guessing them correctly very difficult. While both racket and a chick are likely to be present in a ghetto, these are not the first associations that one would make. Whether *ghetto* is related to *Shakespeare* depends mostly on someone's familiarity with modern interpretations of Shakespeare's work, rather than their command of the English language. This example highlights how difficult it can be to give appropriate hints. Two words which might be related to one person, say someone knowledgeable about Shakespeare, might not be at all obvious to a second person. Similarly, if the hint is a very difficult word that the player does not know, it is hard to make associations. Of course these issues can be sidestepped by providing only the most clear hints which will be obvious to any regular English speaker. Building the models in such a way that they make associations that rely on common knowledge rather than specialized knowledge, is one of the more important parts of this task.

The word embedding model also provides a hint with relatively low similarity scores in regards to the target words. The hint is *quartet* and targets the words *chick*, *wind* and *Shakespeare*. We do not see how any of these words can be related to the hint.

The main reason that these hints are not as good as we would like them to be for any of the models, is because of the aggregation method used. The CombinedMax method selects the hint with the largest sum of PMI scores for the player's own words. A side effect of this is that the model often prefers a hint that targets multiple words with only moderate PMI scores over a hint that targets one word with a high PMI score. This is one of the reasons we introduced the Top1, Top2 and Top3 aggregation methods, which restrict the number of PMI scores that are taken into account to the best 1, 2 or 3 target words.

For turn 3 the words that are eliminated are *racket*, *chick*, *trunk* and *buck*, resulting in the board shown in Table 3.5.

In Table 3.5 we can see that all models target roughly the same set of words in this turn, with *Shakespeare* being the number one target word for all models and *day* being the second target word for the sentence-level collocations and word embedding model and the third word for the dependency-level collocations

		fish	moon	america
pit	change		lion	shakespeare
	wind	marble		casino
cricket	field	mount		
day			yard	witch

	Dependency-level collocations		Sentence-level collocations		Word embedding	
Hint	hooray		hooray		edition	
Target 1	shakespeare	10.732	shakespeare	8.237	shakespeare	0.269
Target 2	america	6.690	day	5.058	day	0.265
Target 3	day	5.122	wind	5.039	america	0.265

Table 3.5: Board state at the start of turn 3 and hints and target words with their similarity scores for each model used in the example game.

model. The two collocation models use the same hint, *hooray*, to target the word *Shakespeare*, probably because the word *hooray* occurs a lot in texts written by Shakespeare and not so much in other texts. Similarly, *hooray* is likely related to *day* through the word *birthday*. This is a very indirect relation, evidenced by the low PMI scores of 5.122 and 5.058 and does not make for a good hint.

The word embedding model takes a different approach and gives the hint *edition*, targeting the words *Shakespeare*, *day* and *America*. The similarity scores are once again low, because of the CombinedMax aggregation method. The word *edition* could be applied to an interpretation of a work from *Shakespeare*, e.g. *a novel edition of Hamlet*, but this association is far-fetched and we doubt that any player would pick up on this in a real game.

			moon	america
	change		lion	
	wind	marble		casino
cricket	field	mount		
day			yard	witch

	Dependency-level collocations		Sentence-level collocations		Word embedding	
Hint	reclaim		oman		annually	
Target 1	america	7.568	america	7.669	day	0.390
Target 2	wind	7.128	wind	5.813	america	0.282
Target 3	day	6.341	day	4.984	wind	0.190

Table 3.6: Board state at the start of turn 4 and hints and target words with their similarity scores for each model used in the example game.

For turn 4 we eliminate the words *Shakespeare*, *pit* and *fish* from the board, resulting in the layout shown in Fable 3.6.

In this turn the word embedding model gives the hint *annually*, the sentence-level collocations model gives the hint *oman* and the dependency-level collocations model gives the hint *reclaim*. In Table 3.6 we can see that the dependency-level collocations model tried to hint at the words *America*, *wind* and *day* with PMI values of 7.568, 7.128 and 6.341 respectively.

The sentence-level model tries to target *America* by using another country as a hint, in this case *Oman*. This is a behaviour we observe often in our models. When a country or capital is present on the board and part of the player's own cards, the model will give another country or capital name as hint.

The word embedding model is the only one that manages to provide a hint with a decent similarity score, *annually*, which has a cosine distance of 0.390 in regards to the target word *day*. Although the inflection of *annually* is unnecessary, the word *annual* is still a good hint for the word *day* since both of them refer to measurements of time.

			moon	
	change			
	wind	marble		casino
cricket		mount		
			yard	witch

	Dependency-level collocations		**Sentence-level collocations**		**Word embedding**	
Hint	gust		gust		turbine	
Target 1	wind	15.502	wind	13.142	wind	0.540

Table 3.7: Board state at the start of turn 5 and hints and target words with their similarity scores for each model used in the example game.

For the fifth and last turn we eliminate the words *day*, *America*, *lion* and *field* from the board, resulting in the layout shown in Fable 3.7.

On the last turn we can see a good example of what happens when our models generate hints for only one word. They will generate a hint with the largest similarity score to the target word that they can find. In the case of the word *wind*, the highest cosine distance is achieved by the word *turbine* in our word embedding model as seen in Table 3.7. Surprisingly the context of *wind* and *turbine* are so similar that *turbine* has the highest cosine similarity with *wind*. The collocate *wind turbine* because the collocation models both choose another word, *gust*, with PMI values of 15.502 and 13.142 for the dependency-level and sentence-level bigram methods respectively. When looking at the PMI values of *wind* and *turbine* (13.842 for dependency-level and 11.777 for sentence-level) we can see that the collocation models did consider the option, but judged *gust* to be more related. The phrase *a gust of wind* is most likely a stronger collocate than *wind turbine*. *Gust* occurs only in the context of *gust of wind* while turbines are not only wind-related.

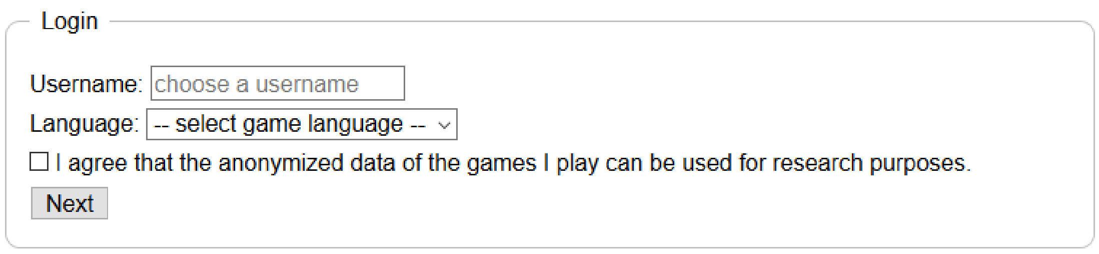
4. Implementation

The code for the application is available on GitHub.[1] It includes code for running the web application, generating models and the anonymized data from our experiments.

We have built a web application implementing a singleplayer version of the game Codenames, where we use several NLP models to generate hints for the player. We use a combination of vanilla HTML, CSS and Javascript for the front-end. The back-end consists of a PHP endpoint and a standalone Python process for each model to produce hints. We use Python 3.6 and the word embedding models are queried using Gensim [Řehůřek and Sojka, 2010].[2]

One of the benefits of implementing the game as a web application is that it is easily accessible to users, participants can play the game from the comfort of their own home, which makes it easier to gather data. A major focus while designing the application was to increase the number of games played, so we can collect more data. One of the strategies is to make the game as engaging as possible, which we achieve by adapting the ruleset of the original game slightly as detailed in Section 1.2.1. Additionally, users cannot start new games before finishing old ones. This way users are discouraged from leaving games when things do not go their way, which might leave us with many incomplete games and skew our data.

4.1 Login


Login

Username: choose a username

Language: -- select game language --

☐ I agree that the anonymized data of the games I play can be used for research purposes.

Next

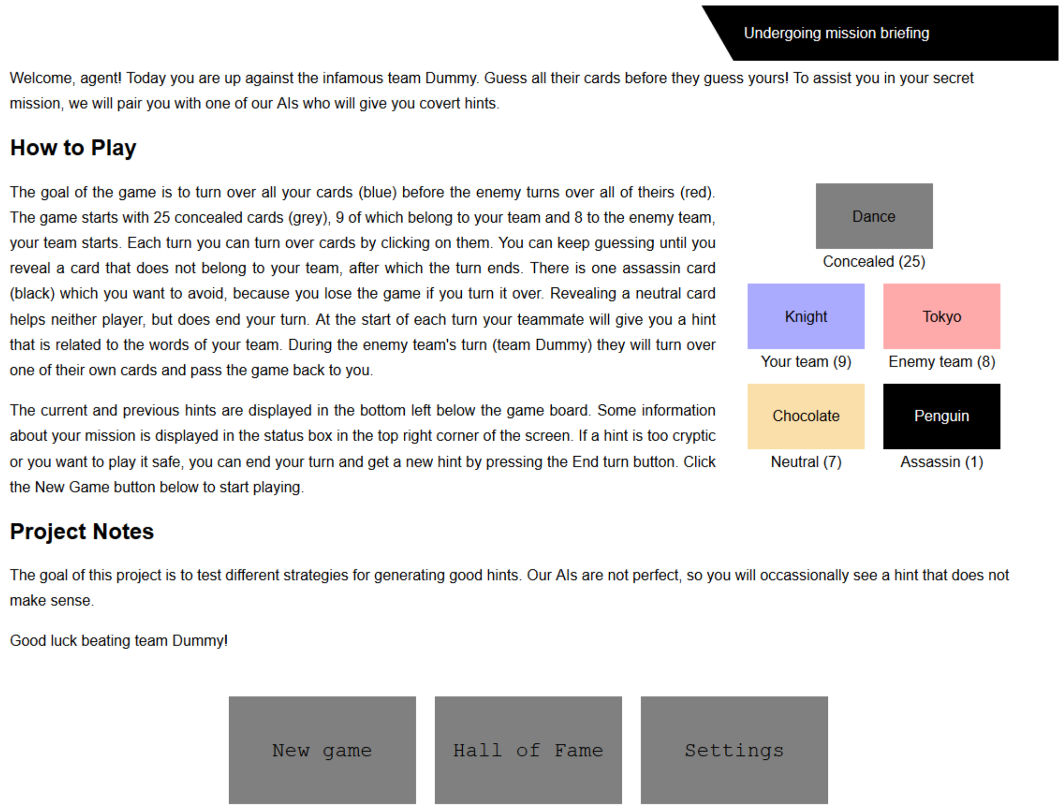
Figure 4.1: Login screen

The login screen asks the player for a username, their language setting and a checkbox to make sure we can use the data we collect. The username is a nickname used for displaying a user's scores in the Hall of Fame and is tied to a unique identifier which is used to identify a user internally in the application. If a user logs in with the same nickname, the games they play will be tied to the same internal identifier. While this method of authentication proved adequate for our small group of participants, it is not suitable for larger applications because it allows users to login as other players. The effects of this type of abuse are limited, because the only actions someone can take when they login to someone else's account, would be to play a game. For the affected player this can only

[1]https://github.com/mderijk/codenames
[2]https://radimrehurek.com/gensim/

have positive consequences: if the game goes well, they get another high score in the Hall of Fame. If the game goes poorly, the score is unlikely to show up in the top 10 provided by the Hall of Fame at all. For data collection and research this is more of an issue, because it makes it harder to analyze the performance of individual users. We do not perform this type of analysis in this thesis, therefore this has no effect on our results. For future work however, it is recommended to change the method of authentication. To protect players' identities the published results on GitHub are anonymized: information about usernames is excluded from the data.

4.2 Menu


Undergoing mission briefing

Welcome, agent! Today you are up against the infamous team Dummy. Guess all their cards before they guess yours! To assist you in your secret mission, we will pair you with one of our AIs who will give you covert hints.

How to Play

The goal of the game is to turn over all your cards (blue) before the enemy turns over all of theirs (red). The game starts with 25 concealed cards (grey), 9 of which belong to your team and 8 to the enemy team, your team starts. Each turn you can turn over cards by clicking on them. You can keep guessing until you reveal a card that does not belong to your team, after which the turn ends. There is one assassin card (black) which you want to avoid, because you lose the game if you turn it over. Revealing a neutral card helps neither player, but does end your turn. At the start of each turn your teammate will give you a hint that is related to the words of your team. During the enemy team's turn (team Dummy) they will turn over one of their own cards and pass the game back to you.

The current and previous hints are displayed in the bottom left below the game board. Some information about your mission is displayed in the status box in the top right corner of the screen. If a hint is too cryptic or you want to play it safe, you can end your turn and get a new hint by pressing the End turn button. Click the New Game button below to start playing.

Project Notes

The goal of this project is to test different strategies for generating good hints. Our AIs are not perfect, so you will occassionally see a hint that does not make sense.

Good luck beating team Dummy!

Figure 4.2: Menu screen

After logging in, the menu screen is shown. This screen provides an explanation of the game, showing the different card types on the right. The player can also start a new game (or resume one if they are in the middle of a game), view the Hall of Fame and change their settings. The settings page shows the username of the logged in user and provides the option to change the language of the game.

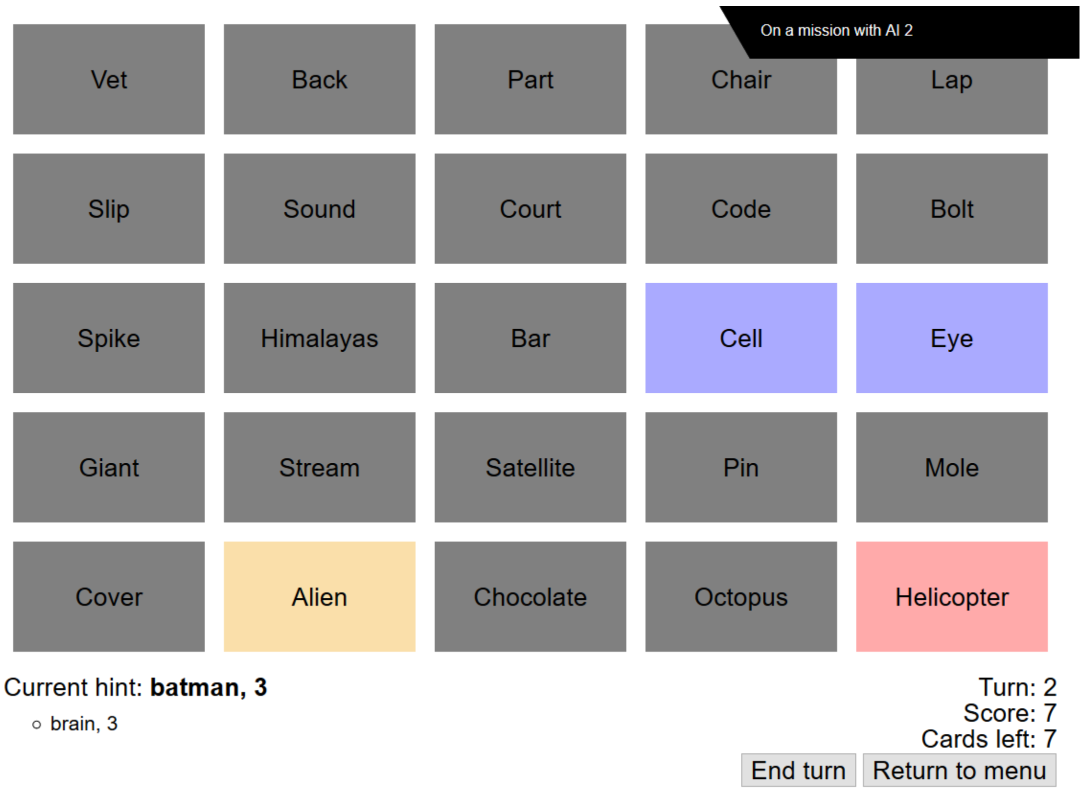
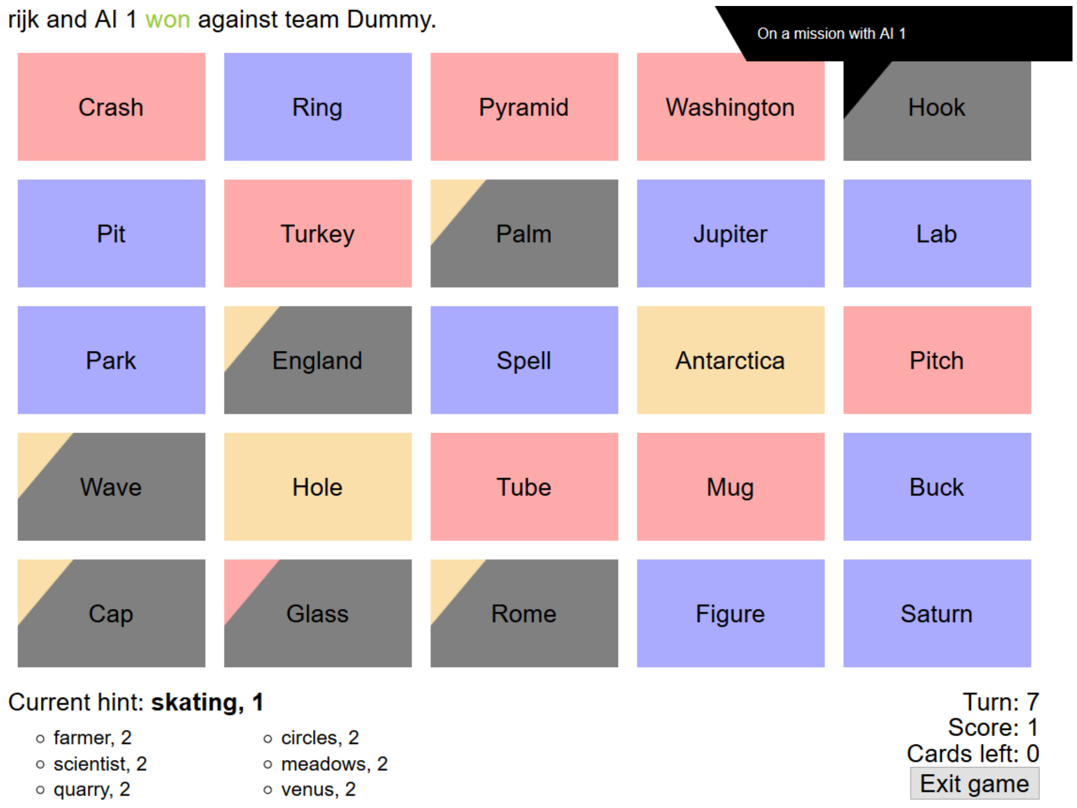


Figure 4.3: Game screen

4.3 Game

Figure 4.3 shows an example game. We use blue for the players own cards, red for the enemy team's cards, yellow for the neutral cards and black to indicate the assassin. Gray cards have not been selected by the player yet and can be of any type. The status bar on the top right shows the name of the AI that generates the hints. On the bottom left the player can see the current hint as well as a history of the previous hints provided by the AI. On the bottom right we show the current turn, the score that the player would achieve if they guessed all of their cards in this turn and the number of own cards that the player has left. The end turn allows the player to end their turn without selecting an incorrect card and the return to menu button allows the user to navigate back to the menu, in case they want to take another look at the explanation of the game or the Hall of Fame.

In Figure 4.4 we see what the game looks like when it has ended. The game players are informed of their win or loss. They can exit the game by clicking the exit game button. The type of the cards that the player did not click is revealed at the end of the game and shown by a coloured strip in the upper left corner of the card. The mapping between colours and types is the same as for the clicked cards.


rijk and AI 1 won against team Dummy.

On a mission with AI 1

Crash	Ring	Pyramid	Washington	Hook
Pit	Turkey	Palm	Jupiter	Lab
Park	England	Spell	Antarctica	Pitch
Wave	Hole	Tube	Mug	Buck
Cap	Glass	Rome	Figure	Saturn

Current hint: **skating, 1**

- farmer, 2
- scientist, 2
- quarry, 2
- circles, 2
- meadows, 2
- venus, 2

Turn: 7
Score: 1
Cards left: 0

Figure 4.4: Game end screen

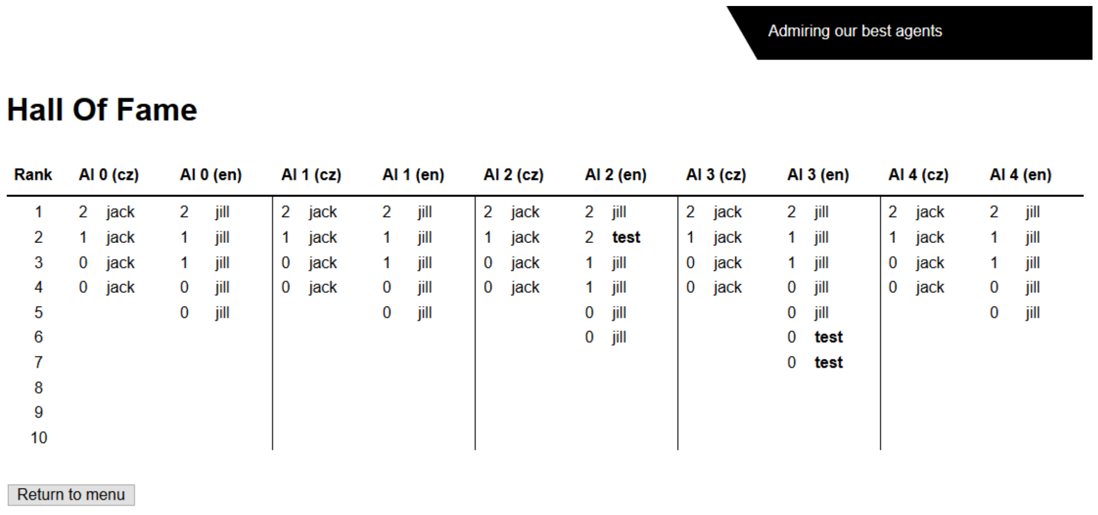
4.3.1 AI names

The player is informed which AI it is playing with during the game. Each AI is indicated with a unique number. We do not name the AI using real names, words, or descriptions such as ”word embeddings” or ”collocations” to avoid influencing how players make decisions. We want to avoid players associating the AI with things they know, because they might change their associations based on this.

Players who played the game frequently report that they were able to recognize the AIs based on the hints that they give. This effect is expected, because something similar happens when someone plays the game in real life with the same people multiple times. One gradually learns what kind of hints other people give. The same happened with our AI and some of the players that played the game. It is important to note that this might affect the results somewhat.

4.4 Hall of Fame

The Hall of Fame provides a simple overview of the top 10 high scores among all players for each AI, ordered by score and then alphabetically by username. Figure 4.5 shows what the Hall of Fame might look like. The username of the logged in user is displayed in bold, in this case **test**.



Hall Of Fame

Rank	AI 0 (cz)	AI 0 (en)	AI 1 (cz)	AI 1 (en)	AI 2 (cz)	AI 2 (en)	AI 3 (cz)	AI 3 (en)	AI 4 (cz)	AI 4 (en)
1	2 jack	2 jill	2 jack	2 jill	2 jack	2 jill	2 jack	2 jill	2 jack	2 jill
2	1 jack	1 jill	1 jack	1 jill	1 jack	2 **test**	1 jack	1 jill	1 jack	1 jill
3	0 jack	1 jill	0 jack	1 jill	0 jack	1 jill	0 jack	1 jill	0 jack	1 jill
4	0 jack	0 jill	0 jack	0 jill	0 jack	1 jill	0 jack	0 jill	0 jack	0 jill
5		0 jill		0 jill		0 jill		0 jill		0 jill
6						0 jill		0 **test**		
7								0 **test**		
8										
9										
10										

Figure 4.5: Hall of Fame screen

4.4.1 Lexicon

The original lexicons are the 400 words used in the original Czech and English board game. We provide these lists in Appendix A.1 and Appendix A.2.

For our implementation we make two important exceptions to ensure compatibility with the word embeddings model. For English we removed 4 entries containing spaces: *ice cream*, *Loch Ness*, *New York* and *scuba diver*. The rationale behind this is that it is difficult to compute similarity scores for multi-word expressions. For example, our pre-trained word embeddings only contain word vectors for single words not for multi-word expressions.

For Czech we had to remove one word (špageta), because it did not occur in our word embeddings model at all. The reason for this is that people rarely speak of a single slice of spaghetti (špageta), so it likely never occurred in the data that the fastText models were trained on.

4.5 Collocations

On the implementation level we perform several simplifications to speed up the generation of the collocations model. $count(a)$ is computed as normal, for each word a that appears in the corpus we count all unigram occurrences. For the bigram count $count(a, b)$, we restrict the input by only considering pairs a, b such that a appears in the lexicon, which reduces the space complexity of the bigrams from $O(n^2)$ to $O(c \cdot n)$. We also check for $a \neq b$, because the rules of the game do not allow us to use the word itself as a hint.

However, during this optimization we also mistakenly changed the way we compute $p(a,b) = \frac{count(a,b)}{\sum_{x,y \in T} count(x,y)}$ to $p(a,b) = \frac{count(a,b)}{\sum_{x \in L, y \in T} count(x,y)}$, where T is the set of words seen in the text and L is the set of words found in the lexicon. Now $\sum_{x \in L, y \in T} count(x,y)$ is a much lower number than $\sum_{x,y \in T} count(x,y)$.

This value depends on the size of T and L, the text from our corpus and the lexicon, which are both fixed. As such this part of the formula is a constant and

	Constant	
Collocate	18 million	182 million
extinguisher	14.382	11.998
peat	13.614	11.230
brigade	12.216	9.832
phaser	11.702	9.318
stoke	11.697	9.314
extinguish	11.515	9.131
picket	11.388	9.005
crackle	11.337	8.953
frost	11.273	8.890
cannon	11.172	8.788

Table 4.1: Hints and their PMI values for the word *fire* generated using two dependency-level collocation models with different constants in the denominator of $p(a, b)$.

the actual formula we are looking at is:

$$PMI(a, b) = \log_2 \frac{count(a, b)}{p(a)p(b)} \tag{4.1}$$

Eliminating this multiplicative constant affects the PMI values produced by the function. Because of the properties of log functions, this multiplicative constant translates to an additive constant. As such all PMI values are shifted by some constant, which means the relative ordering of two words a, b by their PMI values does not change.

In Table 4.1 we show an example of collocates found for the word *fire* using the dependency-level collocations model and an adapted version that uses a constant that is 10 times as large as the denominator of $p(a, b)$ for the model we built. We can see that the ordering of the words is preserved and even the absolute difference between the PMI scores is the same.

If our model would generate candidates for only one target word, this would have had no effect. However, when we sum the similarity scores of two words in our aggregation method and compare them against a candidate hint that sums three words, it does have an effect. For example, consider a hint *jazz* which has two target words with PMI values 4 and 5 and a hint *caramel* with three target words with PMI values of 1, 2 and 3. In the default case *jazz* would receive an aggregate score of 9 and beat *caramel* which would have an aggregate score of 6. Now, if we increase all PMI values by 4 we get $8 + 9 = 17$ for *jazz* which is smaller than $5 + 6 + 7 = 18$. And suddenly *caramel* has a higher score than *jazz*.

Because the additive constant is positive, our model is biased towards hints targeting more than one word, because it includes the positive constant more often in the sum of the PMI values than a hint that target fewer words. This could have influenced the performance of the Top2 and Top3 dependency models that we see in Section 5.3.

5. Results

In this chapter we perform both quantitative and qualitative analysis of our models. We used an iterative approach in the design of our methods, so we will dedicate one section to each iteration of models. First we establish a baseline for our models in Section 5.1 by Monte Carlo simulation. Then we analyze the results of the first test run in Section 5.2 and discuss the improvements we made to our models in Section 5.3. Next, we analyze the PMI scores using both manual annotation as well as empirical data in Section 5.4 and derive several thresholds for PMI and cosine similarity. The results of the models built using these thresholds are covered in Section 5.5.1. Finally, we discuss the performance of a combination of a word embedding and collocation model in Section 5.5.2. A comprehensive overview of all results can be found in the conclusion in Table 6.1.

5.1 Baseline

In this section we establish the baseline precision, recall and f-score for the singleplayer variant of Codenames.

The true positives are the cards clicked by the player that were their own, false positives are the cards that the player clicked which were not their own, and the false negatives are the player's cards that they did not click at the end of the game.

We also provide average win rate and chance to die by assassin for completeness. Although these statistics are useful, they also demand much more data in order to arrive at a meaningful approximation. As such they are less suitable for our purposes since gathering this data would require a lot of time. We therefore do not consider these metrics for the rest of our models. By using precision, recall and f-score on the decision level instead of win and loss rate at the game level, we are able to provide more accurate metrics and evaluate models using much fewer games.

The baseline is set by a scenario in which hints do not provide any help to the player whatsoever, which is equivalent to the situation where there are no hints at all and cards are chosen randomly by the player. The end turn button that is present in the game is not modeled as a possible action, because a player clicking cards randomly does not gain any additional information from getting a new hint, while the opposing team does have the opportunity of turning over an additional card.

We perform a Monte Carlo simulation of playing the game by repeatedly selecting cards at random. We simulate 10 million games in this way, from which we obtain the results displayed in Table 5.1.

	Win rate	**Assassin loss**	**Precision**	**Recall**	**F-score**
Baseline	0.39%	45.43%	0.389	0.339	0.362

Table 5.1: Average win rate, chance to lose by clicking the assassin, precision, recall and f-score achieved by randomly clicking cards until the game ends.

The baseline for the win rate, the chance to win the game by selecting cards at random, is 0.39% as seen in Table 5.1. This is a very low number, on average this means the player wins only one game out of more than 250 games. The baseline precision, recall and f-score are 0.389, 0.39 and 0.362 respectively.

If the generated hints provide any semantic meaning related to the player's words more so than to the other team's words, then we would expect the average win rate to be higher than the baseline. The same can be said for precision, recall and f-score.

Even though the evaluation of our model is task-specific and limited to a subset of only 400 words (the lexicon) for each language, we hypothesize that the results generalize well to other word association tasks. The word embeddings capture semantic similarity for the whole language and the collocation counts can be generated for all words in a corpus instead of limiting them to the words in the lexicon.

5.1.1 Distribution of decisions

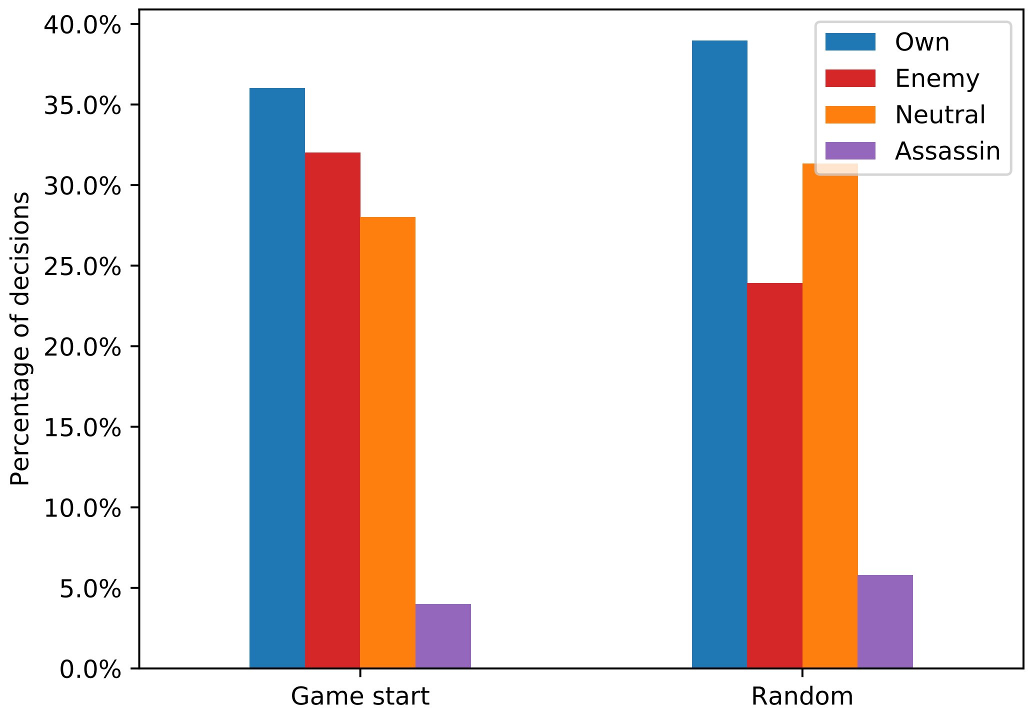


Figure 5.1: Percentage of own, enemy, neutral and assassin cards at the start of the game versus the distribution of cards clicked by the player during the game when selecting cards randomly.

The number of the cards at the start of the game is 9, 8, 7 and 1 for own, enemy, neutral and assassin cards respectively. However, because the opposing team selects one of their own cards in their turn, the chance for the player to select one of the enemy's cards is lower than 32% in practice. We therefore also establish a baseline for the percentage of own, enemy, neutral and assassin cards clicked by the player through Monte Carlo simulation, using the same methodology described in Section 5.1. The results of this simulation can be found in

Figure 5.1, contrasted with the distribution of the card types at the start of the game.

We see that the chance that the player clicks on one of their own cards is 38.95% on average. The average chance to click on an enemy card is much lower with 23.92%, because the opposing team reduces the number of enemy cards in each of their turns. The chance to click on a neutral or an assassin card are 31.33% and 5.80% respectively. In conclusion, the chance for the player to select one of the enemy team's cards is significantly lower, because of the actions of the dummy team and the chance for the player to select one of their own cards, a neutral card or an assassin card is slightly higher. We will use the distribution obtained by random selection of cards as a baseline for the analysis of player decisions in the rest of this chapter.

5.2 Initial models

The results for our initial models are shown in Table 5.2. All of our models perform above the baseline, which means they are better than random chance.

Setup		**Player decisions**					
		CZ			**EN**		
	Aggregation	P	R	F_1	P	R	F_1
Random							
	Baseline	0.389	0.339	0.362	0.389	0.339	0.362
Sentence-level collocations							
	CombinedMax	0.507	0.490	0.498	0.500	0.466	0.482
Dependency-level col.							
	CombinedMax	**0.629**	0.654	0.641	0.547	0.497	0.521
	MeanDiff	0.575	0.636	0.604	0.546	0.544	0.545
Word embeddings							
	most_similar	0.616	**0.753**	**0.677**	0.563	**0.607**	0.584
	CombinedMax	0.558	0.578	0.568	**0.567**	0.606	**0.586**

Table 5.2: Micro-averaged precision, recall and f-score for our initial models, including baseline. Highest precision, recall and f-score are marked in bold separately for each combination of language and setup with more than one aggregation method.

5.2.1 Word embeddings

We observe that the word embeddings with the most_similar method are significantly better for Czech than for English in all aspects. They are more accurate with a precision of 0.616 versus 0.563 and players manage to select a much larger portion of their own cards in a game with a recall of 0.753 and 0.607 respectively. This in turn leads to Czech having a much higher f-score (0.677) for this method than English (0.584).

We do not observe this effect for the CombinedMax aggregation method. Rather, for CombinedMax the results are slightly higher for English than for Czech with a precision of 0.567 and 0.558, recall of 0.606 and 0.578 and an f-score of 0.586 and 0.568 respectively. The word embedding models are the best across the board, with the most_similar method achieving an f-score of 0.677 for Czech and the CombinedMax method achieving an f-score of 0.586.

5.2.2 Collocations

We observe a clear language effect in the results with Czech having f-scores of 0.641 and 0.604 for CombinedMax and MeanDiff respectively, contrasted with English which has f-scores of 0.521 and 0.545 for these methods. While it was unclear for the word embeddings why one method did not have this effect, we can see here that the effect also favours Czech and in this case it does so for both of the methods. Therefore we hypothesize that this is caused by the fact that our group of English speaking players consists mostly of second language learners, while most of the players for Czech were native speakers.

The dependency-level collocations outperform the sentence-level collocations. The sentence-level bigram method performs worse than the dependency-level collocations for both languages with f-scores of 0.500 and 0.482. Even though we expected that the lack of data for the dependency model might hurt its performance, it seems that the constraints on the bigrams lead to more accurate results.

We continue testing with collocations from dependency relations, because they give the most promising results. Although the model with sentence-level collocations contains many more bigrams, dependencies seem to capture more accurate relations between words thus producing better hints.

5.2.3 Distribution of decisions

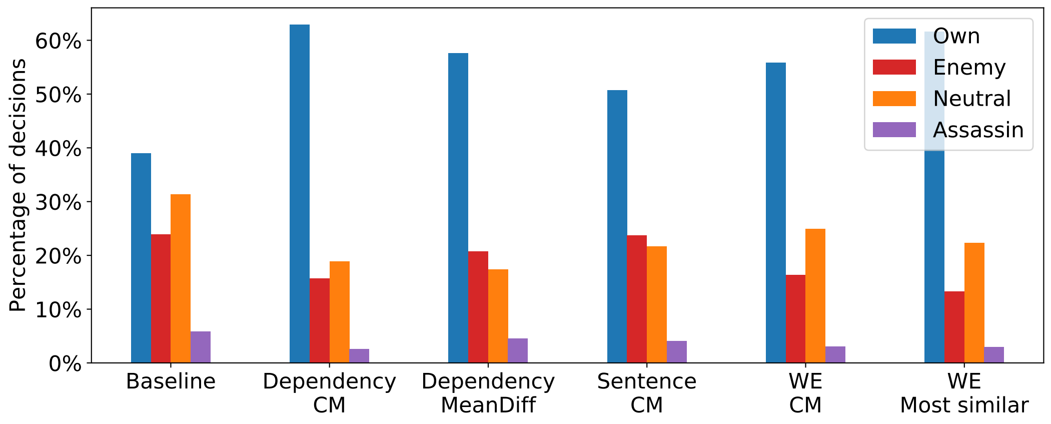


Figure 5.2: Percentage of own, enemy, neutral and assassin cards clicked by players across all games played with the dependency-level collocations model with CombinedMax and MeanDifference aggregation, the sentence-level collocations model with CombinedMax and the word embeddings model with CombinedMax and most_similar aggregation for Czech

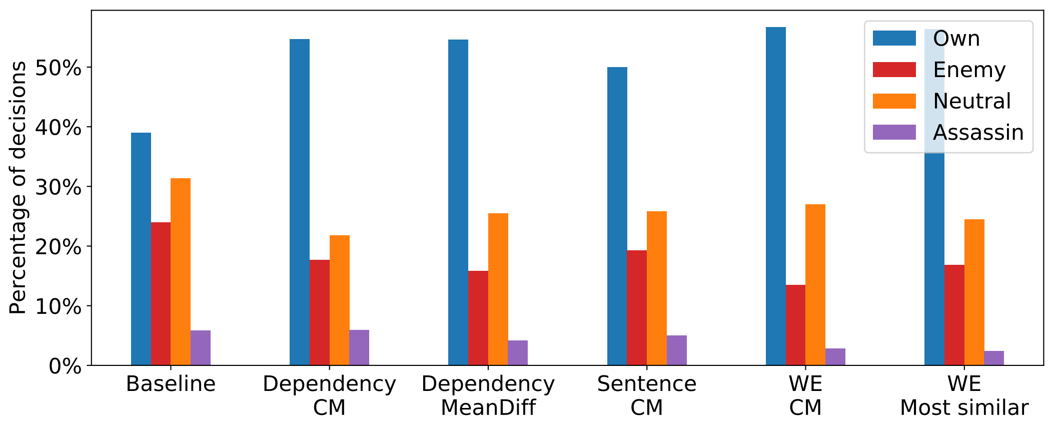


Figure 5.3: Percentage of own, enemy, neutral and assassin cards clicked by players across all games played with the dependency-level collocations model with CombinedMax and MeanDifference aggregation, the sentence-level collocations model with CombinedMax and the word embeddings model with CombinedMax and most similar_aggregation for English

In Figure 5.2 and Figure 5.3 we can see that the distribution of decisions taken is similar to the baseline distribution for almost all models. Only the distributions for the Czech dependency model with MeanDiff aggregation and the Czech sentence-level collocations model are markedly different. The neutral are clicked less frequent than the enemy cards, which is suprising, especially given the weights of 1.2 for enemy and 1 for neutral cards that we use for each model. One would expect the enemy cards to be clicked less frequently and the neutral cards more often. These two models show distributions similar to the one observed at the start of the game, which we saw in Figure 5.1. We hypothesize that this is due to the quality of the models, because sentence-level collocations perform quite bad in comparison to for example word embeddings, which do in fact show a lower percentage of enemy cards clicked than neutral cards when compared to the baseline.

When we graph the distribution of decisions for our other methods, we observe the same relative ordering between the percentage of own, enemy, neutral and assassin cards clicked by the player. The only useful information we can gather from these graphs is that a method performs better if the percentage of own cards selected is higher. We can observe the exact same information from a model's precision. Because the distribution between selected enemy, neutral and assassin cards is not significantly different from the baseline for any of the methods other than the two mentioned above, we do not discuss the distribution of decisions for any of the other models. The remaining graphs can be found in appendix A.4.

5.3 Improved models

In this section we improve our dependency and word embedding models by introducing new aggregation methods. We start with the Top1 method, which always tries to find a hint for only one word. It is not possible to win the game this way through association alone, because the maximum number of hints a player can get is 8, by ending their turn 7 times in a game. Even though this is not as fun for our participants, it provides a useful baseline. We also test a Top2 and a Top3 model, each trying to give hints for 2 and 3 words respectively.

At this point we introduce a number that shows to how many target words the hint relates. We show this number to the player together with the hint in all models other than the initial models we saw in Section 5.2. At first glance the computation of these numbers appears easy for our Top1, 2 and 3 models. However, for models that try to give hints for multiple words it might be the case that they do not manage to find a hint for the intended amount of words. In such cases the number provided by the model will reflect the actual number of words that it has managed to target with the given hint.

During testing we notice a major effect of knowing the number of words that the AI is hinting at. The player now knows when they have exhausted a hint and can stop using it. If the hint was only for one word and the player has selected this card, they will now press the end turn button to gain a new hint whereas previously they might have continued guessing using the same hint. In addition to this, we consider a Top3 model that gives the hint *glass, 2*. When showing a lower number of cards than expected, an observant player might deduce that the third word that they consider most associated to the hint is most likely a bad card which they should not click, even in future turns. As we can see, the information that the model has not managed to find a hint for the target amount of words can be very valuable to the player.

Setup		**Player decisions**					
		CZ			**EN**		
	Aggregation	P	R	F_1	P	R	F_1
Random							
	Baseline	0.389	0.339	0.362	0.389	0.339	0.362
Sentence-level collocations							
	CombinedMax	0.507	0.490	0.498	0.500	0.466	0.482
Dependency-level col.							
	CombinedMax	0.629	0.654	0.641	0.547	0.497	0.521
	MeanDiff	0.575	0.636	0.604	0.546	0.544	0.545
	Top1	**0.722**	0.633	0.675	**0.693**	0.678	**0.685**
	Top2	0.621	**0.778**	**0.691**	0.646	**0.711**	0.677
	Top3	0.552	0.644	0.595	0.655	0.611	0.632
Word embeddings							
	most_similar	0.616	0.753	0.677	0.563	0.607	0.584
	CombinedMax	0.558	0.578	0.568	0.567	0.606	0.586
	Top1	**0.789**	**0.789**	**0.789**	**0.768**	0.735	**0.751**
	Top2	0.608	0.690	0.647	0.614	0.735	0.669
	Top3	0.574	0.626	0.599	0.667	**0.786**	0.722

Table 5.3: Micro-averaged precision, recall and f-score for our improved models. Highest precision, recall and f-score are marked in bold separately for each combination of language and setup with more than one aggregation method.

We can see that the Top1 and 2 models provide a significant improvement over the previous dependency models with f-scores of 0.675 and 0.691 for Czech versus the previous f-score of 0.641 for CombinedMax and 0.604 for MeanDiff and f-scores of 0.685 and 0.677 versus the previous f-scores of 0.521 and 0.545 for English. For English the Top3 model performs much better than the previous model with an f-score of 0.632, but for Czech it does not. The Top2 dependency model achieves a noteworthy recall of 0.778 for Czech and 0.711 for English compared to the other dependency models. In this case high recall means that players on average get much closer to turning over all of their cards and winning the game.

The Top1 models achieve the highest precision across the board. This is not surprising, it is easy to give one good hint for one word, but much harder to give a good hint for two or more words and still have the player guess both of them.

For the word embedding models we see that the Top1 model performed best for both Czech and English with f-scores of 0.789 and 0.751 respectively. The Czech Top3 model performs poorly similar to the dependency models. However, the English Top3 model performs very well in comparison with an f-score of 0.722 and a recall of 0.786. The Top2 word embedding models are considerably worse than their Top1 counterpart, contrary to what we see for the dependency models. We observe across all models that the Top1 model has higher precision than recall and for the Top2 and Top3 models this relation swaps and the recall is higher than the precision. The only anomaly is the English Top3 dependency model which has a precision of 0.655 and a recall of 0.611. Curiously, its precision is much higher than for the Czech model (0.552).

5.4 Analysis of similarity scores

5.4.1 PMI experiment

To get a better idea of the way PMI values behave, we perform a small experiment using the simple bigram model described in Section 3.3. We generate a list of 100 examples with a uniform distribution of PMI values between 0 and 16. This list of bigrams is then shuffled and presented to two annotators without similarity scores. For each bigram the annotators mark the word pair as related or unrelated. We then analyze the results by finding the best cutoff point for both datasets. We perform the experiment for both English and Czech. Both annotators are Czech native speakers and speak English as a second language.

To select the bigrams we take one word from the word list used for the original board game[1] and a hint which can be any word in the model and generate PMI scores for these combinations. We then choose 100 examples by selecting the bigram which has a PMI value closest to 100 evenly spread points in the interval of 0 and 16.

5.4.2 Annotator results

The annotators agree in 62 and 55 percent of the cases (for Czech and English respectively), that two words are not related. But, out of the 38 and 45 cases that either one of them says that the two words are related, only in 17 and 21

[1]For Czech there is one exception, the word *velikonoce* is used but not part of the original word list

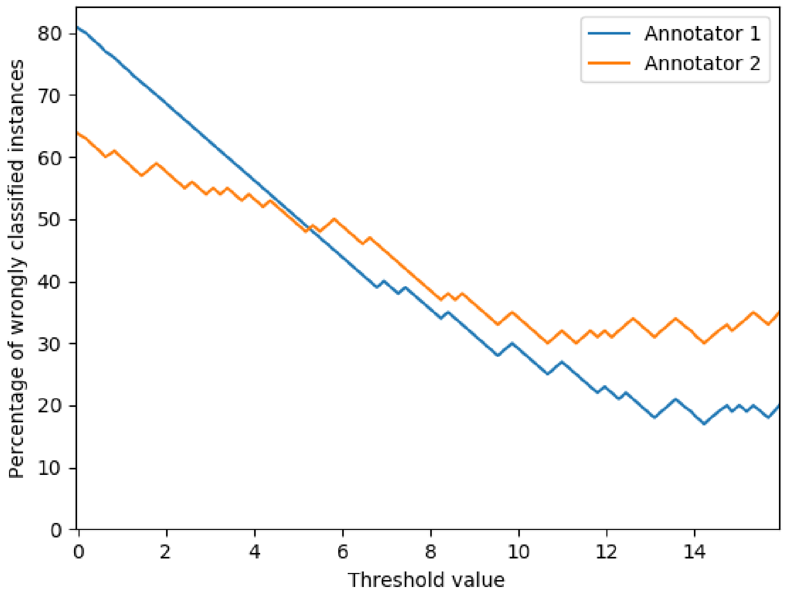


Figure 5.4: Number of incorrectly classified instances for each possible threshold value, given the judgments made by annotator 1 and 2 for 100 Czech bigrams

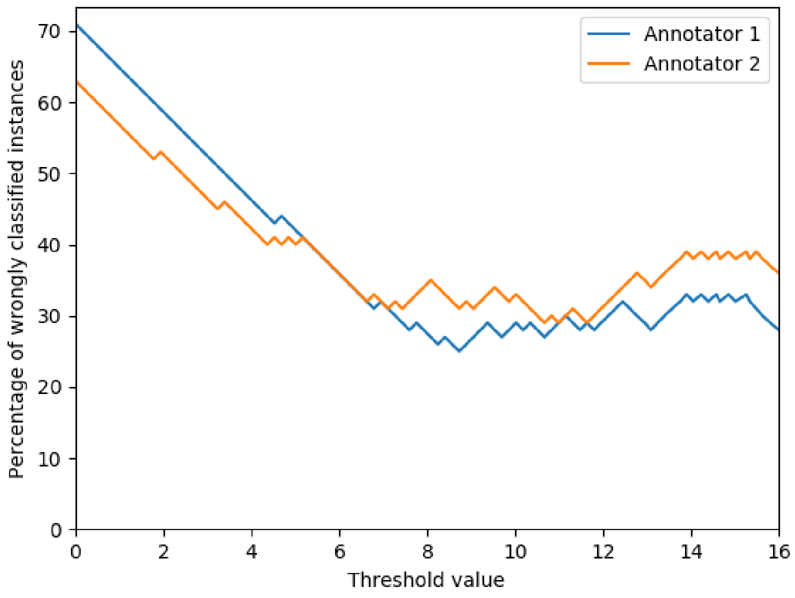


Figure 5.5: Number of incorrectly classified instances for each possible threshold value, given the judgments made by annotator 1 and 2 for 100 English bigrams

	Czech		**English**	
Annotator	Best threshold	Mistakes	Best threshold	Mistakes
Annotator 1	14.221	17%	8.727	25%
Annotator 2	10.667	30%	10.668	29%

Table 5.4: Thresholds with lowest number of wrongly classified bigrams using the annotators classification as gold standard.

	Annotator 1	
Annotator 2	Not related	Related
Not related	62	2
Related	19	17

Table 5.5: Confusion matrix showing the annotator (dis)agreements for Czech.

	Annotator 1	
Annotator 2	Not related	Related
Not related	55	8
Related	16	21

Table 5.6: Confusion matrix showing the annotator (dis)agreements for English.

cases do both of them say that the words are related. What's more, in the case of Czech, one of the annotators said for 19 cases that it was related where the other annotator said that they were not related, while the reverse is true for only 2 cases. We explain this phenomenon using the idea of a mental threshold: a person considers two words to be related if their similarity is above a certain threshold and as not related if the items are not similar enough. We can explain the fact that one of the annotators considered almost twice as many words to be related as the other annotator, by assuming that they put the threshold for relatedness much higher. The fact that the two annotators disagree much more for English (16 and 8) can be explained by the fact that both annotators are not native speakers of English.

Another thing we learnt from this experiment is that the simple bigram model is very noisy. When we inspect the data used for the experiment, we sometimes see completely unrelated words like *cotton* and *single* that receive very high PMI scores. This is reflected in Figures 5.4 and 5.5 where we plot the number of incorrectly classified instances for each possible threshold value. We can see that for English many mistakes still remain, regardless of where we put the threshold for both annotators, while for Czech only one of the annotators approaches a reasonable number of 17 mistakes. Until we take a look at Table 5.5 where we see that annotator 1 made only 19 related judgments for Czech, which means that the largest achievable number of mistakes by placing the threshold anywhere on the scale, is 19. For English we see something similar, with 25 mistakes out of 29 words marked as related. Annotator 2 performs only markedly better with 30 mistakes out of 36 relations for Czech and 29 mistakes out of 37 related words for English.

The reason for this is not that our annotators are bad at associating words, rather our simple bigrams model is terrible at providing accurate similarity scores for a word pair. We therefore conclude that while the experiment setup is useful, we cannot say more about the thresholds for relatedness because the computational model used was not good enough at capturing the strength of associations.

5.4.3 Empirical results for PMI

We would like to build a model that can give hints for 1, 2 and 3 words depending on the situation. Naturally, we would like to prioritize hints that target more words, so we propose a threshold model which gives hints using the Top3 model while these hints score above some threshold and switches to the Top2 model when no hint from the Top3 model passes this threshold anymore. Similarly, it will switch to the Top1 model if the score threshold for the Top2 model can no longer be surpassed by any hint. In order to build this model we will first need to determine adequate thresholds. We will define this threshold as the lowest acceptable similarity score of the first n positive words in relation to the hint. If the first n positive words do not cross this threshold, we switch to the next method with $n = n - 1$. The Top1 model is the last model in this sequence and acts as a fallback: if there are no hints that cross the threshold for the other models, we use the Top1 model. As such we do not need to determine a threshold for this model. To determine these thresholds we will study the decisions made by players playing with the Top1, 2 and 3 dependency models. We take the data from all games played before October 30^{th} and plot the PMI scores of words selected by players in these games in figures 5.6, 5.7 and 5.8. We do not show or discuss all figures here, because most of them show similar trends. The remaining figures can be found in appendix A.3.

Figure 5.6 shows the PMI scores for each card clicked by players in a game where the hints were generated by the Czech Top1 dependency model. The hinted cards are the cards with a word that belongs to the player and that was targeted by the AI with the hint given in the turn that the player selected the card. The positive cards are words that were clicked by the player and belonged to their team, but that were not hinted at during the turn that they were selected. The negative cards are words that were selected by the player, but do not belong their team at all, which can be either an enemy, neutral or assassin card.

The cards that the player selected which were not hinted at by the AI can be split into two categories: words for which the model either correctly or incorrectly predicted a PMI score of 0 and words for which the model predicted, hopefully correctly, a PMI value above 0.

For the second class of words it means that our weights have failed to an extent. They did not adequately protect against giving hints with high association with the negative cards. However, the remedy is simple: we can improve our weights or, instead of using relative negative thresholds, we can introduce an absolute negative threshold to prevent the model from giving these hints. An attempt to introduce such a threshold is made in Section 5.5.1 where we introduce the concept of threshold models.

The problem of words for which the PMI score is 0, is more difficult to solve. There are three reasons why a player can select a card with a PMI of 0 in relation to the hint: 1) the player got a hint related to this card in a previous turn, 2) the model incorrectly predicted that the word and hint are not associated or 3) the

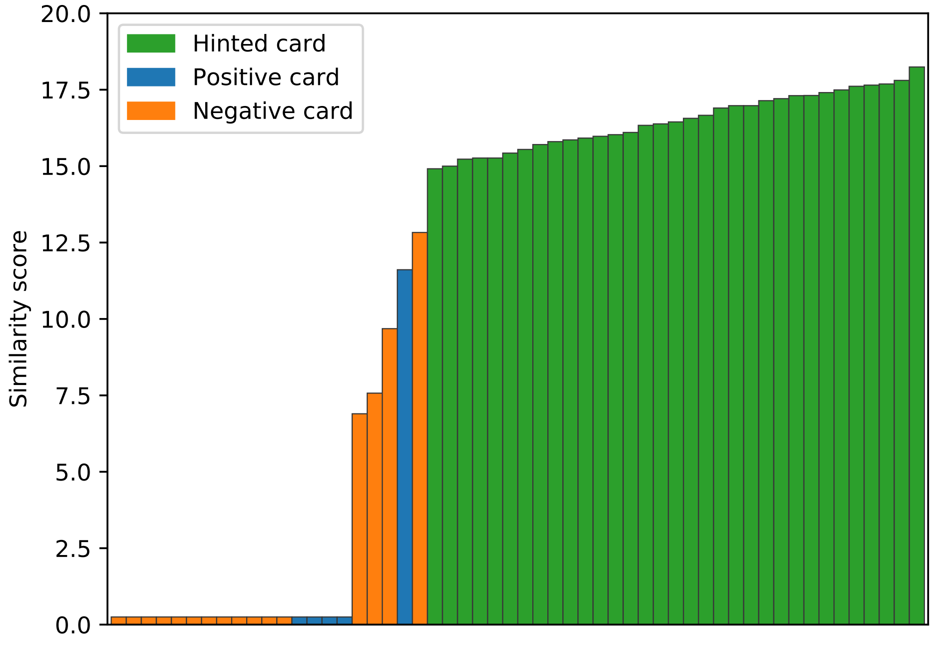


Figure 5.6: Similarity scores for each card clicked by players across several test games for the Czech Top1 dependency model

player selected the card even though the word is not associated with the hint. The last and least likely point is something we cannot change, people make mistakes which does not say anything about the quality of our model. The first point addresses what we will call **accidental hits** where a player selects a card that is related based on a previous hint. While they are not as good as **direct hits** where the player selects a word directly based on the current hint, the decisions are still a results of associations that the player makes on the hints provided by the model. Therefore they are not a reason of concern.

Now we shift our focus to the second case where the model incorrectly predicts an association between a word and the current hint. These **misses** are problematic because the similarity scores are not representative of the word association made by the player, and by proxy the word associations made by humans. They indicate a potential blind spot in our model for the PMI scores: we might not have enough data to accurately model the level of association between these words. This might be the case for words with a low frequency in our corpus, such as *kamion* for Czech (2 occurrences) and *platypus* for English (98 occurrences) as mentioned in Section 3.4.

We see the same trend across all methods: the cards with high PMI scores are the ones that were hinted at by the model. And there are a considerable amount of them for each model, which means that the model provides hints that the player understands, which explains why they perform better than the baseline as seen in Section 5.3.

We do not observe any major differences between Czech and English when it

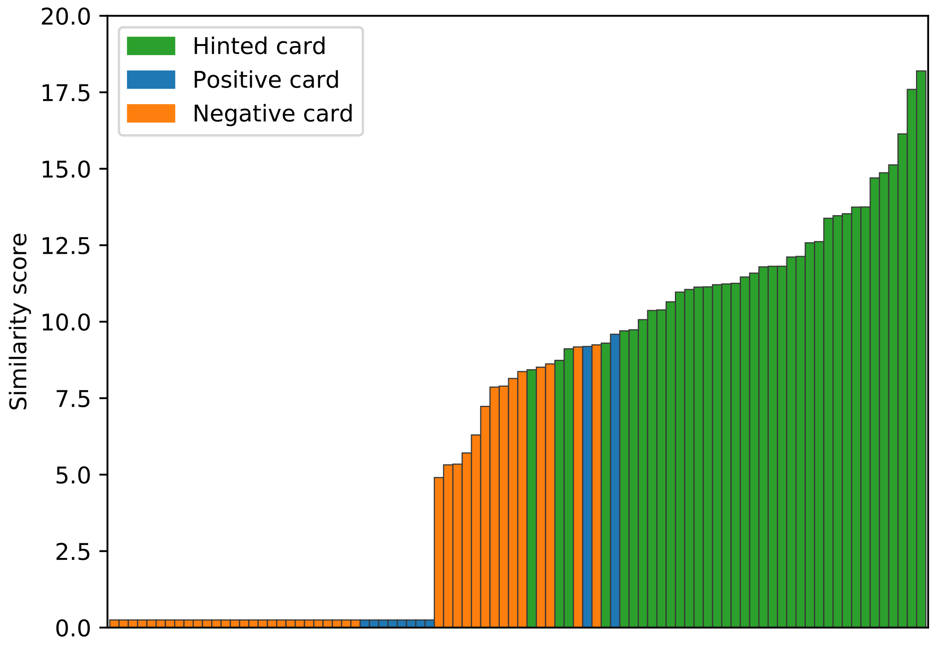


Figure 5.7: Similarity scores for each card clicked by players across several test games for the Czech Top3 dependency model

comes to the PMI scores of cards selected by the player.

The PMI scores of hinted cards that were not clicked are not as interesting, because they will always be high since the model selects the hint with the highest PMI values. Similarly, one would have to make very strong assumptions about the intent of the player in order to draw reasonable conclusions from the cards that were not clicked by the player. While it is reasonable to assume that the player considers the cards they click to be ”related”, not clicking a card does not necessarily imply a ”not related” judgment. Players often wait with clicking a card that they think is related, because they are not certain and want to get more information from the next hint. Other times players simply glance over one of the words while searching the board for words related to the hint. For these reasons we cannot assume that not clicking a card means that the player judged it as unrelated, which is why we do not show graphs of PMI scores for cards that were not clicked by the player.

5.4.4 Empirical results for cosine similarity

Similar to what we did for dependencies and PMI, we also show graphs of the cosine similarities for several games played using the Top1, 2 and 3 word embedding models.

The first thing to note is that these graphs are different from the graphs with PMI scores for they do not start with a sequence of zero values. Cosine similarity is a similarity metric which produces more gradual results in comparison to PMI.

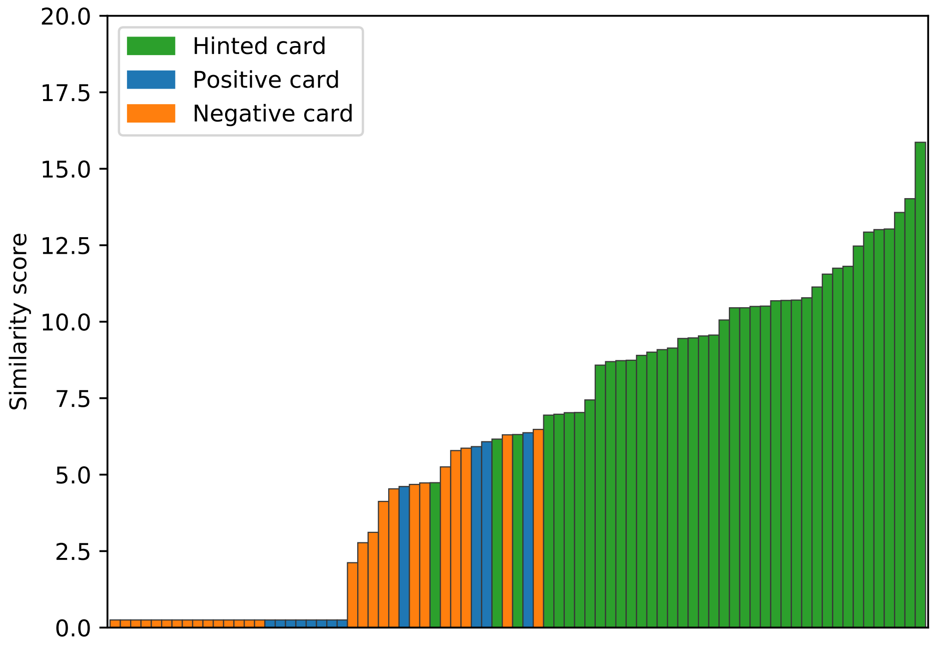


Figure 5.8: Similarity scores for each card clicked by players across several test games for the English Top3 dependency model

While it is very easy for PMI to be 0, which happens when two words never occur close together in a text, cosine similarity can only be 0 when two words are at opposite ends of the modelled space in all dimensions, which is very unlikely.

In Figure 5.9 and 5.12 we can see that the Top1 model for Czech and English behave the same. Even though we have many more data points for English than for Czech, we have one block of only hinted cards on the right and a block of negative cards with some positive cards mixed in on the left. What is so special about this is that, unlike the other models, for Top1 the cutoff point is clean. On one side there are only positive and negative cards and on the other side there are only hinted cards. This means that there was never a situation in which the model generated a hint with high similarity scores for negative cards and the player clicked on these cards. The Top1 model seems to be much better at selecting hints where only the target words have high similarity scores.

The right side of these graphs is clearly what we want: the player almost always clicks on one of their own cards which were hinted at by the model. The left side is what we don't want: the player clicking on incorrect cards or sometimes on correct cards that the model did not hint at.

These negative instances cannot be explained by out of turn selection of words by the player. While this is a valid reason for positive instances with low similarity score with the current hint, one would also expect to see negative words with high similarity scores if this was the case. Because we do not see such instances in the graph, we conclude that these negative instances must be caused by some other factor.

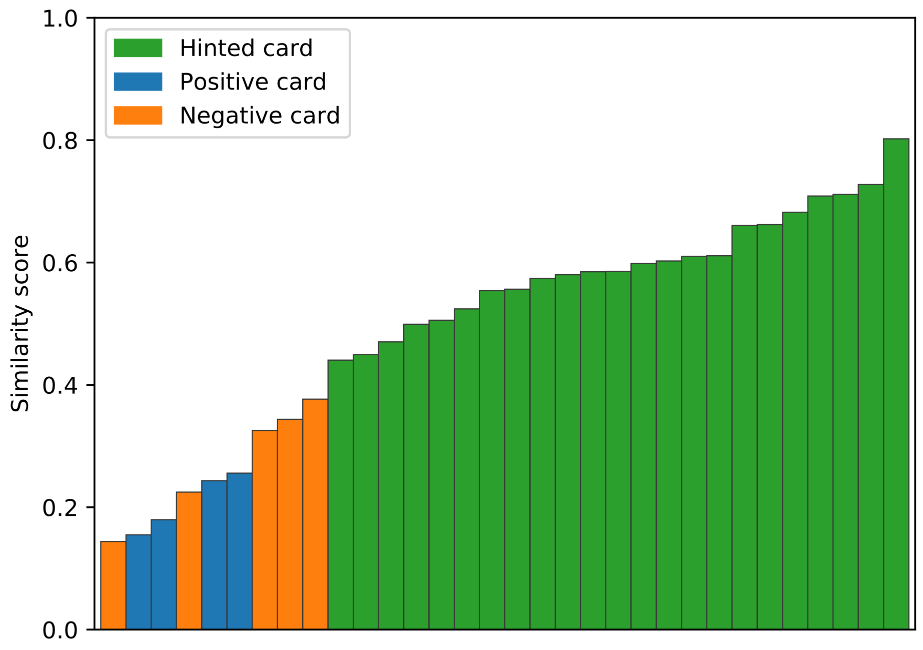


Figure 5.9: Similarity scores for each card clicked by players across several test games for the Czech Top1 word embedding model

This leaves us with two possibilities. Either the player selects negative instances with low similarity scores because the player sees them as very related, which means the similarity scores are incorrect. Or, alternatively, the player deems these negative instances to be only slightly related, but selects them because they do not see better alternatives. In this case improving our hints in such a way that they are more obvious to the player would increase the performance of our models. One such approach would be to find better hints with higher similarity scores to the words, which means improving our model. Another approach would be to make sure we only give hints above a certain threshold, which ensures that the gap in similarity between the hint and positive and negative words is always large enough. We explore this second approach in Section 5.5.1.

The idea of the threshold models is that we avoid hints with word similarity scores below the threshold. If all hints given by the model are hints for which all n target words cross this threshold, we expect the overall distribution of clicked cards to get closer to the results on the right side of the graph. We would then expect the player to mostly click cards that both belong to their team and are hinted at by the model.

The Top2 models are unique because they have the highest amount of positive and hinted cards interspersed throughout the block of negative cards in comparison to the other models. In Figure 5.10 and A.5 we can see that there are many words with low similarity scores that were selected by the player and were in fact part of their team. One possible explanation for this is that there are many cases where the model generates a hint, but the player is not certain about the

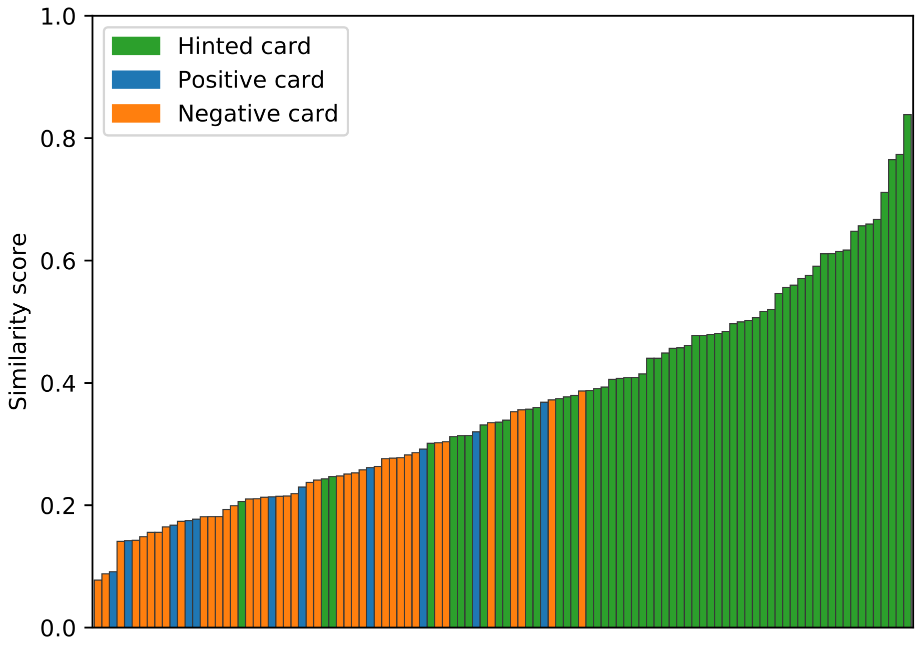


Figure 5.10: Similarity scores for each card clicked by players across several test games for the Czech Top2 word embedding model

association between the word and the hint and selects the word at a later turn.

In Figure 5.11 we see something that we have not seen before, the block of hinted cards is regularly interrupted by negative cards. While these instances are also visible to a limited extent in the other figures, they never extend above a cosine similarity of 0.4. For the Czech Top3 model the highest negative instance has a similarity score of 0.530. A similar outlier of 0.484 is present for the English model in Figure A.4. We suggest that these outliers exist because it is more difficult to hint at 3 words and not at any of the negative cards than it is to hint at only 1 card while avoiding the negative ones.

5.4.5 Empirical thresholds

	Czech		**English**	
Method	Dependency	Word embeddings	Dependency	Word embeddings
Top3	9.299	0.282	6.946	0.350
Top2	10.811	0.387	9.653	0.371

Table 5.7: Empirically derived thresholds for dependency and word embedding models generating hints for 3 and 2 words in Czech and English.

We choose thresholds by selecting the lowest value of the leftmost block of 4 or more hinted and positive cards in each graph. The resulting thresholds can be

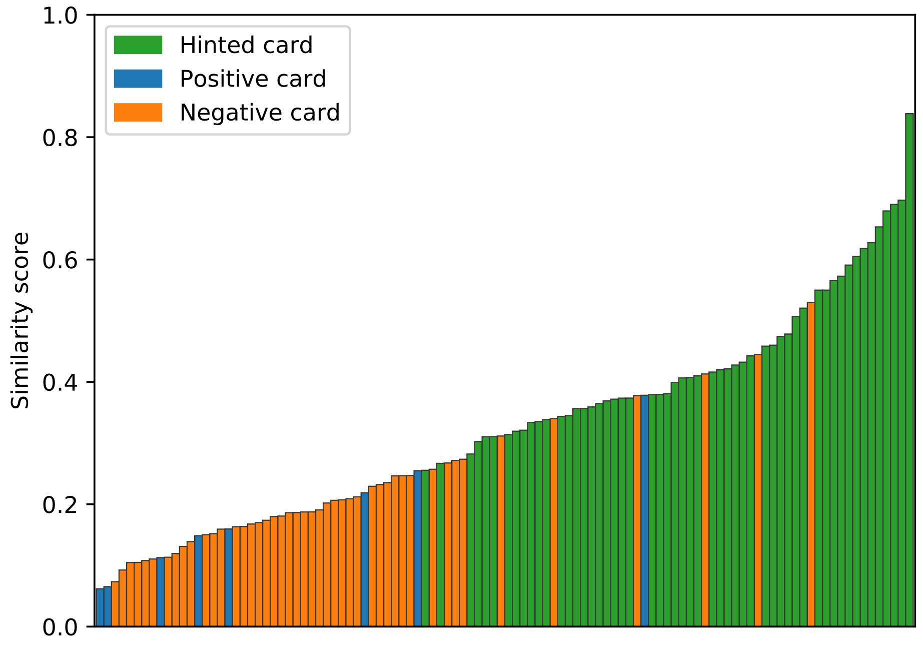


Figure 5.11: Similarity scores for each card clicked by players across several test games for the Czech Top3 word embedding model

found in Table 5.7. The difference in the thresholds taken for Czech and English is quite large for the dependency model. This is expected, because the PMI scores that these thresholds are based on were not extracted from the same text. Even though CzEng is a parallel corpus, the different distributional characteristics of the two languages lead to two separate PMI scales. When inspecting the data we can see that the PMI values for Czech are structurally higher than the ones for English. For example for the Top3 dependency model, the consecutive block of hinted cards on the right goes from 9.701 to 18.199 for Czech, while it starts at 6.946 and ends at 15.866 for English.

When comparing between methods, the difference between the thresholds for Top3 and Top2 for the English dependency model (6.946 and 9.653) is very large compared to Czech (9.299 and 10.811), while the difference in thresholds for the English word embeddings model (0.350 and 0.371) is quite small compared to Czech (0.282 and 0.387). It is difficult to explain these differences, other than the idea that more data might produce more regular thresholds. As to the quality of the thresholds, this will become clear when we test these models in Section 5.5.1.

The similarity scores for word embeddings are normalized. As such, the threshold for Top2 for Czech and English (0.371 and 0.387) are more comparable for this method. It is likely that some value between 0.371 and 0.387 is the optimal threshold for a Top3 word embeddings model.

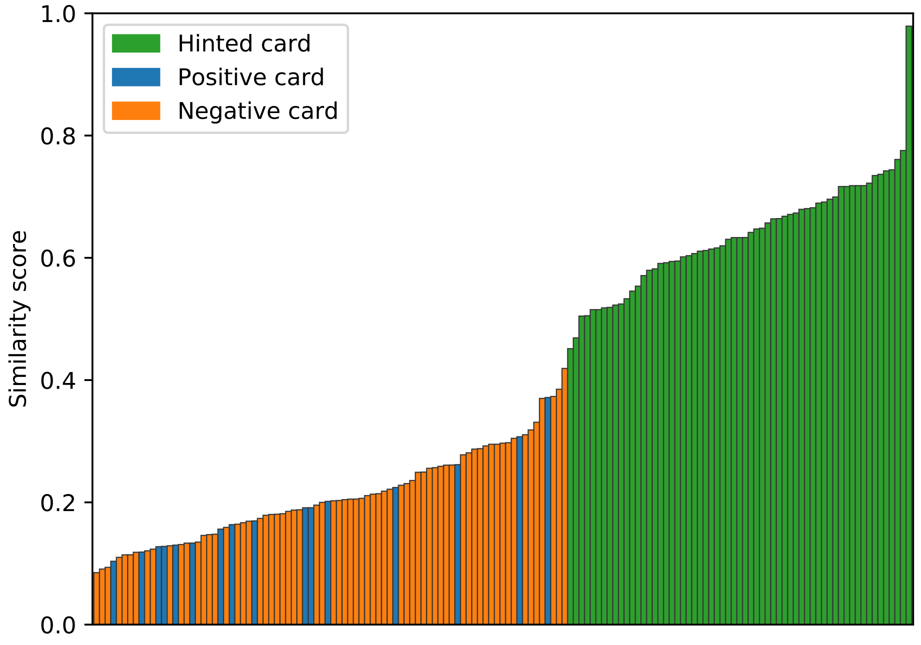


Figure 5.12: Similarity scores for each card clicked by players across several test games for the English Top1 word embedding model

5.5 Ensemble models

5.5.1 Threshold models

We experiment with an ensemble model for both dependencies and word embeddings using the thresholds determined in the previous section. A model consists of three submodels which we have already tested individually so we can see if there is an improvement. Hints are chosen by querying the Top3, Top2 and Top1 models in that order and selecting the first hint from the model that passes its respective threshold, defaulting to the Top1 model if none of the thresholds are passed. The TopN dependency model uses the Top3, Top2 and Top1 dependency models with thresholds of 9.299 and 10.811 for Czech and 6.946 and 9.653 for English. The TopN word embeddings model uses the Top3, Top2 and Top1 dependency models with thresholds of 0.282 and 0.387 for Czech and 0.350 and 0.371 for English.

Table 5.8 shows the results of the TopN models and the individual models. The dependency model performed very poorly with an f-score of 0.591 for Czech and 0.519 for English. It did not manage to outperform even the worst individual model, which was the Top3 model with f-scores of 0.595 and 0.632. The performance of the English model is exceptionally bad when contrasted with the performance of its worst submodel (0.519 and 0.632) and performs much worse than the Czech model in this regard (0.591 and 0.595). We hypothesize that the low threshold for the English Top3 model, the worst performing model, has contributed significantly to this poor performance. The Czech model has a higher

Setup	Player decisions					
	CZ			EN		
Aggregation	P	R	F_1	P	R	F_1
Random						
Baseline	0.389	0.339	0.362	0.389	0.339	0.362
Dependency-level col.						
Top1	**0.722**	0.633	0.675	**0.693**	0.678	**0.685**
Top2	0.621	**0.778**	**0.691**	0.646	**0.711**	0.677
Top3	0.552	0.644	0.595	0.655	0.611	0.632
TopN	0.598	0.585	0.591	0.570	0.476	0.519
Word embeddings						
Top1	**0.789**	**0.789**	**0.789**	**0.768**	0.735	**0.751**
Top2	0.608	0.690	0.647	0.614	0.735	0.669
Top3	0.574	0.626	0.599	0.667	**0.786**	0.722
TopN	0.673	0.623	0.647	0.738	0.679	0.707

Table 5.8: Micro-averaged precision, recall and f-score for the top-n dependency and word embedding models. Highest precision, recall and f-score are marked in bold separately for each combination of language and setup with more than one aggregation method.

threshold and a much smaller gap between the threshold of the Top3 and Top2 model as mentioned in Section 5.4.5. In addition, we can say that the ensemble method has not had the desired effect. While we would expect from an ensemble method that it would perform equally or better than the worst performing model, our English dependency model performed much worse than the worst individual model.

For word embeddings the picture looks slightly better. The TopN models perform worse than the best individual model with an f-score of 0.647 versus 0.789 for Czech and 0.707 versus 0.751 for English, but better than the worst individual model (0.599 and 0.669). While this performance is certainly better than that of the TopN dependency model, it does not improve over the best individual model in any way. When we look at Figure 5.13 we see that the threshold model did not prevent the player from selecting cards with low similarity scores. The graph looks similar in shape and distribution to Figure 5.11 which we were trying to improve on. The number of positive cards selected that were not hinted at in the current turn is much higher, which explains why the model has a higher precision than the Top2 and 3 models. Therefore we conclude that the threshold system successfully improves the precision of the model. However, this happened at the cost of recall. And it still performs worse than the Top1 model across all statistics.

All TopN models suffered in terms of recall when compared to the individual models. None of the TopN models has higher recall than the lowest recall of the submodels that it is built from. Precision on the other hand increased considerably in comparison to the Top3 and Top2 models.

We suspect that the thresholds we have set for the word embedding models

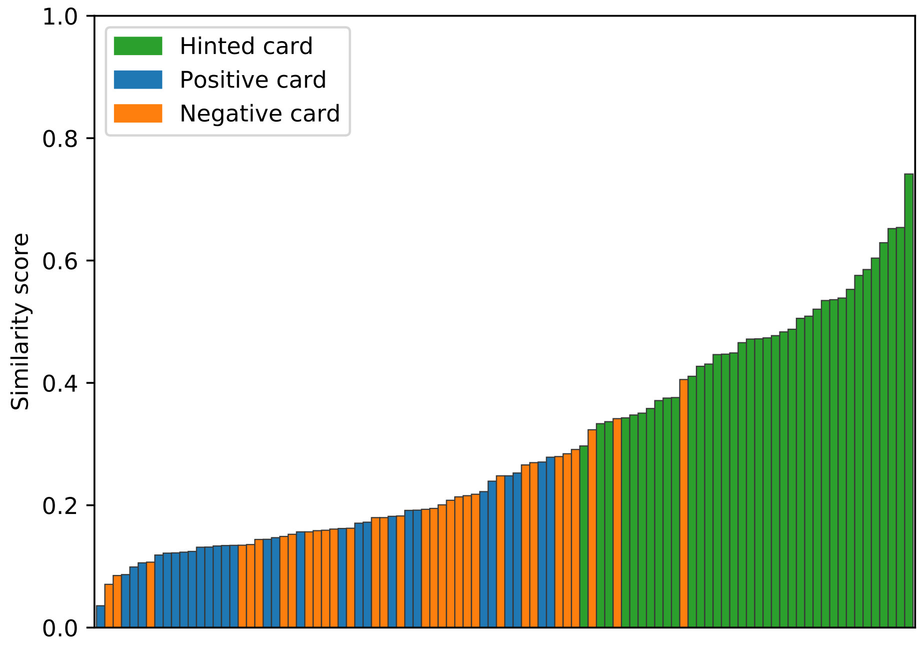


Figure 5.13: Similarity scores for each card clicked by players across several games for the Czech TopN word embedding model

are better than the ones for the dependency models. Furthermore, the threshold model might be better suited for word embeddings in general than for collocations, because the cosine similarity is a normalized scale and pointwise mutual information is not. However, the threshold model did not live up to expectations, because it did not prevent the player from clicking on cards with low similarity scores in regards to the hint.

To conclude, the threshold dependency model performed worse than even a baseline expectation of an ensemble model for English. The threshold word embedding model achieved moderate results, performing within the range of the best and worst submodels that it contains. As such, we conclude that thresholds are not a good method for ensembling.

The threshold models do not behave the way we expected them to; they do not solve the problem of words with low similarity scores being clicked by the player.

We suspect that the selected thresholds are far from optimal and a linear approach might achieve better results. Finding a good way to combine the Top1, 2, 3 methods to achieve the same or better performance than either of the individual methods is an interesting direction for future research.

5.5.2 Combined models

Lastly, we would like to test a model that combines both dependency collocations and word embedding models. Since the threshold system turned out to be a poor ensembling method, we have to consider a new way in which we can combine our models. One method is to find a mapping between PMI values and cosine similarity. However, one of these measures is normalized and the other is not and their scales are radically different, so this relationship can be hard to find through trial-and-error, and is in the worst case non-linear. Instead we choose to perform ensembling through mutual agreement, where we let both models predict hints, until one of the models gives a hint that the other model has also predicted for the current board state.

We expect an ensemble model that combines word embeddings and collocations to perform better than the individual models, because they both model very different things. Word embeddings capture similarity while collocations usually capture other types of relations. Combining the best of both models should lead to better results.

We test an ensemble model that combines the TopN dependency and TopN word embedding models described in Section 5.5.1 through mutual agreement for Czech and English.

Setup		**Player decisions**					
		CZ			**EN**		
	Aggregation	P	R	F_1	P	R	F_1
Random							
	Baseline	0.389	0.339	0.362	0.389	0.339	0.362
Dependency-level col.							
	TopN	0.598	0.585	0.591	0.570	0.476	0.519
Word embeddings							
	TopN	0.673	0.623	0.647	0.738	0.679	0.707
Dep. collocations & WE							
	TopN - mutual	0.642	0.678	0.659	0.711	0.697	0.704

Table 5.9: Micro-averaged precision, recall and f-score for the combined TopN dependency and word embedding model.

In Table 5.9 we can see the results of combining dependency and word embedding models by finding hints through mutual agreement between models. The combined model performed similarly to the best models included in them with an f-score of 0.659 for Czech and 0.704 for English. The f-score of the TopN word embedding model is slightly lower for Czech (0.647) and slightly higher for English (0.707).

Although these results are promising, they do not significantly improve the results of the models they combine. The model is successful at mimicking the performance of the best internal model, but it does not select either or depending on what is best in a given situation. This is due to the ensembling method used. As such, more research on good ensembling methods is needed to find models that do improve above the performance of their internal parts.

In Table 5.10 we show the number of games played and number of decisions made for each model. The number of decisions for a model is the sum of all the cards clicked by players in all the games played with that model.

Setup / **Aggregation**	**Player decisions** CZ #G	CZ #D	**EN** #G	EN #D
Sentence-level collocations				
CombinedMax	17	148	62	520
Dependency-level col.				
CombinedMax	17	159	68	556
MeanDiff	18	179	67	601
Top1	10	79	10	88
Top2	11	124	10	99
Top3	10	105	10	84
TopN	10	92	10	86
Word embeddings				
most_similar	22	242	65	630
CombinedMax	25	233	77	741
Top1	10	90	13	112
Top2	14	143	13	140
Top3	11	108	13	138
TopN	10	98	11	103
Dep. collocations & WE				
TopN - mutual	10	95	11	97

Table 5.10: Number of games played and number of decisions made for all models tested.

6. Conclusion

Setup	Player decisions					
	CZ			EN		
Aggregation	P	R	F_1	P	R	F_1
Random						
Baseline	0.389	0.339	0.362	0.389	0.339	0.362
Sentence-level collocations						
CombinedMax	0.507	0.490	0.498	0.500	0.466	0.482
Dependency-level col.						
CombinedMax	0.629	0.654	0.641	0.547	0.497	0.521
MeanDiff	0.575	0.636	0.604	0.546	0.544	0.545
Top1	**0.722**	0.633	0.675	**0.693**	0.678	**0.685**
Top2	0.621	**0.778**	**0.691**	0.646	**0.711**	0.677
Top3	0.552	0.644	0.595	0.655	0.611	0.632
TopN	0.598	0.585	0.591	0.570	0.476	0.519
Word embeddings						
most_similar	0.616	0.753	0.677	0.563	0.607	0.584
CombinedMax	0.558	0.578	0.568	0.567	0.606	0.586
Top1	**0.789**	**0.789**	**0.789**	**0.768**	0.735	**0.751**
Top2	0.608	0.690	0.647	0.614	0.735	0.669
Top3	0.574	0.626	0.599	0.667	**0.786**	0.722
TopN	0.673	0.623	0.647	0.738	0.679	0.707
Dep. collocations & WE						
TopN - mutual	0.642	0.678	0.659	0.711	0.697	0.704

Table 6.1: Micro-averaged precision, recall and f-score for all models tested, including baseline. Highest precision, recall and f-score are marked in bold separately for each combination of language and setup with more than one aggregation method.

We have provided both a theoretical and a practical framework for the evaluation of computational models of word association. We started out by establishing a baseline for the task of Codenames with a single human player. After this we explored several methods all of which performed well above the baseline f-score of 0.362. The restriction of dependency-level bigrams proved a definite improvement over broad sentence-level bigrams for the collocation model, beating the sentence-level collocations model with an f-score of 0.641 versus 0.498 for Czech and 0.521 versus 0.482 for English.

Large improvements to our model were made by aggregating the similarity scores of the words on the board and weighting them more cleverly, as well as the introduction of the number of target words which was presented to the player alongside the hint. Our best dependency models achieved an f-score of 0.691 for Czech and 0.685 for English. The word embedding models based on the same aggregation technique in turn outclassed these models with f-scores of 0.789 and

0.751 for Czech and English respectively. The model that got closest to helping the player turn over all their cards, was the Top2 dependency model for Czech with a recall of 0.778. For English the best model in this regard was the Top3 word embeddings model with a recall of 0.786. We made several attempts to build ensemble models that combine the best performing models to boost their performance. We were not successful in this regard, our TopN dependency model achieved f-scores of 0.591 and 0.519 for Czech and English respectively. The TopN word embeddings performed better, with an f-score of 0.647 for Czech and 0.707 for English, but neither outperformed the best individual top-n model for their respective language. A final attempt at combining dependency and word embedding models by finding hints through mutual agreement between models, performed similarly to the best models included in them with an f-score of 0.659 for Czech and 0.704 for English. Although these results are promising, we believe that many better ensembling methods still remain.

We have shown that both dependency-level collocation models and word embedding models can provide hints of considerable quality, given the right constraints. Dependency models manage to capture several types of relations between words which the player is able to pick up on, while the word embedding models excel at finding semantically similar hints.

6.1 Future Work

We have provided an overview of only the most basic methods and we believe that many improvements can still be made to achieve better performance on the Codenames word association task. For example by finding better ensemble methods to combine models that give hints for a different number of words, as well as successfully combining models of different types such as collocation and word embedding models. Introducing an extra weighting step into the pipeline might also prove useful, such as multiplying the final score of a possible hint by the log of its frequency in the CzEng corpus, similar to the method described in Obrtlík [2018].

The methods we use are themselves simple baselines for the technique that they are based on. There exist many more measures of association other than pointwise mutual information and there have been many improvements in recent years over the FastText word embeddings that we tested, many of which might surpass our best word embedding models when compared. This could be a fruitful direction for future research.

The same framework can be used to analyze the effect of time taken between receiving the word prompt and making a decision. We did not incorporate any timing mechanism in our application, so it is not possible to extract this type of information from our dataset. However, it is easy to modify the application and record this data as well, so this is nonetheless an interesting avenue for future work.

While this thesis was mainly focussed on the computational side of word association, it must be noted that a human baseline for the Codenames word association task would be very useful to give more context to the results achieved on this task. Similarly, comparing the predictions made by the models to human-level word associations would also be a useful direction in this area.

Bibliography

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. ISSN 2307-387X.

Ondřej Bojar, Ondřej Dušek, Tom Kocmi, Jindřich Libovický, Michal Novák, Martin Popel, Roman Sudarikov, and Dušan Variš. CzEng 1.6: Enlarged Czech-English Parallel Corpus with Processing Tools Dockered. In Petr Sojka, Aleš Horák, Ivan Kopeček, and Karel Pala, editors, *Text, Speech, and Dialogue: 19th International Conference, TSD 2016*, number 9924 in Lecture Notes in Computer Science, pages 231–238, Cham / Heidelberg / New York / Dordrecht / London, 2016. Masaryk University, Springer International Publishing. ISBN 978-3-319-45509-9.

Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

Gemma Bel Enguix, Reinhard Rapp, and Michael Zock. A graph-based approach for computing free word associations. In *LREC*, pages 3027–3033, 2014.

Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, 2014.

Yang Liu, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. Topical word embeddings. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

Douglas L Nelson, Cathy L McEvoy, and Thomas A Schreiber. The university of south florida free association, rhyme, and word fragment norms. *Behavior Research Methods, Instruments, & Computers*, 36(3):402–407, 2004.

Petr Obrtlík. Computer as an intelligent partner in the word-association game codenames. Master's thesis, Brno University of Technology, Brno, 2018.

Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

Princeton University. "About WordNet." WordNet, 2010. URL `https://wordnet.princeton.edu/`.

Manfred Wettler and Reinhard Rapp. Computation of word associations based on co-occurrences of words in large corpora. In *VERY LARGE CORPORA: ACADEMIC AND INDUSTRIAL PERSPECTIVES*, 1993.

List of Figures

List of Tables

List of Abbreviations

CM	Combined maximum
MD	Mean difference
PMI	Pointwse Mutual Information
WE	Word Embeddings

A. Attachments

A.1 Lexicon for Czech

AFRIKA	AMERIČAN	AMERIKA	ANDĚL
ANGLIE	ASIE	ATLET	AUSTRÁLIE
AUTO	AUTOBUS	BAČKORA	BANÁN
BANKÉŘ	BASA	BÁSNÍK	BERLÍN
BETON	BIOLOG	BOTA	BRAMBORA
BRÁNA	BRATISLAVA	BRATR	BRAZÍLIE
BŘICHO	BRNO	BROUK	BRÝLE
BUBLINA	BUŇKA	ČARODĚJ	ČECH
ČEPICE	ČERT	CESTA	CESTOVATEL
CHLÉB	CHOBOTNICE	ČÍNA	CIRKUS
CIZINEC	ČOČKA	ČOKOLÁDA	CUKR
DĚLNÍK	DĚLO	DÉMON	DÉŠŤ
DIAMANT	DINOSAURUS	DÍTĚ	DOKTOR
DRAK	DRÁT	DŘEVO	DUB
DUCH	DŮM	DVEŘE	DÝKA
DŽUNGLE	FIGURKA	FILM	FILOZOF
FLÉTNA	FOTBALISTA	FRANCIE	FYZIK
GUMA	HÁK	HEREC	HLAS
HLAVA	HLÍNA	HLINÍK	HODINKY
HOKEJISTA	HOLANĎAN	HORA	HOSPODA
HOUBA	HRA	HRAD	HŘBITOV
HŘEBEN	HŘEBÍK	HRNEC	HUDBA
HŮL	HVĚZDA	ITÁLIE	JABLKO
JÁDRO	JARO	JAZYK	JEDNOROŽEC
JEHLA	JEŘÁB	JESKYNĚ	JEŠTĚRKA
JEZDEC	JEŽEK	JEZERO	KABÁT
KAKTUS	KALHOTY	KAMARÁD	KÁMEN
KAMION	KANADA	KAPR	KARBANÍK
KARTA	KARTÁČ	KENTAUR	KEŘ
KINO	KLADIVO	KLAUN	KLÁVESNICE
KLAVÍR	KLEŠTĚ	KLÍČ	KLIKA
KLOBOUK	KLOKAN	KMEN	KNEDLÍK
KNIHA	KNIHOVNA	KNÍR	KOČKA
KOLEJ	KOLENO	KOLO	KOLOBĚŽKA
KOMETA	KOMÍN	KONEV	KONÍK
KOŘEN	KOŘENÍ	KORUNA	KOŠ
KOSA	KOŠILE	KOSMONAUT	KOSTKA
KOULE	KOUZLO	KRÁL	KRÁPNÍK

KRÁVA	KŘÍŽ	KROKODÝL	KUŘE
KVĚTINA	KYTARA	KYVADLO	LÁHEV
LAMPA	LASER	LÁSKA	LÁTKA
LÁVA	LED	LES	LETADLO
LÉTO	LEV	LIMONÁDA	LIŠKA
LIST	LÍZÁTKO	LOCHNESKA	LOKOMOTIVA
LONDÝN	LOPATA	LOS	LOUKA
LUPIČ	LŽÍCE	MAĎAR	MARS
MASO	MATEMATIK	MATKA	MEČ
MEDVĚD	MELOUN	MĚSÍC	MĚSTO
MÍČ	MÍR	MLÉKO	MOŘE
MOSKVA	MOTORKA	MOTÝLEK	MOUKA
MRAKODRAP	MRAVENEC	MRÁZ	MRKEV
MUCHOMŮRKA	MYŠ	NÁDRAŽÍ	NĚMECKO
NEMOC	NEMOCNICE	NEPŘÍTEL	NETOPÝR
NINJA	NOC	NOHA	NOS
NŮŽ	OBCHOD	OBR	OHEŇ
OKNO	OKO	OMÁČKA	OREL
ORGÁN	OSTROV	OTEC	OVCE
PANÁK	PÁNEV	PANNA	PAPÍR
PAPOUŠEK	PAŘÍŽ	PARK	PARNÍK
PAS	PAVOUK	PEC	PEKING
PES	PILA	PISTOLE	PIVO
PLACKA	PLANETA	PLAST	PLÁŠŤ
PLASTELÍNA	PLOT	PLYN	POČÍTAČ
PODNIKATEL	PODVODNÍK	PODZIM	POEZIE
POHÁDKA	PÓL	POLE	POLÉVKA
POLICISTA	POMERANČ	PONOŽKA	POPEL
POUŠŤ	PRÁCE	PRACH	PRAČKA
PRAHA	PRASE	PREZIDENT	PRINCEZNA
PRODAVAČ	PROGRAMÁTOR	PROVAZ	PRSTEN
PTAKOPYSK	RÁDIO	RADOST	RAJČE
RAKETA	ŘECKO	ŘEKA	ŘETĚZ
ŘIDIČ	ROBOT	ROH	ROLE
ROPA	RUČNÍK	RUKA	RUS
RŮŽE	SALÁM	SALÁT	SAVEC
SEDLÁK	SEDMIKRÁSKA	SEKERA	SESTRA
SILNICE	ŠIPKA	SKLO	ŠKOLA
SKŘÍTEK	SLON	SLUNCE	SMRK
SMRT	SMŮLA	SNĚŽENKA	SNÍH
ŠPAGETA	ŠPANĚL	SRDCE	ŠROUBEK
ŠROUBOVÁK	ŠTĚSTÍ	ŠTIKA	ŠTÍR
STŘELEC	STŘÍBRO	STROM	SUKNĚ
SŮL	SUPERHRDINA	ŠVESTKA	SVĚTLO

SVÍČKA	SÝR	TALÍŘ	TAŠKA
TELEFON	TELEVIZE	TLAČÍTKO	TOPOL
TRÁVA	TŘEŠEŇ	TROUBA	TRPASLÍK
TRUBKA	TUČŇÁK	TULIPÁN	TUŽKA
TYGR	UCHO	UČITEL	ÚDOLÍ
UHLÍ	UMĚLEC	UPÍR	ÚŘAD
VÁHA	VÁLKA	VĚDEC	VEJCE
VELRYBA	VENUŠE	VESMÍR	VESNICE
VĚTRNÍK	VĚŽ	VĚZEŇ	VÍČKO
VÍDEŇ	VIDLIČKA	VÍLA	VÍNO
VÍRA	VÍTR	VLAK	VLNA
VODA	VODNÍK	VOJEVŮDCE	VRCHOL
VŮZ	VZDUCH	YETTI	ZADEK
ZÁKON	ZÁKUSEK	ZÁMEK	ZEBRA
ŽEBRÁK	ŽEBRO	ZEĎ	ŽEHLIČKA
ŽELEZO	ZELÍ	ŽELVA	ZEMĚ
ZIMA	ZLATO	ZOMBIE	ZPĚVÁK
ŽRALOK	ZRCADLO	ZUB	ZVONEK

A.2 Lexicon for English

AFRICA	AGENT	AIR	ALIEN
ALPS	AMAZON	AMBULANCE	AMERICA
ANGEL	ANTARCTICA	APPLE	ARM
ATLANTIS	AUSTRALIA	AZTEC	BACK
BALL	BAND	BANK	BAR
BARK	BAT	BATTERY	BEACH
BEAR	BEAT	BED	BEIJING
BELL	BELT	BERLIN	BERMUDA
BERRY	BILL	BLOCK	BOARD
BOLT	BOMB	BOND	BOOM
BOOT	BOTTLE	BOW	BOX
BRIDGE	BRUSH	BUCK	BUFFALO
BUG	BUGLE	BUTTON	CALF
CANADA	CAP	CAPITAL	CAR
CARD	CARROT	CASINO	CAST
CAT	CELL	CENTAUR	CENTER
CHAIR	CHANGE	CHARGE	CHECK
CHEST	CHICK	CHINA	CHOCOLATE
CHURCH	CIRCLE	CLIFF	CLOAK
CLUB	CODE	COLD	COMIC
COMPOUND	CONCERT	CONDUCTOR	CONTRACT
COOK	COPPER	COTTON	COURT
COVER	CRANE	CRASH	CRICKET
CROSS	CROWN	CYCLE	CZECH
DANCE	DATE	DAY	DEATH
DECK	DEGREE	DIAMOND	DICE
DINOSAUR	DISEASE	DOCTOR	DOG
DRAFT	DRAGON	DRESS	DRILL
DROP	DUCK	DWARF	EAGLE
EGYPT	EMBASSY	ENGINE	ENGLAND
EUROPE	EYE	FACE	FAIR
FALL	FAN	FENCE	FIELD
FIGHTER	FIGURE	FILE	FILM
FIRE	FISH	FLUTE	FLY
FOOT	FORCE	FOREST	FORK
FRANCE	GAME	GAS	GENIUS
GERMANY	GHOST	GIANT	GLASS
GLOVE	GOLD	GRACE	GRASS
GREECE	GREEN	GROUND	HAM
HAND	HAWK	HEAD	HEART
HELICOPTER	HIMALAYAS	HOLE	HOLLYWOOD
HONEY	HOOD	HOOK	HORN
HORSE	HORSESHOE	HOSPITAL	HOTEL
ICE	ICE CREAM	INDIA	IRON
IVORY	JACK	JAM	JET

JUPITER	KANGAROO	KETCHUP	KEY
KID	KING	KIWI	KNIFE
KNIGHT	LAB	LAP	LASER
LAWYER	LEAD	LEMON	LEPRECHAUN
LIFE	LIGHT	LIMOUSINE	LINE
LINK	LION	LITTER	LOCH NESS
LOCK	LOG	LONDON	LUCK
MAIL	MAMMOTH	MAPLE	MARBLE
MARCH	MASS	MATCH	MERCURY
MEXICO	MICROSCOPE	MILLIONAIRE	MINE
MINT	MISSILE	MODEL	MOLE
MOON	MOSCOW	MOUNT	MOUSE
MOUTH	MUG	NAIL	NEEDLE
NET	NEW YORK	NIGHT	NINJA
NOTE	NOVEL	NURSE	NUT
OCTOPUS	OIL	OLIVE	OLYMPUS
OPERA	ORANGE	ORGAN	PALM
PAN	PANTS	PAPER	PARACHUTE
PARK	PART	PASS	PASTE
PENGUIN	PHOENIX	PIANO	PIE
PILOT	PIN	PIPE	PIRATE
PISTOL	PIT	PITCH	PLANE
PLASTIC	PLATE	PLATYPUS	PLAY
PLOT	POINT	POISON	POLE
POLICE	POOL	PORT	POST
POUND	PRESS	PRINCESS	PUMPKIN
PUPIL	PYRAMID	QUEEN	RABBIT
RACKET	RAY	REVOLUTION	RING
ROBIN	ROBOT	ROCK	ROME
ROOT	ROSE	ROULETTE	ROUND
ROW	RULER	SATELLITE	SATURN
SCALE	SCHOOL	SCIENTIST	SCORPION
SCREEN	SCUBA DIVER	SEAL	SERVER
SHADOW	SHAKESPEARE	SHARK	SHIP
SHOE	SHOP	SHOT	SINK
SKYSCRAPER	SLIP	SLUG	SMUGGLER
SNOW	SNOWMAN	SOCK	SOLDIER
SOUL	SOUND	SPACE	SPELL
SPIDER	SPIKE	SPINE	SPOT
SPRING	SPY	SQUARE	STADIUM
STAFF	STAR	STATE	STICK
STOCK	STRAW	STREAM	STRIKE
STRING	SUB	SUIT	SUPERHERO
SWING	SWITCH	TABLE	TABLET
TAG	TAIL	TAP	TEACHER
TELESCOPE	TEMPLE	THEATER	THIEF
THUMB	TICK	TIE	TIME
TOKYO	TOOTH	TORCH	TOWER

TRACK	TRAIN	TRIANGLE	TRIP
TRUNK	TUBE	TURKEY	UNDERTAKER
UNICORN	VACUUM	VAN	VET
WAKE	WALL	WAR	WASHER
WASHINGTON	WATCH	WATER	WAVE
WEB	WELL	WHALE	WHIP
WIND	WITCH	WORM	YARD

A.3 Similarity scores

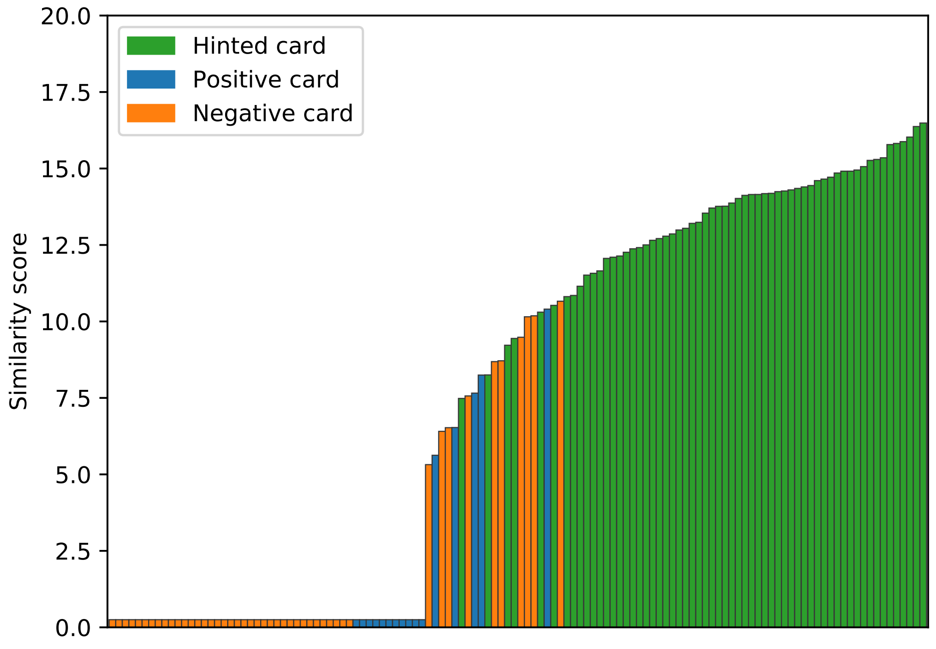


Figure A.1: Similarity scores for each card clicked by players across several test games for the Czech Top2 dependency model

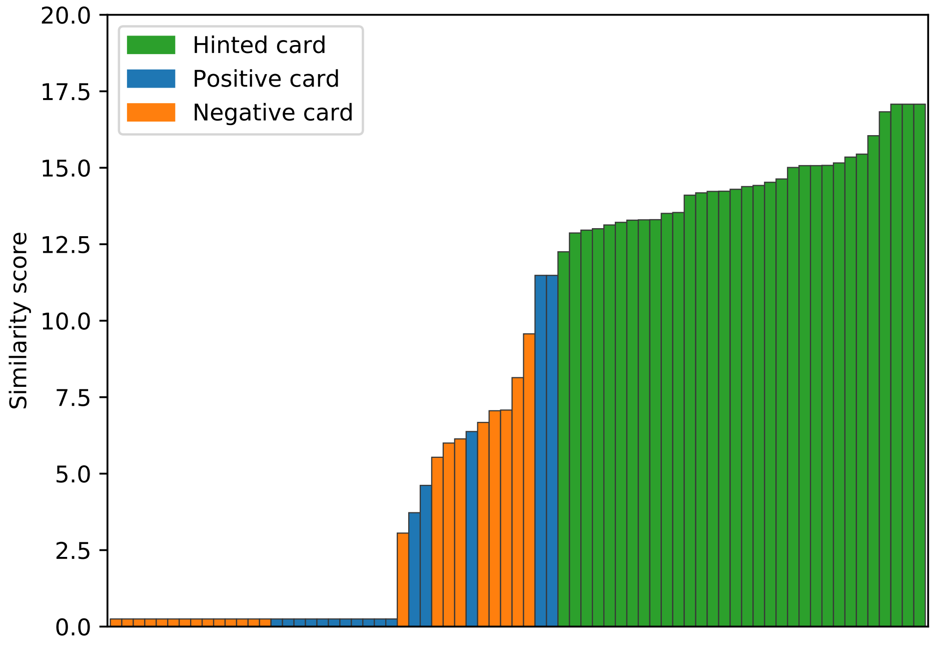


Figure A.2: Similarity scores for each card clicked by players across several test games for the English Top1 dependency model

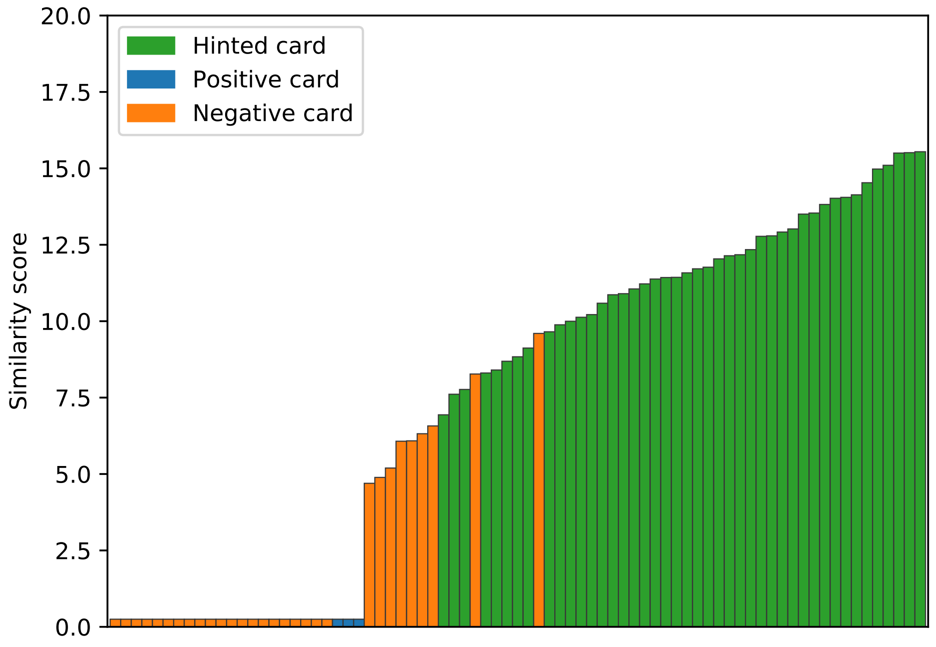


Figure A.3: Similarity scores for each card clicked by players across several test games for the English Top2 dependency model

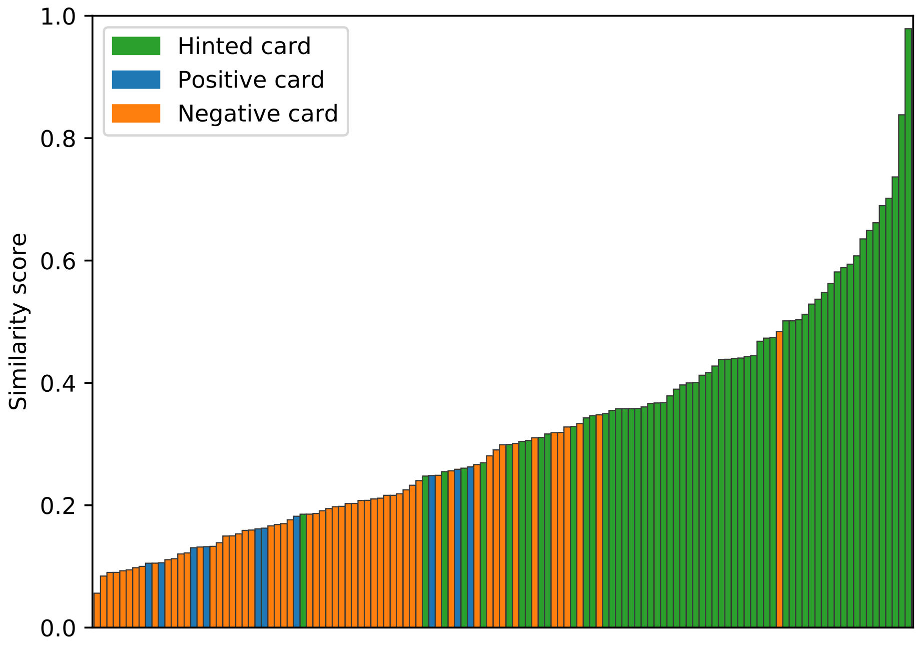


Figure A.4: Similarity scores for each card clicked by players across several test games for the English Top3 word embedding model

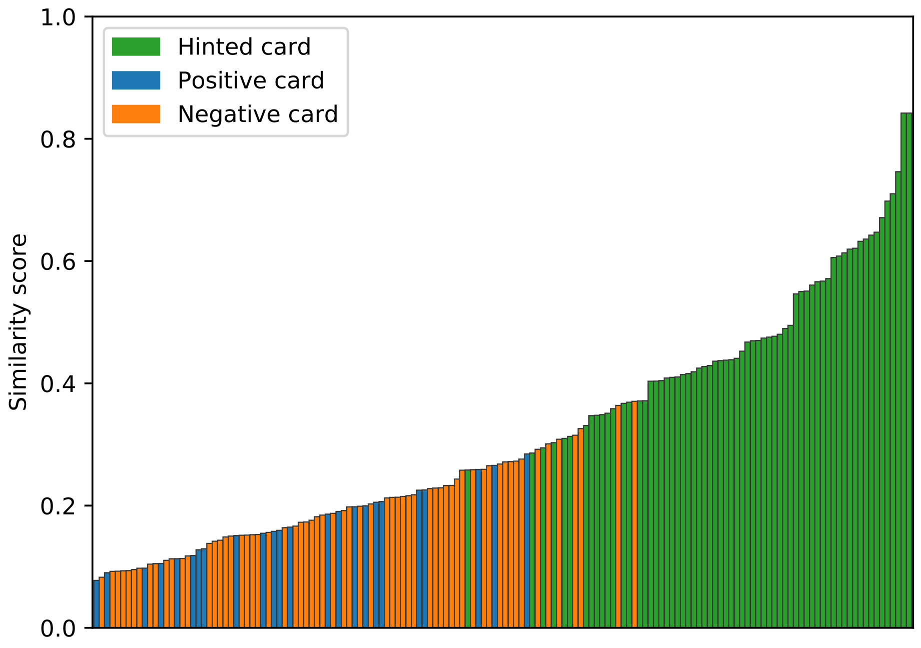


Figure A.5: Similarity scores for each card clicked by players across several test games for the English Top2 word embedding model

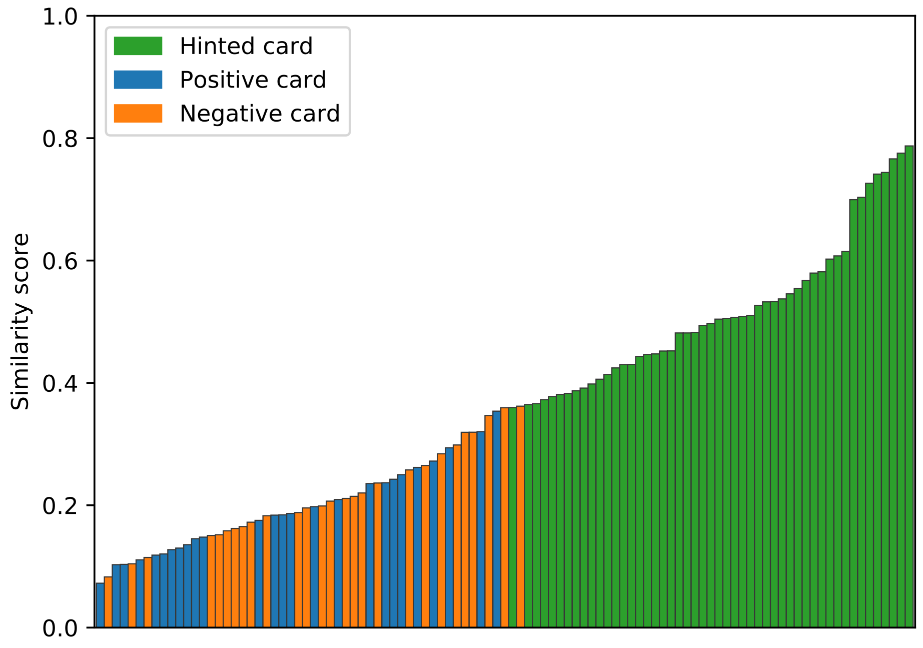


Figure A.6: Similarity scores for each card clicked by players across all games for the English topN word embedding model

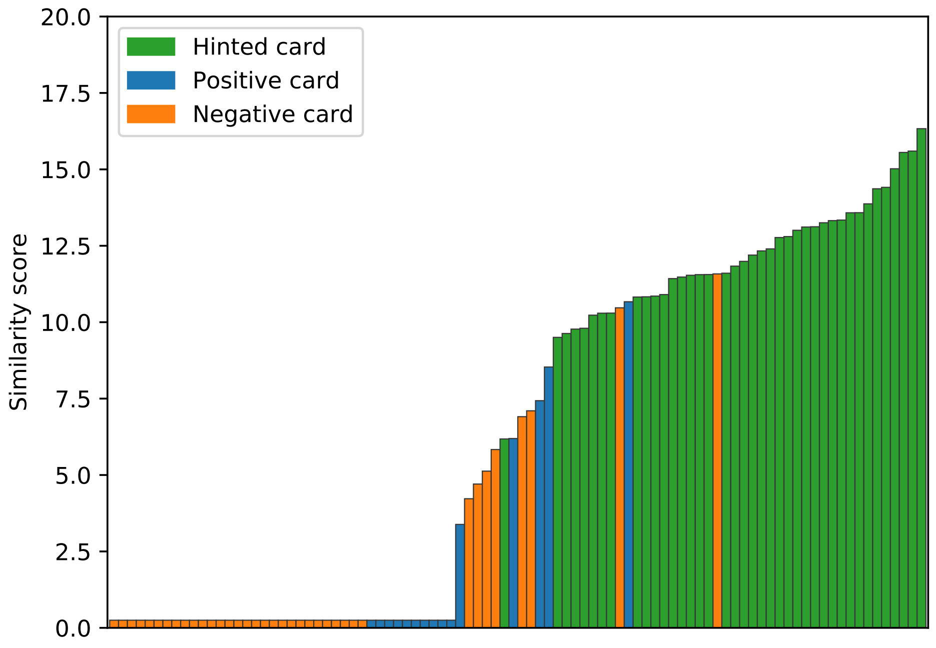


Figure A.7: Similarity scores for each card clicked by players across all games for the Czech topN dependency model

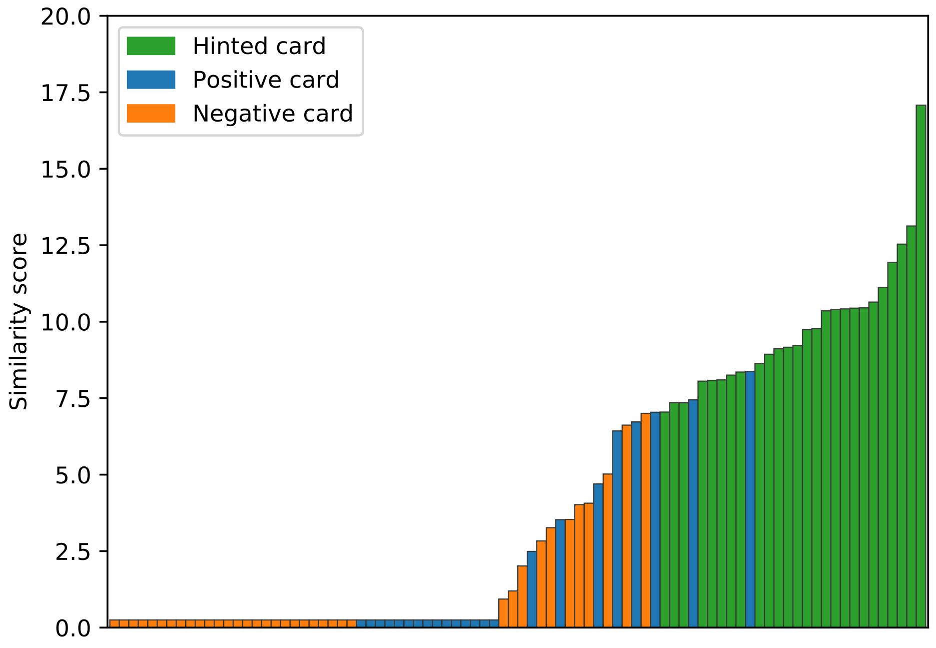


Figure A.8: Similarity scores for each card clicked by players across all games for the English topN dependency model

A.4 Distribution of decisions

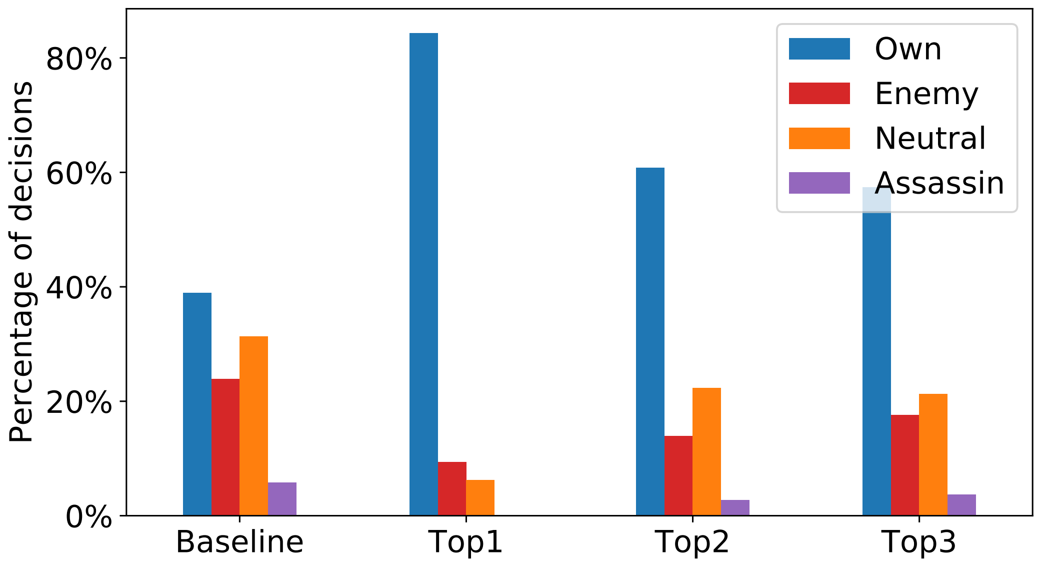


Figure A.9: Percentage of own, enemy, neutral and assassin cards clicked by players across all games played with the word embeddings model with Top1, Top2 and Top3 aggregation methods for Czech

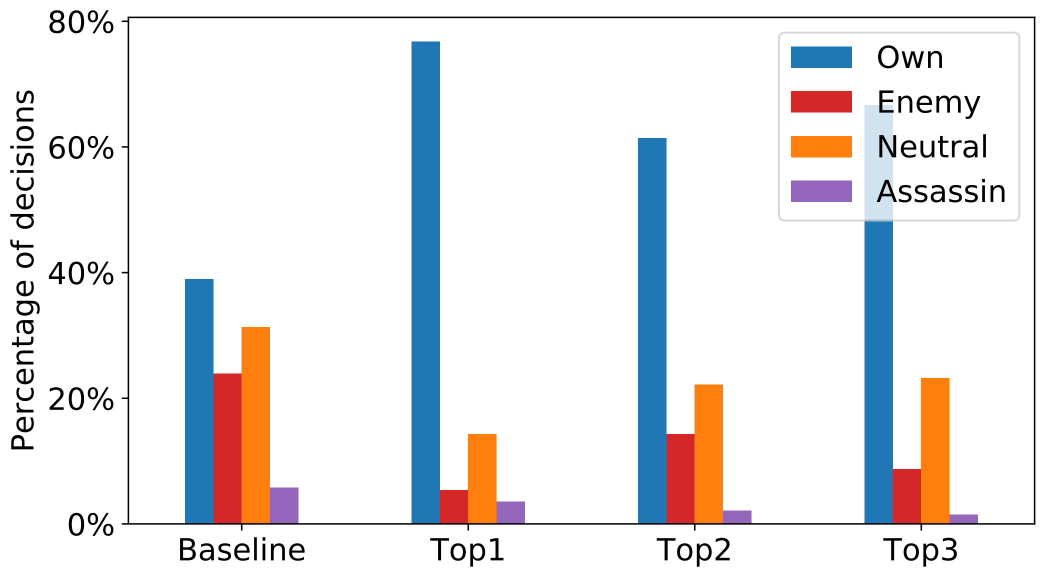


Figure A.10: Percentage of own, enemy, neutral and assassin cards clicked by players across all games played with the word embeddings model with Top1, Top2 and Top3 aggregation methods for English

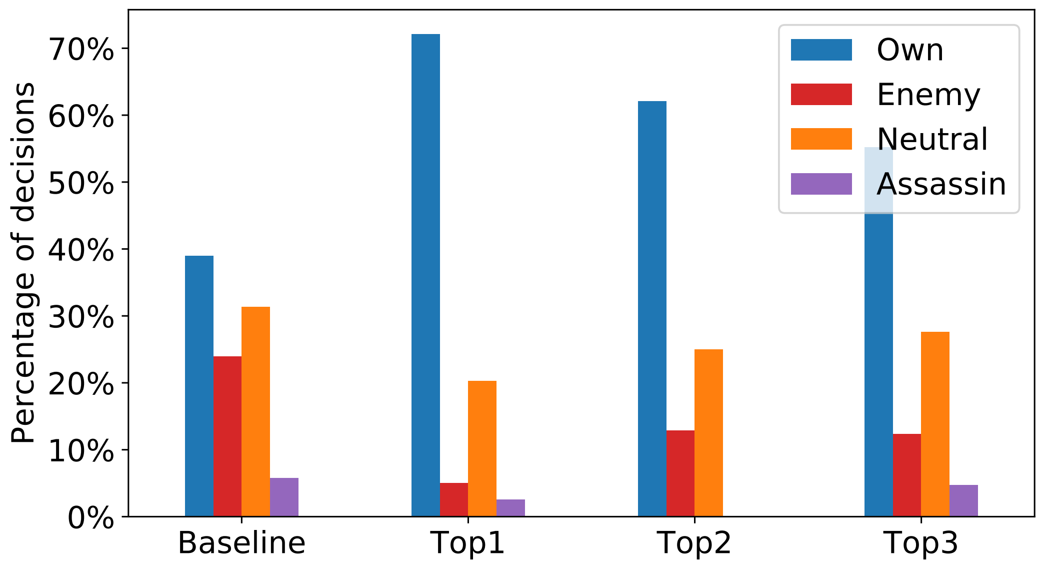


Figure A.11: Percentage of own, enemy, neutral and assassin cards clicked by players across all games played with the dependency-level collocations model with Top1, Top2 and Top3 aggregation methods for Czech

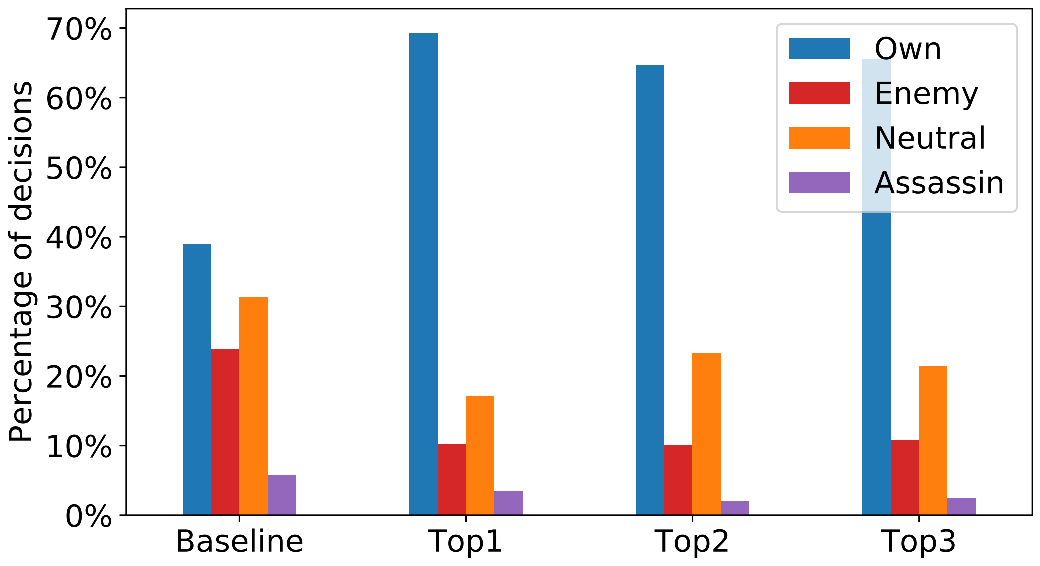


Figure A.12: Percentage of own, enemy, neutral and assassin cards clicked by players across all games played with the dependency-level collocations model with Top1, Top2 and Top3 aggregation methods for English

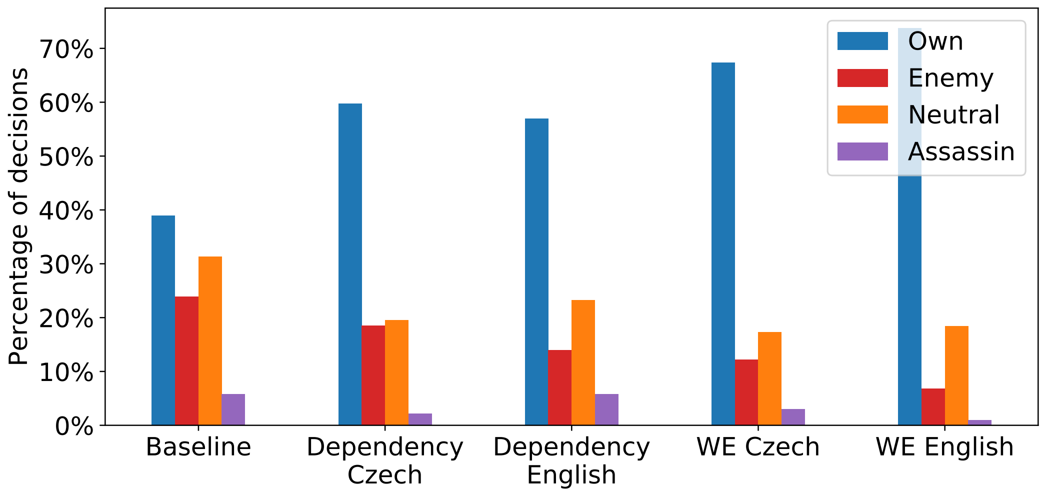


Figure A.13: Percentage of own, enemy, neutral and assassin cards clicked by players across all games played with the dependency-level collocations threshold model and the word embeddings threshold model for both Czech and English

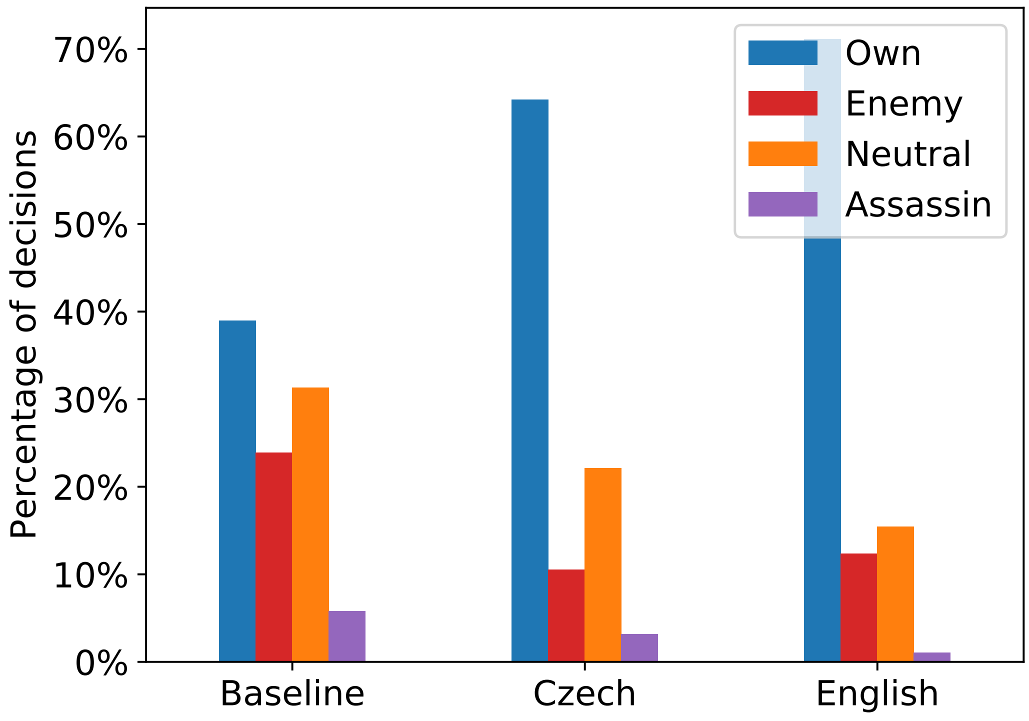


Figure A.14: Percentage of own, enemy, neutral and assassin cards clicked by players across all games played with the combination model based on the dependency-level collocations and word embeddings threshold models for both Czech and English