

RODRIGO AMORIM RUIZ

Jurisprudence Search Based on Facts Similarity Using NLP and ML Techniques

Master thesis presented to the Escola
Politécnica da Universidade de São Paulo to
obtain the degree of Master of Science.

São Paulo
2021

RODRIGO AMORIM RUIZ

Jurisprudence Search Based on Facts Similarity Using NLP and ML Techniques

Corrected Version

Master thesis presented to the Escola
Politécnica da Universidade de São Paulo to
obtain the degree of Master of Science.

Concentration field:
Computer Engineering

Advisor:
Prof. Dr. Galuber De Bona


São Paulo
2021

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 19 de Outubro de 2021

Assinatura do autor: Rodrigo Amorim Ruiz

Assinatura do orientador: 

Catálogo-na-publicação

Ruiz, Rodrigo Amorim
Jurisprudence Search Based on Facts Similarity Using NLP and ML
Techniques / R. A. Ruiz -- versão corr. -- São Paulo, 2021.
124 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.JURISPRUDÊNCIA 2.INTELIGÊNCIA ARTIFICIAL 3.APRENDIZADO DE MÁQUINA 4.LINGUAGEM NATURAL 5.REDES NEURAIS I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

ABSTRACT

Part of a lawyer's job is to understand the client's problem, to textually describe its facts and to apply the sources of law. To support a new legal case, a handful of past judgments on similar cases are typically employed by the lawyers, but finding them is currently a time-consuming procedure. To address this problem, we built a machine learning model responsible for classifying similarity between two facts' descriptions. This similarity metric measures how much (from 0 to 1) a legal decision may be used to support another. We trained different model architectures combining several state-of-the-art natural language processing and machine learning techniques using an extracted dataset from the Superior Court of Justice website of past judgments, which enabled the dynamic construction of facts' description pairs when one case cites another as a reference. The final best architecture employs TF-IDF for encoding and reducing dimensionality of our input documents, a Siamese Neural Network (SNN) with a Multilayer Perceptron (MLP) for feature extraction and a final layer, another MLP, responsible for concatenating and classifying the features into the similarity metric, achieving 85.98% accuracy, 83.89% precision and 89.06% recall. Such a model would enable the lawyer to compare a case facts description with several judgments of the jurisprudence and start their search on the most similar ones.

Keywords: Jurisprudence. Artificial Intelligence. Machine Learning. Neural Network. Deep Learning. Natural Language Processing. Transfer Learning. Bag-of-words. TF-IDF. Word Embedding. Word2Vec. GloVe. FastText. Naive Bayes. Cosine Similarity. Logistic Regression. Multilayer Perceptron. Long Short-Term Memory. Transformer. Siamese Neural Network.

RESUMO

Parte do trabalho de um advogado é entender o problema do cliente, descrever textualmente seus fatos e aplicar as fontes da lei. Para apoiar um novo processo legal, uma série de julgamentos anteriores em casos semelhantes são normalmente empregados pelos advogados, mas encontrá-los é atualmente um procedimento que demanda tempo. Para resolver esse problema, construímos um modelo de aprendizado de máquina responsável por classificar a similaridade entre as descrições de dois fatos. Essa métrica de similaridade mede quanto (de 0 a 1) uma decisão legal pode ser usada para apoiar outra. Treinamos diferentes arquiteturas combinando várias técnicas de processamento de linguagem natural e aprendizado de máquina do estado da arte usando um conjunto de dados extraído do site do Superior Tribunal de Justiça de julgamentos anteriores, o que possibilitou a construção dinâmica de pares de descrição de fatos quando um caso cita outro como referência. A melhor arquitetura final emprega TF-IDF para codificar e reduzir a dimensionalidade dos documentos de entrada, uma Rede Neural Siamesa (SNN) com um Multilayer Perceptron (MLP) para extração de "features" e uma camada final, outro MLP, responsável por concatenar e classificar essas "features" na métrica de similaridade, alcançando 85,98% de acurácia, 83,89% de precisão e 89,06% de sensibilidade. Tal modelo permitiria ao advogado comparar a descrição dos fatos de um caso com vários julgamentos da jurisprudência e iniciar sua busca pelos mais semelhantes.

Keywords: Jurisprudência. Inteligência Artificial. Aprendizado de Máquina. Rede neural. Aprendizado Profundo. Processamento Natural de Linguagem. Transferência de Aprendizado. Bag-of-words. Term frequency-inverse document frequency. Word Embedding. Word2Vec. GloVe. FastText. Naive Bayes. Similaridade de cosseno. Regressão Logística. Multilayer Perceptron. Long Short-Term Memory. Transformer. Rede Neural Siamesa.

LIST OF FIGURES

1	Training / validation / text split.	28
2	Building document pairs dataset.	29
3	Number of words per case distribution.	30
4	Number of references per case distribution.	31
5	Text length per case distribution.	32
6	Top unigrams with stopwords per case distribution.	32
7	Top unigrams without stopwords per case distribution.	33
8	Top bigrams with stopwords per case distribution.	33
9	Top bigrams without stopwords per case distribution.	33
10	Bag-of-words.	34
11	Sentence processing with word embedding.	36
12	Word2vec.	37
13	Multilayer Perceptron with 3 inputs, 2 hidden layers of size 4 and output layer of size 1.	42
14	Recurrent Neural Network representation - full representation to the left and shorter representation to the right.	42
15	LSTM cell structure.	43
16	Transformer architecture. Copied from (VASWANI et al., 2017).	45
17	Scaled Dot-Product Attention. Copied from (VASWANI et al., 2017).	46
18	Multi-head attention. Copied from (VASWANI et al., 2017).	47
19	Siamese network structure.	48
20	Architecture combinations.	49
21	Accuracy, precision and recall for BERT + SNN + Output MLP.	53

22	Validation confusion matrix for BERT + SNN + Output MLP.	54
23	Confusion matrix.	56
24	Final architecture.	60
25	STJ website search by "acórdão".	68
26	Report section from a case's PDF.	69
27	Reference from a case's PDF.	70
28	Accuracy, precision and recall for BoW + NB.	71
29	Validation confusion matrix for BoW + NB.	72
30	Validation confusion matrix for BoW + CS.	73
31	Accuracy, precision and recall for BoW + LR.	74
32	Validation confusion matrix for BoW + LR.	75
33	Accuracy, precision and recall for BoW + MLP.	76
34	Validation confusion matrix for BoW + MLP.	77
35	Accuracy, precision and recall for TF-IDF + NB.	78
36	Validation confusion matrix for TF-IDF + NB.	79
37	Validation confusion matrix for TF-IDF + CS.	80
38	Accuracy, precision and recall for TF-IDF + LR.	81
39	Validation confusion matrix for TF-IDF + LR.	82
40	Accuracy, precision and recall for TF-IDF + MLP.	83
41	Validation confusion matrix for TF-IDF + MLP.	84
42	Accuracy, precision and recall for BoW + SNN + MLP + Output MLP. . .	85
43	Validation confusion matrix for BoW + SNN + MLP + Output MLP. . . .	86
44	Accuracy, precision and recall for BoW + SNN + MLP + Output Manhattan.	87
45	Validation confusion matrix for BoW + SNN + MLP + Output Manhattan.	88
46	Accuracy, precision and recall for TF-IDF + SNN + MLP + Output MLP. .	89

47	Validation confusion matrix for TF-IDF + SNN + MLP + Output MLP. . .	90
48	Accuracy, precision and recall for TF-IDF + SNN + MLP + Output Manhattan.	91
49	Validation confusion matrix for TF-IDF + SNN + MLP + Output Manhattan.	92
50	Accuracy, precision and recall for WE + SNN + LSTM + Output MLP. . .	93
51	Validation confusion matrix for WE + SNN + LSTM + Output MLP. . . .	94
52	Accuracy, precision and recall for WE + SNN + LSTM + Output Manhattan.	95
53	Validation confusion matrix for WE + SNN + LSTM + Output Manhattan.	96
54	Accuracy, precision and recall for WE + SNN + TN + Output MLP. . . .	97
55	Validation confusion matrix for WE + SNN + TN + Output MLP.	98
56	Accuracy, precision and recall for WE + SNN + TN + Output Manhattan.	99
57	Validation confusion matrix for WE + SNN + TN + Output Manhattan. . .	100
58	Accuracy, precision and recall for Word2Vec + SNN + LSTM + Output MLP.	101
59	Validation confusion matrix for Word2Vec + SNN + LSTM + Output MLP.	102
60	Accuracy, precision and recall for Word2Vec + SNN + LSTM + Output Manhattan.	103
61	Validation confusion matrix for Word2Vec + SNN + LSTM + Output Manhattan.	104
62	Accuracy, precision and recall for Word2Vec + SNN + TN + Output MLP.	105
63	Validation confusion matrix for Word2Vec + SNN + TN + Output MLP. .	106
64	Accuracy, precision and recall for Word2Vec + SNN + TN + Output Manhattan.	107
65	Validation confusion matrix for Word2Vec + SNN + TN + Output Manhattan.	108
66	Accuracy, precision and recall for Glove + SNN + LSTM + Output MLP. .	109
67	Validation confusion matrix for Glove + SNN + LSTM + Output MLP. . .	110

68	Accuracy, precision and recall for Glove + SNN + LSTM + Output Manhattan.	111
69	Validation confusion matrix for Glove + SNN + LSTM + Output Manhattan.	112
70	Accuracy, precision and recall for Glove + SNN + TN + Output MLP. . .	113
71	Validation confusion matrix for Glove + SNN + TN + Output MLP. . . .	114
72	Accuracy, precision and recall for Glove + SNN + TN + Output Manhattan.	115
73	Validation confusion matrix for Glove + SNN + TN + Output Manhattan.	116
74	Accuracy, precision and recall for FastText + SNN + LSTM + Output MLP.	117
75	Validation confusion matrix for FastText + SNN + LSTM + Output MLP. .	118
76	Accuracy, precision and recall for FastText + SNN + LSTM + Output Manhattan.	119
77	Validation confusion matrix for FastText + SNN + LSTM + Output Manhattan.	120
78	Accuracy, precision and recall for FastText + SNN + TN + Output MLP. .	121
79	Validation confusion matrix for FastText + SNN + TN + Output MLP. . .	122
80	Accuracy, precision and recall for FastText + SNN + TN + Output Manhattan.	123
81	Validation confusion matrix for FastText + SNN + TN + Output Manhattan.	124

LIST OF TABLES

1	STJ results on validation dataset	57
2	Results for the final model	59

LIST OF ABBREVIATIONS

AUC	Area under the curve
BoW	Bag-of-Words
CS	Cosine Similarity
FFNN	Feed-forward neural network
LR	Logistic Regression
LSTM	Long Short-Term Memory
MLP	Multilayer Perceptron
NB	Naive Bayes
NLP	Natural Language Processing
RNN	Recurrent Neural Network
SNN	Siamese Neural Network
TF-IDF	Term frequency-inverse document frequency
TJGO	Tribunal de Justiça do Estado de Goiás (Goiás State Court of Justice)
WE	Word Embedding

CONTENTS

1	Introduction	15
1.1	Objective	16
1.2	Work overview	17
1.3	Paper organization	20
2	Related work	21
3	Methods	26
3.1	Dataset	26
3.2	Document representations	34
3.2.1	One vector per document	34
3.2.1.1	Bag-of-Words	34
3.2.1.2	Term frequency-inverse document frequency	35
3.2.2	One vector per word	35
3.2.2.1	Word Embedding	36
3.2.2.2	Word2vec	36
3.2.2.3	GloVe	38
3.2.2.4	FastText	38
3.3	Models	39
3.3.1	Naive Bayes	39
3.3.2	Cosine Similarity	40
3.3.3	Logistic Regression	40
3.3.4	Multilayer Perceptron	41

3.3.5	Long Short-term Memory	41
3.3.6	Transformer	44
3.3.7	Siamese Neural Network	47
3.4	Architectures	48
4	Experiments and results	55
5	Conclusion	62
	References	64
	Appendix A - STJ search and extraction	67
	Appendix B - STJ results	71

1 INTRODUCTION

Brazilian law is considered to be a fusion between “civil law” (Roman-Germanic) and “common law” (North American), since its constitution was inherited from the American system, which allows formalizing the theory of judge-made law. When the law does not fit a particular case, it needs to be interpreted. When that happens from judges, judges of a court of law or ministers of justice, a decision is born, which becomes part of the *jurisprudence* – a set of decisions, applications and interpretations of laws, or the study of laws.

In the Brazilian legal environment, jurisprudence is used as a second source to sustain new cases worked on by operators of the law. Nevertheless, the research in jurisprudence consumes substantial lawyer’s time when reading past judgments for identifying the relevant ones, and those with a similar description of facts. This is because such a search is usually performed through keywords, thus returning a number of irrelevant results. A lawyer must analyze about five to ten decisions in order to find one that is similar to the current case and that can be used as a source.

In a small survey performed by us with 12 lawyers, we found that on average, for each new case, the following applies:

- Jurisprudence search time: 100.83 minutes
- Number of past decisions necessary to sustain the current case: 3.92 decisions
- Number of decisions that must be read before filtering the correct ones to sustain the case: 25.17 decisions

The search time and the number of decisions read by the lawyer can be reduced with our work, by creating a model that identifies only the decisions most likely to be similar to the new case.

In addition, our survey showed that 91.7% of our lawyers feel positive about the use of artificial intelligence to facilitate the jurisprudence search, while the rest felt indifferent about it. This implies that if we manage to accomplish satisfying results, lawyers would actually use the final solution.

A preliminary study published in the WPGEC 2019 workshop (RUIZ; BONA, 2019) was also conducted previously to this work, where we (same authors as this current work) tested four different models, with three of them based on bag-of-words, using cosine similarity, a naive Bayes classifier or a Multilayer Perceptron, and the fourth model being a Siamese Neural Network with word embedding and Long Short-Term Memory layers. All models were trained and tested with data obtained from the Court of Justice of Goiás' system (TJGO), a smaller dataset of 115k fixed pairs of cases with 15k of those being considered similar, having the best model achieve 94.2% accuracy and 76.2% precision, with 97% precision on the 100 most similar samples. Though we did not achieve higher accuracy on our most recent study, the TJGO dataset was heavily skewed towards negative samples (non-similar), explaining the lower precision. In addition, we also employ a much bigger dataset in our current study, providing higher reliability as will be mentioned later.

1.1 Objective

The objective of this work is to reduce the jurisprudence search time for a given new case and to provide a starting point for other researchers when choosing an architecture for dealing with this kind of problem or when looking for a large legal cases dataset in Portuguese, which will be presented as one of our work's contributions. By exploring the most recent and successful Natural Language Processing (NLP) and Machine Learning (ML) techniques, we intend to find the most similar cases in a database of past judgments.

By training a classifier, one can predict the similarity between the new case and each case in the jurisprudence, based on their description of facts. The lawyer could thus focus his search on the most similar cases, instead of reading through many unwanted references.

NLP and ML techniques have been increasingly used in many data intensive tasks including in the legal environment (ZHONG et al., 2020). That is partly because of the increase in available data and part because now we have a better understanding of how to make a computer interpret text and make conclusions out of it, thus justifying our approach to solving the proposed problem.

1.2 Work overview

If two separate legal cases presented the exact same circumstances (not considering when and with whom they happened), which are described in the case by their facts, they should, theoretically, receive the same judgment, given the law has not changed between one and the other (maybe one of those cases happens 50 years after the other). Furthermore, despite the justice system being somewhat subjective, it is not meant to be, it is meant to pass a judgment solely based on the case's facts. For this reason, in Brazil, judges sometimes accept past decisions on previous cases as basis to judge a current similar case (similar in the sense that they share similarity in their facts). So when lawyers are building their current case, they look for past decisions that have similar facts to their current case in the hopes that a judge will evaluate their case in the same way.

This process is done manually by the lawyer (or its' assistants) by searching via word comparison on one (or more) of the many legal databases. Then they read the facts from the old cases until they find a few that share similarities to their case.

Thus in order to achieve our objective of reducing the time of that search process, we use NLP and ML techniques to produce a resulting model that compares two pieces of text and classify them as similar or not. "Similar" meaning they can be used as a jurisprudence reference to each other (not that they have the same text). Those two pieces of text being the facts' description of the current case and each past decision from the database that it is being compared to.

With that, one can compare each old case to the current one and filter only the most similar ones for the lawyer to assess, but this time, with much more assertiveness than if they had to manually search for a case just by regular word filters.

There are many ways to go about building this model, with several NLP techniques such as word embedding (COLLOBERT; WESTON, 2008), LSTM (HOCHREITER; SCHMIDHUBER, 1997), Transformer (VASWANI et al., 2017) or Siamese Neural Network (BROMLEY et al., 1994), some using ML and others not. Some of those techniques were chosen based on current state of the art while others were chosen as a baseline. Because we do not have enough time or computer power to test them all,

this study focused on a selection of techniques described in Chapter 3 resulting in a total of 28 different architectures that will be presented in Section 3.4.

Those architectures were based on one of seven different types of models:

- Naive Bayes (NB)
- Cosine Similarity (CS)
- Logistic Regression (LR)
- Multilayer Perceptron (MLP)
- Siamese Neural Network (SNN) + MLP
- SNN + Long Short-Term Memory (LSTM)
- SNN + Transformer (TN)

The first four types of models were chosen mainly as a point of comparison, since we expected the last two to achieve better results (by being more complex and better capable of abstracting the data), though later we will see this was not exactly the case. The fifth type was an intermediary approach between the other, using the SNN technique, but still using a one vector per document approach that will be explained in Section 3.2.1.

The SNN + LSTM based models with a few word embedding methods (3.2.2) were chosen due to the previous success of these techniques in the domain of NLP. Siamese networks are responsible for some of the best results in comparing two pieces of text, being the state of the art in sentence semantic similarity tasks (MUELLER; THYAGARAJAN, 2016). Word embeddings (PENNINGTON; SOCHER; MANNING, 2014) employ a vector space to represent words, usually yielding better results than one-hot encoding in NLP tasks, as word semantic similarity measuring (MIKOLOV et al., 2013). LSTM has consistently shown state-of-the-art results in a variety of NLP tasks (PLANK; SØGAARD; GOLDBERG, 2016) (WANG et al., 2016) (CHEN et al., 2017) and was first introduced with the objective of storing information of extended sequence intervals (HOCHREITER; SCHMIDHUBER, 1997).

For similar reasons to the SNN + LSTM based models, SNN + TN was also chosen as a second option, also showing very good results in many NLP benchmarks (VASWANI et al., 2017). Though the main paper focuses on text to text language translation, part of the Transformer can be used as a text encoding with the SNN, so our two pieces of text can be compared. This will be better explained in Section 3.3.6.

The results will be presented and further elaborated on in Chapter 4, but the best model was the combination of TF-IDF + SNN + MLP + Output MLP, most likely due to the nature of our problem, dealing with large documents that are usually hard to handle for LSTM based architectures and are complex enough to make it difficult for simpler models such as NB to achieve good results. The SNN architecture also proved ideal for our solution, since the two pieces of text should be interpreted in the same way (meaning there are no two different types of inputs, only two inputs of the same type, where order, for instance, does not matter). Finally, the MLP showed very good flexibility for further processing the extracted features of TF-IDF and for constructing the similarity function responsible for comparing those features.

In order to train these supervised learning models, a labeled dataset was required. Since there is no publicly available option, to the best of our knowledge, one was constructed from records of past decisions of the Superior Court of Justice ("Supremo Tribunal de Justiça" – STJ). Each decision consists of a whole assembled case, which contains a description of the case's facts and references to the jurisprudence used as a source. With these references, it was possible to build a dataset with pairs of cases whose facts' descriptions were similar (or not) - as briefly explained before, by similar we do not mean they necessarily have similar words in their text, we only mean one can be used as a reference to sustain the other. Section 3.1 will detail this extraction and generation process.

Besides the results of this work, two additional contributions are the extracted dataset, which can be accessed upon request, and the entire code for the data extraction and training / evaluation process of all the models, which is available in a public repository.

1.3 Paper organization

The rest of the text is organized as follows: In Section 2 we present the related work; Section 3 describes how the dataset is constructed and the NLP techniques that will be used to build the models; Section 4 shows the results achieved by the 28 different models, along with the final best model expanded from one of the 28; and Section 5 brings some final considerations about our conclusions and possible future work.

2 RELATED WORK

A common machine learning classification problem receives an input and classifies it into one of a fixed number of categories. Our work can be seen as a classification problem, as we intend to classify two pieces of text as being similar or not. However, it is not a classification problem with a single, atomic entry, as our input is formed by a pair of facts' descriptions, corresponding to the two cases whose similarity is to be determined.

Machine learning techniques have been employed to deal with these multiple entry classification problems in various tasks, such as face recognition (SCHROFF; KALENICHENKO; PHILBIN, 2015) or determination of semantic similarity in Quora questions (COHEN, 2017), but most applications are restricted to small texts, containing few sentences, whereas the facts' description of a legal case can be quite extensive, sometimes containing a few pages, not to mention the issue of finding a large enough legal dataset to achieve top performance on more complex models.

Maheshwary and Misra (2018) try to solve the problem of recommending appropriate jobs to candidates based on the job description and their curriculum. They propose a Siamese convolutional neural network (CNN) (LECUN et al., 1999) capable of determining the similarity between the two. As input, first the resumes and job descriptions are converted into vectors using Doc2Vec (LE; MIKLOV, 2014) which are fed to the pair of identical CNNs of the Siamese network. The output of those CNNs is then concatenated and processed by a fully connected layer, generating the final similarity metric. Their problem resembles ours by having a text pair as input and trying to measure the similarity between it, though comparing two different distributions of text (resumes and job descriptions). They employ a dataset of 1314 resumes and 3809 job descriptions, totalizing 5,005,026 pairs, annotated in a semi-supervised manner which is not specified. Despite having a big pairs dataset, the text variation is small when it comes to complex machine learning models, since there are very few different texts generating the pairs.

Shih et al. (2017) investigate the use of Siamese LSTM networks for generating

document representations for text classification. They test their model on two different datasets: IMDB reviews (MAAS et al., 2011) and 20Newsgroups (LANG, 1995). Starting by building pairs of texts and classifying each as similar if they were in the same category (positive or negative sentiment for IMBD reviews and one of 20 different classes for 20Newsgroups) and different otherwise. Those pairs are processed by an embedding layer (Word2vec, Skip Gram or GloVe) with the result being fed to the Siamese LSTM network responsible for generating final document encodings (vectors), which were then used to calculate their Euclidean distance. The distance was used as a negative similarity function, measuring how related the two documents are (smaller distance meaning more related). This training process ends up generating a sub-network (one of the LSTMs from the Siamese network) capable of generating a document representation, which was in turn fed to a three-layer deep neural network predicting the probability distribution over the different classes. They end up using a similar method to ours (Siamese LSTM network), but not with the final objective of comparing how similar two documents were and rather just generating encodings to be used in the text classification problem. Another key difference from our proposed work is that they use datasets with well-defined categories to generate the pairs, which might facilitate the encoding process, in contrast to our legal texts having many (not defined) categories.

In the legal domain, Bueno et al. (1999) tackle the same problem as this work, the search in jurisprudence for similar cases. They split the problem into three parts: representation of legal cases, automatic information extraction and similarity based retrieval process. The representation is constructed via a set of indexes such as publication date, indicative expressions and case category. Those indexes were determined by a domain expert in accordance with their importance for comparing with a new case. To support the retrieval process and automatic extraction of those indexes, a controlled vocabulary and a juridical thesaurus were manually developed by juridical professionals. For the automatic extraction, they used different keyword references for each index, such as “DATA” (“date” in Portuguese) for identifying the date or words / expressions from the built vocabulary to identify the category. For the retrieval process, the user of the system describes the new case in natural language and specifies other limiting parameters such as period or resource type. From that, relevant information is auto-

matically extracted using the controlled vocabulary and juridical thesaurus for the new case (in the same way the automatic extraction for previous cases was conducted). Then they use the nearest neighbor approach to compare the similarity between cases in the database and the new described case using the extracted information, presenting to the user only the top 10 most similar cases. Overall they use a manual / professional knowledge base system approach rather than pure machine learning. Nevertheless, no quantitative results from a well-defined dataset are provided, which could be used as a benchmark.

Yet another index approach in the legal domain, Rissland, Skalak and Friedman (1993) discuss the use of manually selected indexes and heuristics on cases to be used as precedents for argument generation. They mention five indexes: types—citation, prototypical story, factor, family resemblance, and legal theory. Which are meant to be used in conjunction with one another rather than separate filters as the previous work from Bueno et al. (1999). The use of those indexes also acts as filters for retrieved cases, but not individually, they are paired with a set of indexing strategies with cases being nodes in a graph. The connections are used to determine how each of the defined heuristics will filter the resulting set of cases. They also briefly mention the data used came from the U.S. Federal statute that governs the approval of bankruptcy plans, but they do not publicly provide that data for access in their work. Moreover, no accuracy results are provided whatsoever.

Trappey, Trappey and Liu (2020) try to analyze and identify judgment documents of US trademark (TM) litigation cases as precedents of a given target case. They use a combination of Word2Vec, Cosine Similarity, Latent Dirichlet Allocation with clustering to create a set of clusters in which the target case can be placed (determining a percentage of likelihood for each cluster). They use a total of 4835 TM litigation documents to train the entire model and separate 42 target cases with 5 cluster cases to be manually evaluated during testing, generating a total of 210 pairs for comparison. In their final results, they achieved 83.8% accuracy in matching the clusters of their original judgments. Albeit similar to our work, they employ a much smaller and restricted dataset in comparison. They also assume the generated clusters are sufficiently representative of case similarity, which might not be the case. Though their approach is understandable since they also do not have a manually labeled dataset of similar pairs.

Liu and Yang (2018) also work on a similar case retrieval process, but more specific than our work. We try to predict similarity in a broader sense, where cases can be used as a reference to support each other, while they literally mean similar, since their work attempts to build a question and answer system where a user can ask questions about a criminal case, with the system gathering information about its facts and then searching for the most similar one. Their implementation happens in three steps: 1) a question processing module, responsible for identifying the most relevant semantic slots pertaining to the user's questions; 2) the information retrieval module, which queries the legal database for each semantic slot previously identified in a weighted manner, looking for the most similar cases. 3) the answering module, responsible for generating answers from the retrieved data that responds to the intended question formulated by the user. The similarity comparison does not happen between two pieces of text, but rather between the weighted semantic slots allocation and the case library with pre-established semantic values of their own. The authors claim their system has a high accuracy of information retrieval, but they do not specify a number, nor do they provide a public dataset used in their system.

Al-Kofahi et al. (2001) approach the problem of prior case retrieval from the appellate chain of the current case using court opinions as input. They start by processing those opinions to extract information about party names, courts, dates, docket numbers and history language. With that, their system generates appropriate queries to a citator database, retrieving possible prior cases. Finally, they filter those possible priors with a machine learning system (Support Vector Machine), identifying the actual prior cases linked to the current case. The model uses assembled features from the extracted information as input. Though also in the legal domain of information retrieval, their work does not attempt to process large bodies of text, but rather small titles and court opinions.

The COLIEE-2018¹ is a competition with a legal case retrieval task (Task 1), which involves finding supporting cases for a new case, very similar to our problem, but using a database from the Federal Court of Canada case laws. The training data consists of 285 query cases to be analyzed (to find references) with each having 200 candidate cases, with some of those being relevant (working as references) to the analyzed case,

¹<https://sites.ualberta.ca/~miyoung2/COLIEE2018/>

i.e., they used a much smaller database than ours. Nevertheless, the best results for this task were achieved by Tran, Nguyen and Nguyen (2018) (precision: 0.6763, recall: 0.6343, f-score: 0.6546), using a combination of lexical matching and encoded summarization (encoding the entire document into a vector space embedding properties of summarization). The former tries to estimate the lexical matching between a query and a candidate case in different types of n-grams, skip-grams and longest common subsequence to measure various degrees of lexical similarity by employing matching formulas from the ROUGE evaluation (LIN, 2004). The latter is based on Tran, Nguyen and Satoh (2018) to extract catchphrase and generate encoded summaries for the documents (cases). With those two sets of features, they use a Linear-SVM (CHANG et al., 2008) as learning algorithm to solve the optimization problem of ranking the similarity between query and candidate cases.

There are many other works addressing tasks related to text similarity, as question-answering problems (MINAEE; LIU, 2017) or detecting character-based similarity of job titles (NECULOIU; VERSTEEGH; ROTARU, 2016). However, none of them deals with a large database of long legal texts without well-defined categories, especially in Portuguese, with the exception of Ruiz and Bona (2019), mentioned before in Chapter 1.

3 METHODS

The proposed solution has two parts: a classifier model capable of determining the similarity between two facts' descriptions and the recommendation process, which uses the classifier to determine the most similar judgments.

In this section, we explain the details of the data extraction and the NLP techniques used such as document encodings and model architectures.

3.1 Dataset

Ideally, we would have access to a dataset built by lawyers containing pairs of judgments manually classified, via their facts' description, as similar or different. As a viable alternative, we decided to extract a dataset from the Superior Court of Justice¹ (STJ) which contains a broad spectrum of cases. We are currently not filtering by instance or case type, rather we are trying to extract as many cases as possible.

The extraction process started by building a web crawler responsible for downloading the PDFs of a total of 682k cases. After dropping the ones that were duplicates, had missing PDFs, missing dates, missing identifiers and missing reports (part of the text used in the machine learning algorithms), we were left with over 440k cases.

Within each case, we are interested only in the facts' description and the jurisprudence references. The former can be found in the section of the document called "Relatório" (report in Portuguese), which starts with the word "RELATÓRIO" and ends with "É o relatório" (with a few variations such as letter case). With some text processing and pattern matching, we were able to extract the report section from the PDF text. The latter (jurisprudence references) were extracted via pattern matching using a regular expression for the format *d.ddd.ddd/cc*, with *d* being a digit and *c* being a character, since those have all this same format. All references inside the report text were removed to avoid bias in the model training later on.

Examples of the search step, report section and jurisprudence reference can be

¹<http://www.stj.jus.br/>

found in Appendix A.

The report section contains more than just the facts' description and the jurisprudence references, it may also include other components of the case. In order to discard these irrelevant components, manual inspection by an expert would be required, due to the lack of structure in the text. Therefore, we assumed that the description of facts comprises all the text within the report section, even though it is not entirely true, but likely good enough.

With those at hand, a dataset was built with four main columns: the case identifier; the case's date; its' known references; and the report text for said case.

As the first contribution of our work, this dataset can be used by anyone for research purposes upon request via email² (it was not publicly uploaded to a repository due to its size – almost 13 GB).

The next step was to sort the 440k cases by date in descending order, setting aside the first 10k case for a test dataset, the next 30k for validation and the rest for training. Note that we did not pair the cases as similar or not before splitting them into test / validation / training datasets. The reasoning behind splitting this way (and not after building the pairs) and for sorting by date is the fact that in order to predict references, one should not have access to future cases, only to past ones, so all test cases are more recent than validation cases, that in turn are more recent than training cases.

To actually build the pairs we then proceed to create a generator that would output half of the pairs with their known references (references that were actually used on those cases) and the other half with random cases from the lookup data. For training, this lookup data is the same training data, but for validation, it is the training and validation dataset combined. Meaning the validation cases can reference training cases, but not the other way around. The reason for this half / half distribution is to avoid bias towards similar or not similar classifications.

This data split process can better be seen in Figure 1, with the pairs for each split of the data being built by taking one sample from the start of the respective blue arrow and one from the end of that arrow. For example, to build a pair for the validation dataset used on the model, we take one case from the 30k cases in the validation data

²rodriguiz@usp.br

split and one from the validation lookup data, consisted of the validation and training data combined, i.e., 430k cases.

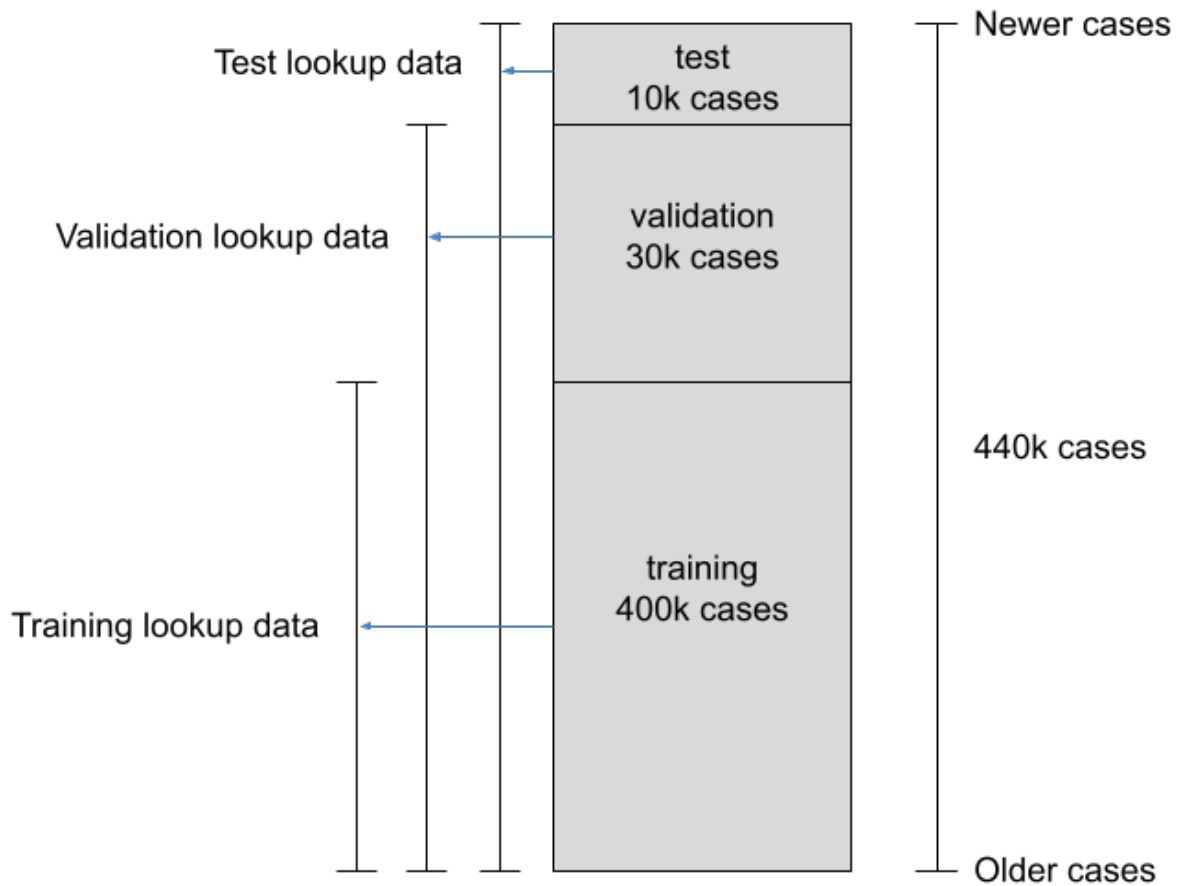


Figure 1: Training / validation / text split.

The validation dataset of pairs was randomly constructed in the same way as the training pairs, the only difference is that for training, those pairs were chosen as the model was being trained ³ whilst the validation pairs were chosen randomly only once and then kept fixed throughout different epochs and models. A total of 30k pairs were generated to compose the validation dataset and those were built as explained to maintain metrics' consistency.

Each pair of facts' descriptions was labeled as similar or different based on whether one of the corresponding judgments had the other in its references (if so, then they were considered similar). This can better be explained by looking at Figure 2, where document *X* references documents *A*, *B* and *C*, and document *Y* references documents *D*, *E* and *F*. Note that, in the table in Figure 2, there are rows for *X/C*, *Y/E* and *Y/F*

³Only when necessary - models like the CS based ones do not use training data

with similarity equal to 1 (meaning they are similar). However, without inspecting the references in A, B, \dots, F , we cannot determine if the pairs Y/A or X/D are similar or not, for instance.

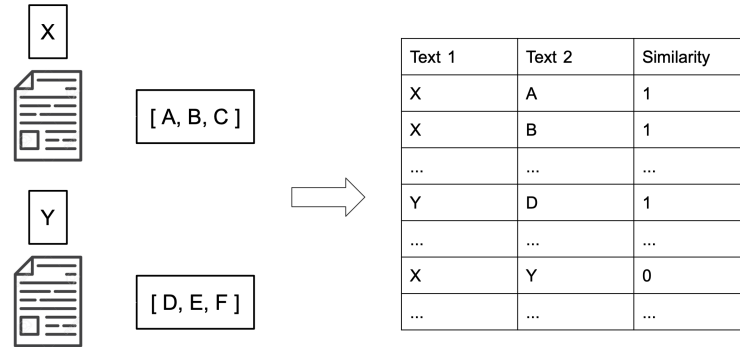


Figure 2: Building document pairs dataset.

In addition to that, we also did an exploratory analysis of the data (only for the training cases so we do not taint the results) by looking into the following aspects:

- Number of words per case distribution (Figure 3): indicating there are very few cases with more than 1000 words, meaning we can trim the report to a maximum length of 1000 words and not lose information on most of our cases.
- Number of references per case distribution (Figure 4): showing that on average, each case references 3.83 other cases, further sustaining our survey's conclusion of 3.92 past decisions necessary to sustain the current case.
- Text lengths' distribution (Figure 5): text length distribution quickly peaks its occurrences around 1000 characters and decreases at a slower rate, reaching almost zero around 7000 characters.
- Top words distribution (unigrams and bigrams) before and after removing stop-words / processing (Figures 6, 7, 8 and 9): comparing unigrams between Figures 7 and 8, and bigrams between Figures 9 and 10, with and without stopwords, we can clearly see many common words are removed by filtering stopwords; After removing the them, we can also notice that legal texts appear to be using a very specific subset of the language.

Minimum: 5
Maximum: 14917
Mean: 284.98
Median: 249
Percentile (99): 748

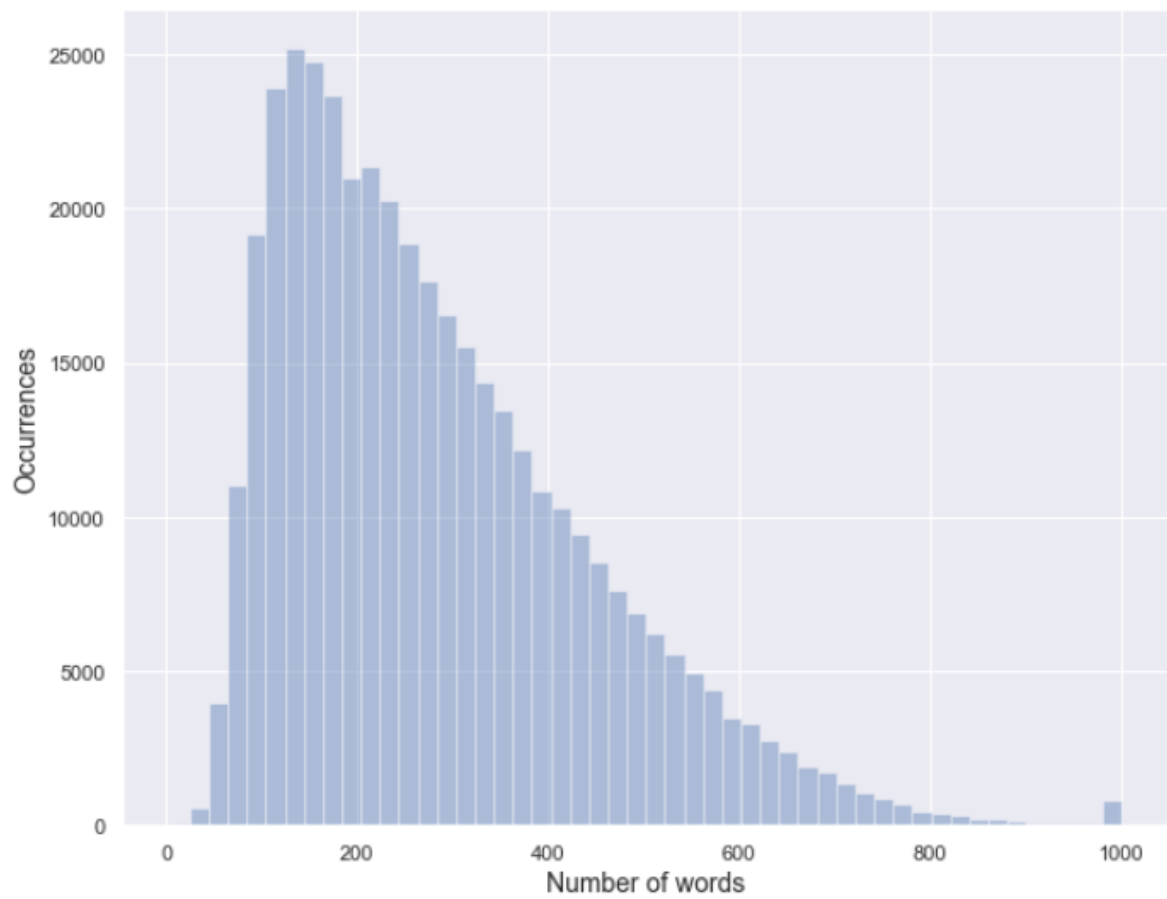


Figure 3: Number of words per case distribution.

Minimum: 0
Maximum: 120
Mean: 3.83
Median 3
Percentile (90): 9

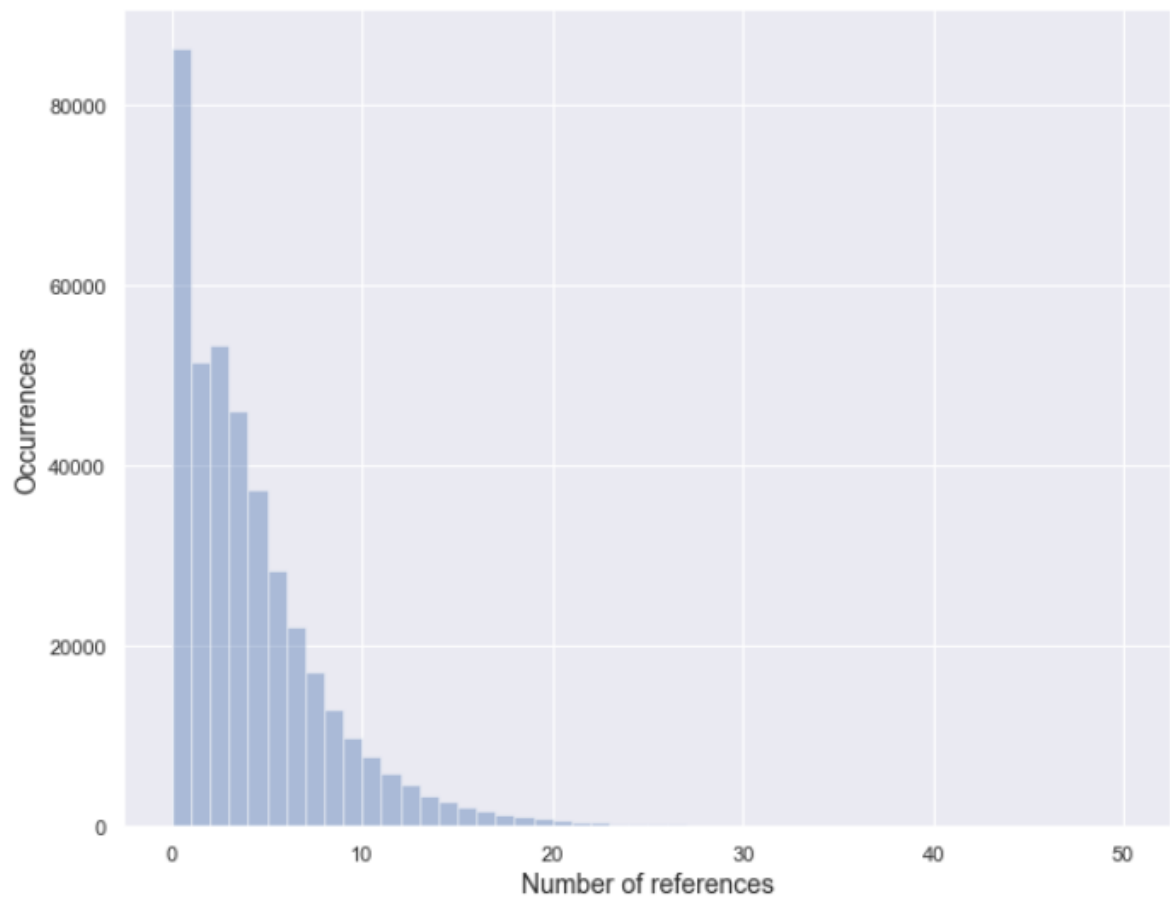


Figure 4: Number of references per case distribution.

Minimum: 41
 Maximum: 114114
 Mean: 2143.15
 Median: 1813
 Percentile (99): 5941

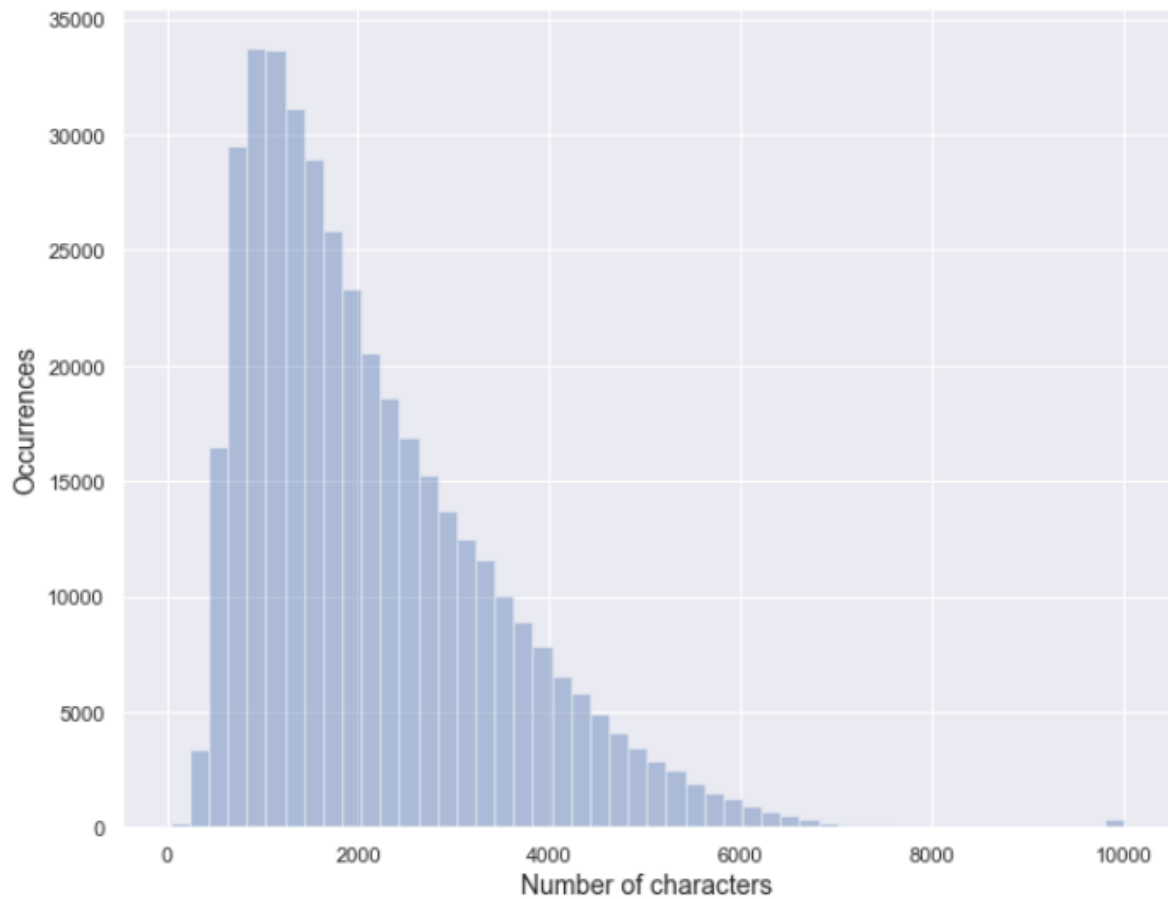


Figure 5: Text length per case distribution.

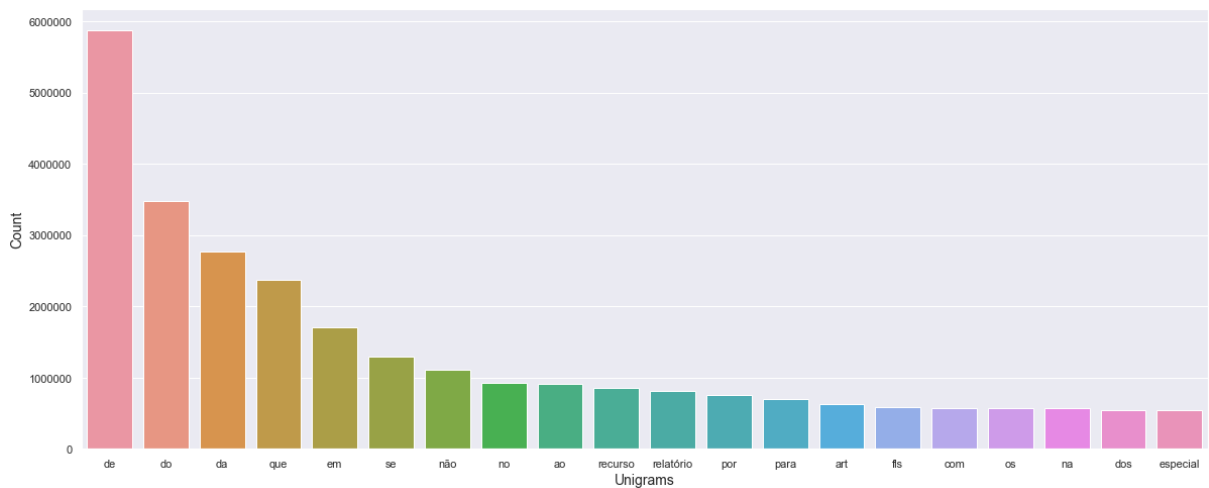


Figure 6: Top unigrams with stopwords per case distribution.

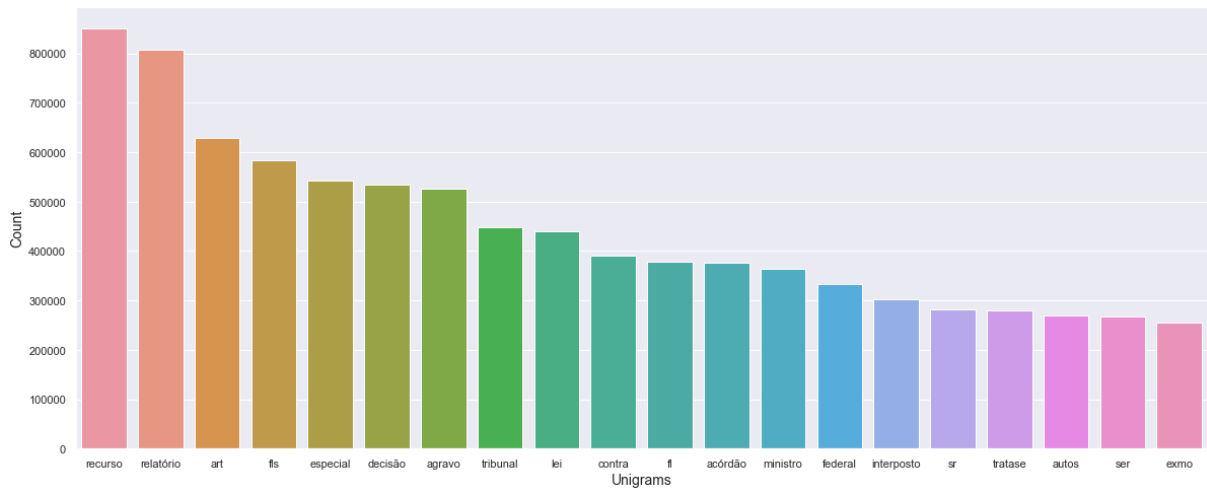


Figure 7: Top unigrams without stopwords per case distribution.

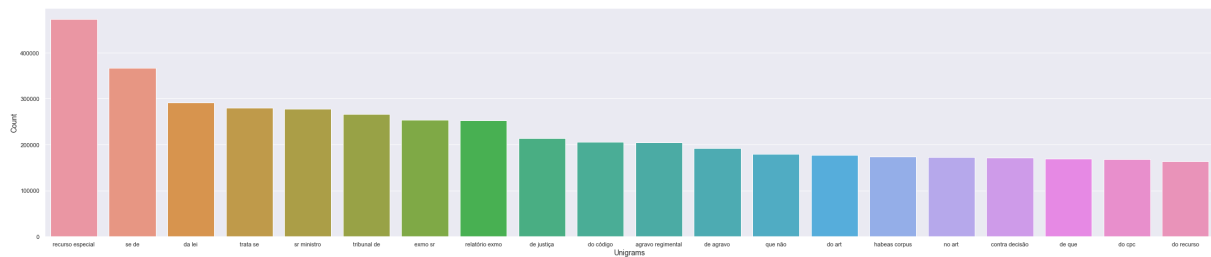


Figure 8: Top bigrams with stopwords per case distribution.

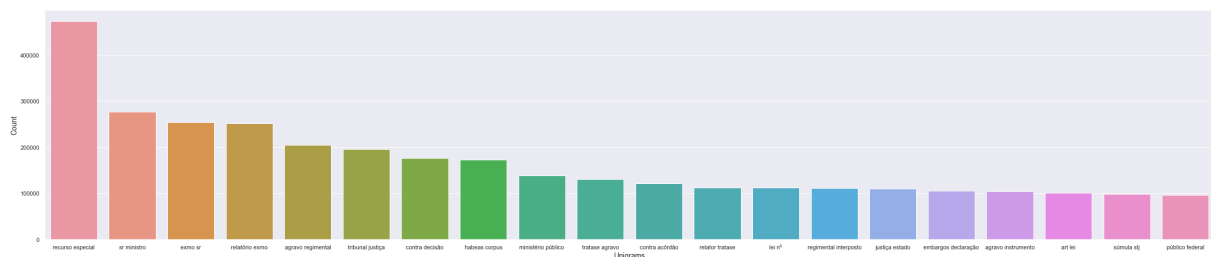


Figure 9: Top bigrams without stopwords per case distribution.

3.2 Document representations

A document can not be processed directly by a model, it first has to be converted into a set of numbers, which then the model can understand, i.e., do math operations with it.

We will be employing two different representations: one vector per document and one vector per word.

3.2.1 One vector per document

For representing an entire document as a single vector, we will be using two different techniques: Bag-of-Words (BoW) and Term frequency-inverse document frequency (TF-IDF).

3.2.1.1 Bag-of-Words

The first step of the bag-of-words method is determining a vocabulary, usually formed by a number of most frequent words, after eliminating stopwords. Once fixed the vocabulary, BoW (HARRIS, 1954) transforms a document into a vector of natural numbers by counting the occurrences of each word in the vocabulary. Figure 10 shows an example of a dataset with only two documents, each being converted into a vector using BoW.

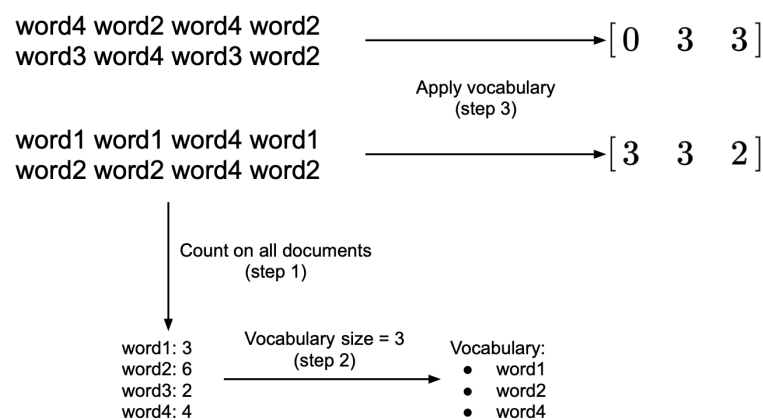


Figure 10: Bag-of-words.

3.2.1.2 Term frequency-inverse document frequency

Term frequency-inverse document frequency (TF-IDF) (RAMOS et al., 2003) is a technique similar to BoW which considers the count for each word that appears in a document, but is inversely proportional to how many time that word appears in all the other documents, thus lowering the importance of very common words such as "a" or "the".

Equation (3.1) shows the term frequency function, which represents the number of times a term (word) t appears in document d .

Equation (3.2) shows the inverse document frequency function, with N being the total number of documents and $|\{d \in D : t \in d\}|$ the number of documents in dataset D that contain a term t .

And finally, Equation (3.3) shows the function for TF-IDF which is just the multiplication of tf and idf .

Intuitively this gives us the importance of each word in a document, with such being lower for words that appear many times in all the other documents and higher for specific words that appear in the analyzed document.

There are some variations for both Equations (3.1) and (3.2), but the one mentioned above is the one we will be using.

$$tf(t, d) = \text{raw count of terms in a document} \quad (3.1)$$

$$idf(t, D) = \log \left(\frac{N}{|\{d \in D : t \in d\}|} \right) \quad (3.2)$$

$$tdidf(t, d, D) = tf(t, d) \cdot idf(t, D) \quad (3.3)$$

3.2.2 One vector per word

Representing the document as a single vector will limit the representation capacity, since meaning in word order is usually not captured, thus we can represent each word

as a vector and the entire document becomes a list of vectors, i.e., a matrix. The techniques we will be using for this form of representation will be Word embedding, Word2Vec (CBOW and Skip-gram), Glove and FastText.

In the following sections, we will be explaining each one separately.

3.2.2.1 Word Embedding

A word embedding (COLLOBERT; WESTON, 2008) can be seen as a neural network layer that converts each word into a vector of real numbers. Typically, each word is initially represented as a vector of 0's with a single 1, in what is called *one-hot encoding*. This vector has the size of the vocabulary and each word is represented by a unique vector. The embedding layer reduces the dimensionality of these vectors by multiplying them by a matrix of parameters. Formally, this matrix has size $V \times E$, V being the size of your original vector and E the output size of the reduced vector. Figure 11 shows an example of a sentence being encoded with one-hot encoding and then with word embedding.

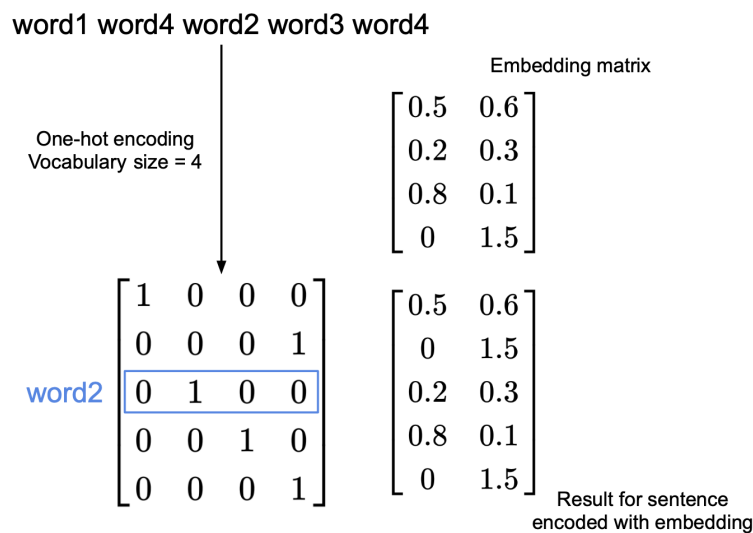


Figure 11: Sentence processing with word embedding.

3.2.2.2 Word2vec

Word2vec (MIKOLOV et al., 2013) is a way of constructing a word embedding representation and presents two variations: Skip Gram and Continuous Bag Of Words (CBOW).

The CBOW model tries to predict a target word based on a context, such as the previous word. It does it by using a shallow neural network (two layers plus the input layer) with the output layer using a softmax function, as in Figure 12, having the one-hot encoded context word as input and the one-hot encoded target word as output. This context can also be a window size around the target word, i.e., the context can be a number of previous and next words. When multiple words are used as context, after processed by the hidden layer, their resulting embeddings are averaged before passing through the softmax output layer. As an example, a window of size 2 would generate 4 words as context to be used as input and the output, the target word. Repeating this process for each word in the corpus as a target word would give us the full training dataset for the CBOW Word2vec model. After trained, we ignore the softmax (output) layer and take only the hidden layer's weights as the embedding for each word (multiplying the one-hot encoded version of each word by the hidden layer's weight matrix generates the embedding for said word).

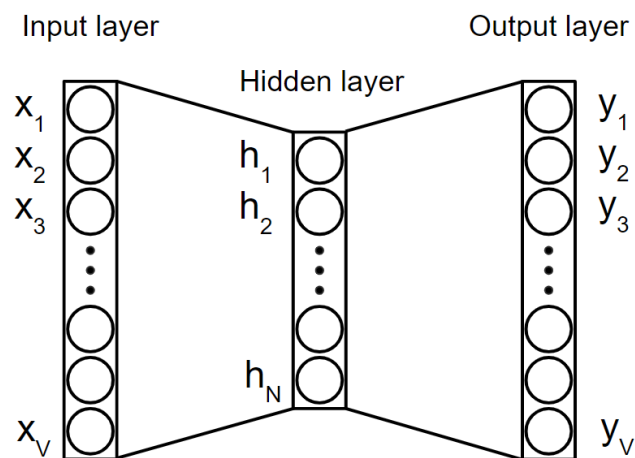


Figure 12: Word2vec.

The Skip Gram model works in a similar way to the CBOW, but instead of trying to predict the target word, it used the target word as input and tries to predict each word from the context. For each word from the context, it generates a separate pair of context word to the same target word, to be used as a training sample. So using the same example of size 2 window, instead of generating a single training sample, Skip Gram would generate 4 samples, pairing each context word with the same target word.

On a final note, when sampling the target word for both methods we said the process is repeated for each word in the corpus, but that does not have to be the case, we

can choose those by picking random words from the corpus (and then using the window technique to construct the pairs). In practice, they are not picked entirely uniformly at random, because more common words would be used much more often to train the network, so usually some heuristics are used.

3.2.2.3 GloVe

GloVe (PENNINGTON; SOCHER; MANNING, 2014) is another technique for constructing semantic word embeddings. First, it builds a co-occurrence matrix X with X_{ij} being the number of times word j appears in the context of word i , with the context being defined in the same way as explained in Section 3.2.2.2. Then it tries to minimize the Equation (3.4) by learning the correct weights w_i and \tilde{w}_j and biases b_i and b_j with f being a weighting function that can be used to give less importance to frequent terms and zero out the result if the count X_{ij} is zero, among other things.

$$J = \sum_{i,j=1}^V f(X_{ij}) (w_i^T \tilde{w}_j + b_i + b_j - \log X_{ij})^2 \quad (3.4)$$

Because w and \tilde{w} are equivalent when X is symmetric and to help reduce overfitting, the sum of the two can be used as the final word vectors for the embeddings as in Equation (3.5).

$$w_{final} = w + \tilde{w} \quad (3.5)$$

3.2.2.4 FastText

fastText⁴ is not an embedding technique per se, it is a library that provides tools to learn text representations and classifiers.

The part we are interested in is the text representation, more specifically embeddings. fastText proposes a framework (JOULIN et al., 2016) that works on top of other embedding generating methods: Skip Gram and CBOW from Word2vec. The way it differentiates is by changing how we choose the input / output pairs. Instead of using

⁴<https://fasttext.cc/>

just the one-hot encoded words, first, it breaks each one into n-grams, then building a vocabulary with the full words and their n-grams (not necessarily all possible n-grams due to memory constraints, sometimes only picking the most frequent) and one-hot encoding that final vocabulary. Finally to build the training pairs, whether it uses the CBOW or Skip Gram method, the input comes from the full words and their n-grams and the output being the corresponding (based on method) full word. With those pairs, they apply one of the Word2vec methods from Section 3.2.2.2.

This will generate embeddings for the full known words from the vocabulary and their n-grams. To compute the final embedding for a word to be used in an NLP task, for words in the vocabulary, we use the sum of the full word embedding with its n-grams embeddings as in Equation (3.6). If the word is not in the original vocabulary, then only the second part of the equation is used, the sum of its n-grams' embeddings.

$$Embedding^{final} = Embedding^{full\ word} + \sum_i^{word\ n-grams} Embedding_i^{n-gram} \quad (3.6)$$

3.3 Models

In this section, we present the different models that were or will be tested. Some of those will use the single vector representation for a document and some will use the matrix representation (one vector per word) as input.

3.3.1 Naive Bayes

Naive Bayes is a probabilistic classifier (RISH, 2001) based on the Bayes theorem which assumes independence between the input variables, or features, given a class. The method estimates the conditional probability of an input being in each class C_K , given the features x_1, \dots, x_n . The class maximizing this probability can then be assigned to the input. In order to train the classifier – *i.e.*, learn the conditional distribution $P(C | x)$ given the features vector x – we use the data to learn $P(x_i | C_k)$ for each feature x_i and class C_k , besides the prior $P(C_k)$. For a given x , the class C_k maximizing $P(C_k | x)$ can be obtained via the Bayes theorem by employing the independence assumption and ignoring the fixed term $P(x)$, as shown in Equation (3.7). In our case, there are two

possible classes, 0 if the facts' descriptions are different, 1 if similar.

To estimate the probabilities from the data, we must assume a probability distribution for the features x given a class C_k in order to learn its parameters. Our NB classifier assumed the multinomial distribution.

$$P(C_k | x) \propto P(C_k)P(x_1 | C_k) \dots P(x_n | C_k) \quad (3.7)$$

The Naive Bayes model does not require training as most of the other models do, as in learning from many iterations on the data. Instead, it just fits its internal parameters (probabilities) to said data. Because in our study we do not have the full training data (pairs of cases) assembled, rather we generate them randomly, we only "fit" the Naive Bayes model to a randomly generated subset of what would be the whole data.

3.3.2 Cosine Similarity

Cosine Similarity (NGUYEN; BAI, 2010) is a way to measure the distance between two vectors, given by the cosine of the angle between them, as defined in Equation (3.8), with A and B being the vectors and θ_{AB} the angle. It is worth noting that the Euclidean distance between normalized vectors is a monotone function of their cosine similarity. We also point out that the CS model has no parameter to be trained.

$$\text{CosineSimilarity}(A, B) = \cos \theta_{AB} = \frac{A \bullet B}{\|A\| \|B\|} \quad (3.8)$$

This distance can then be compared to a threshold, turning the result into a binary classification. We can determine this threshold the same way it is done for other machine learning techniques, by testing multiple values and choosing whichever gives the best results, depending on what we are looking for (the precision / recall trade-off).

3.3.3 Logistic Regression

Logistic regression (KLEINBAUM et al., 2002) in the context of machine learning is used for binary classification (problems with two classes), fitting our work of determining similarity between two pieces of text. It uses the logistic function or sigmoid from

Equation (3.9) by having input features x (vector) and output labels y (single value, from 0 to 1) as in Equations (3.10) and trying to learn weights (vector) w and biases (single value) b from the data.

$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}} \quad (3.9)$$

$$\begin{aligned} z &= w^T x + b \\ y &= \text{sigmoid}(z) \end{aligned} \quad (3.10)$$

3.3.4 Multilayer Perceptron

Multilayer Perceptron (ROSENBLATT, 1958) is a type of feedforward neural network with an input layer, one or more hidden layers and an output layer. The input layer represents the features (in our case being the concatenated vector from BoW), the hidden and output layers are composed of artificial neurons with weights, biases and an activation function. Each neuron process its inputs according to Equation (3.11), with x_i being each of the neuron's input, w_i the weight corresponding to input x_i , b the bias, φ the activation function and y the neuron's output. All w_i 's and b are parameters to be learned when training the network. Figure 13 shows an example of MLP with a single output, as is our model.

$$y = \varphi \left(\sum_{i=1}^n w_i x_i + b \right) \quad (3.11)$$

The layers of an MLP are sometimes called fully connected (or Dense) layers.

3.3.5 Long Short-term Memory

Long Short-Term Memory (HOCHREITER; SCHMIDHUBER, 1997) is a neural network layer that has memory capabilities, especially suitable for processing sequences of data, such as text. LSTM is a sophisticated type of *Recurrent Neural Network* or RNN. To understand RNN, a simple example is depicted in Figure 14.

An RNN process a sequence in step, or cells. Each step receives an activation

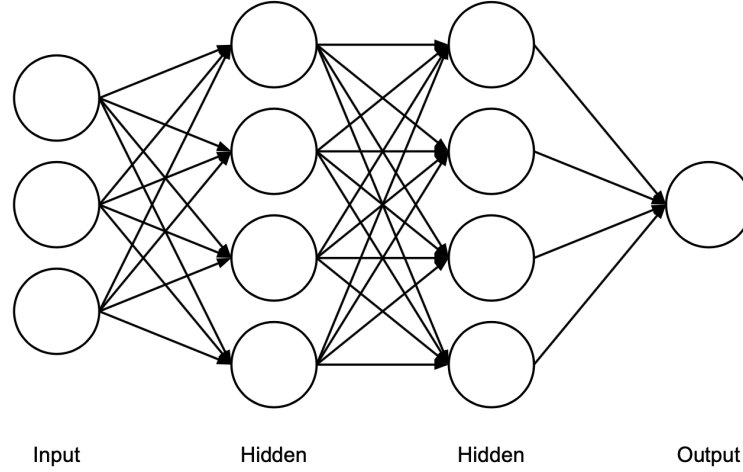


Figure 13: Multilayer Perceptron with 3 inputs, 2 hidden layers of size 4 and output layer of size 1.

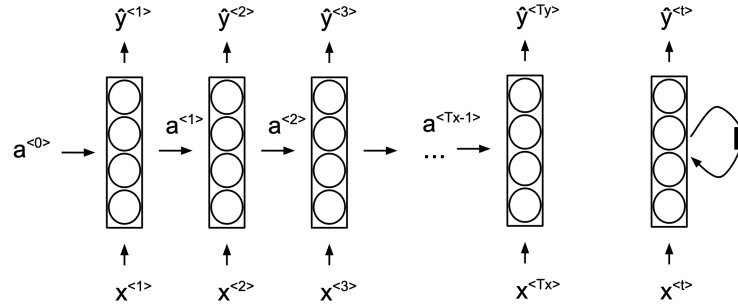


Figure 14: Recurrent Neural Network representation - full representation to the left and shorter representation to the right.

$(a^{(t-1)})$ and an element $(x^{(t)})$ of the sequence that is being interpreted, outputting another activation $(a^{(t)})$ and, sometimes, a separate result $(\hat{y}^{(t)})$ – all inputs and outputs being vectors. The first activation $(a^{(0)})$ is usually a vector of zeros, while the activation input for the next step is the activation output from the current step. Equations (3.12)-(3.15) formalizes the behavior of a RNN layer, where t denotes the processing step, $x^{(t)}$, the input element for step t , $\hat{y}^{(t)}$, the output element for step t , $a^{(t)}$, the activation for step t , W , the weights (each different sub-index represents a different set of weights), b , the biases (each different sub-index represents a different set of biases) and T , the sequence size (the separate result $\hat{y}^{(t)}$ is not always needed for every step, that is why there is a T_y for that and a T_x for the input sequence - see Figure 14).

$$a^{(t)} = g(W_{aa}a^{(t-1)} + W_{ax}x^{(t)} + b_a) \quad (3.12)$$

$$\begin{aligned}
\tilde{c}^{(t)} &= \tanh(W_c[a^{(t-1)}, x^{(t)}] + b_c) \\
\Gamma_u &= \sigma(W_u[a^{(t-1)}, x^{(t)}] + b_u) \\
\Gamma_f &= \sigma(W_f[a^{(t-1)}, x^{(t)}] + b_f) \\
\Gamma_o &= \sigma(W_o[a^{(t-1)}, x^{(t)}] + b_o) \\
c^{(t)} &= \Gamma_u * \tilde{c}^{(t)} + \Gamma_f * c^{(t-1)} \\
a^{(t)} &= \Gamma_o * \tanh(c^{(t)})
\end{aligned} \tag{3.16}$$

For a detailed explanation of the LSTM architecture, see (HOCHREITER; SCHMID-HUBER, 1997).

3.3.6 Transformer

Transformer (VASWANI et al., 2017) is a network architecture based on attention mechanisms without the use of recurrence or convolutions.

The architecture from the referenced paper was built for sequence to sequence tasks (namely language translations), but part of this model can also be used for other tasks (DEVLIN et al., 2018), and that is what we are interested in for our work.

The Transformer is composed of an encoder and a decoder stack, with the encoder converting the input sequence to contextual encodings that can then be consumed by the decoder to generate the output sequence, as in Figure 16. We will only explain the encoder part, since that is what we will be using for some of the proposed architectures for the legal texts similarity task.

The encoding process can be summarized as converting the input to embeddings, adding a positional encoding, passing it through the self-attention block, then through a feed-forward neural network (FFNN) finally generating the result vector for each word.

The input to embedding can use any embedding technique such as Word2vec or GloVe.

The positional encoding is another vector that represents the position of the word in the sentence (similar to a binary encoding for the position number, but the Transformer paper uses a different technique with sine and cosine functions) and is added to the generated embedding to allow the attention model to learn from the order of the words.

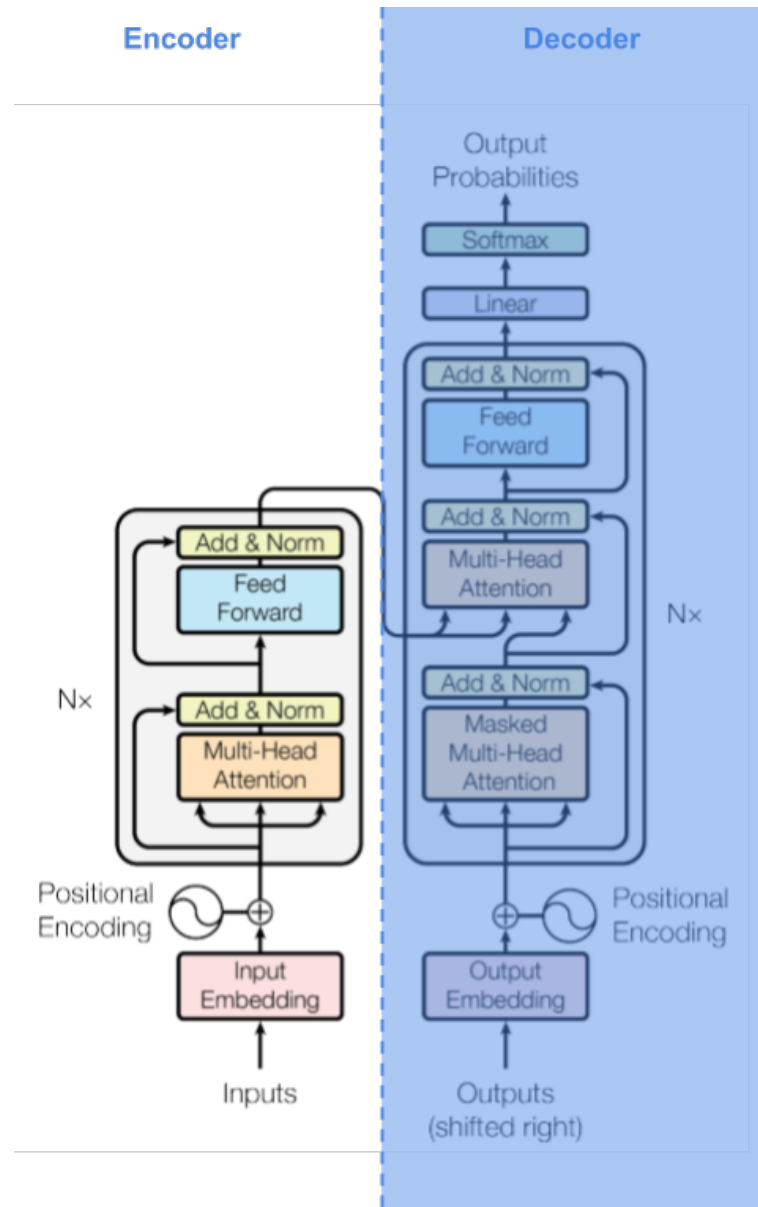


Figure 16: Transformer architecture. Copied from (VASWANI et al., 2017).

The self-attention head can be split into the following steps: calculate query (Q), key (K) and value (V) \rightarrow calculating scores for each word \rightarrow divide by the square root of the dimension of the key vectors \rightarrow apply softmax \rightarrow multiply by V \rightarrow sum up the vectors generating the result. Considering X to be the input sequence encodings generated by the embeddings, W to be the multiple weight matrices for Q , K and V , and Z to be the final output from the self-attention head, Equations (3.17) show the math behind it and

Figure 17 presents an overview of the attention mechanism.

$$\begin{aligned}
 X_{n \times d_E} \times W_Q_{d_E \times d_W} &= Q_{n \times d_W} \\
 X_{n \times d_E} \times W_K_{d_E \times d_W} &= K_{n \times d_W} \\
 X_{n \times d_E} \times W_V_{d_E \times d_W} &= V_{n \times d_W} \\
 \text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_W}}\right) V &= Z_{n \times d_W}
 \end{aligned} \tag{3.17}$$

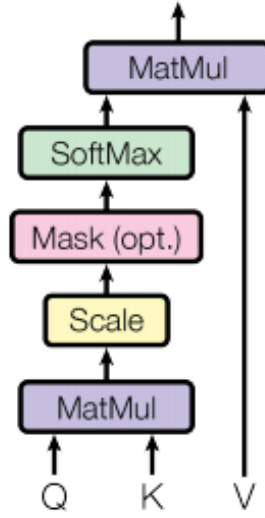


Figure 17: Scaled Dot-Product Attention. Copied from (VASWANI et al., 2017).

To complete the full self-attention block from the paper, they actually use multiple heads, concatenate the results, multiply by yet another weight matrix and then generate the final output as in Figure 18. Equation (3.18) shows how the final output from the self-attention block is calculated. Note that Z_{final} has the same dimensions as the embeddings X , allowing multiple encoder blocks to be stacked easily.

$$[Z_0 Z_1 \dots Z_h]_{n \times d_W \cdot h} \times W_0_{d_W \cdot h \times d_E \cdot h} = Z_{final}_{n \times d_E} \tag{3.18}$$

The last part of the encoder block consists of passing each vector from the self-attention block result through an FFNN, generating the final result for this encoder block. One important detail is that the FFNN does not change those vectors' (one for each word, or a matrix) dimensions, allowing multiple blocks to be stacked. This brings us to the final important note of the encoder, it comprises multiple of those encoder blocks stacked together to form the full encoder.

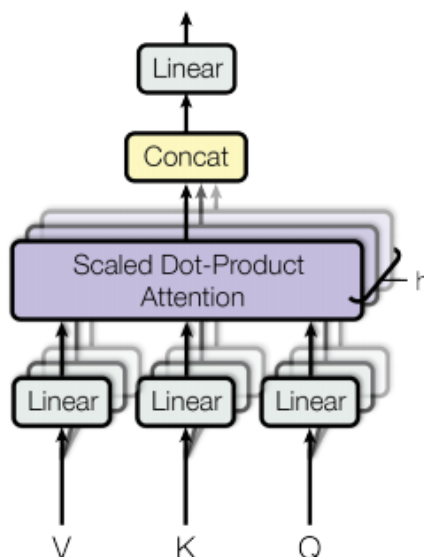


Figure 18: Multi-head attention. Copied from (VASWANI et al., 2017).

Section 3.3.7 will explain how Siamese neural networks work and what a feature extractor is, but in order to use the Transformer encoder as a feature extractor, there is one extra step we will be taking: summing up (element-wise) the output vectors for each word into a single vector and dividing by the square root of the sequence size as in (CER et al., 2018).

3.3.7 Siamese Neural Network

Siamese Neural Network (BROMLEY et al., 1994) is a type of neural network architecture that has at least two sub-networks (shared layers S1 and S2) with identical weights and configurations. In our case, the two inputs (facts' descriptions) are processed by these shared layers separately, and the outputs are concatenated (Concat) before being processed by other layers (C1 and C2). A simple structure of a Siamese network can be seen in Figure 19.

Those sub-networks are sometimes called feature extractors, because they are responsible for receiving the input and processing it into outputs that will be concatenated and compared in some way to generate the final output type (a similarity measure between zero and one in our case).

After processing the two inputs through the shared layers, they can together (concatenated) be further processed by the final layers which are responsible for learning

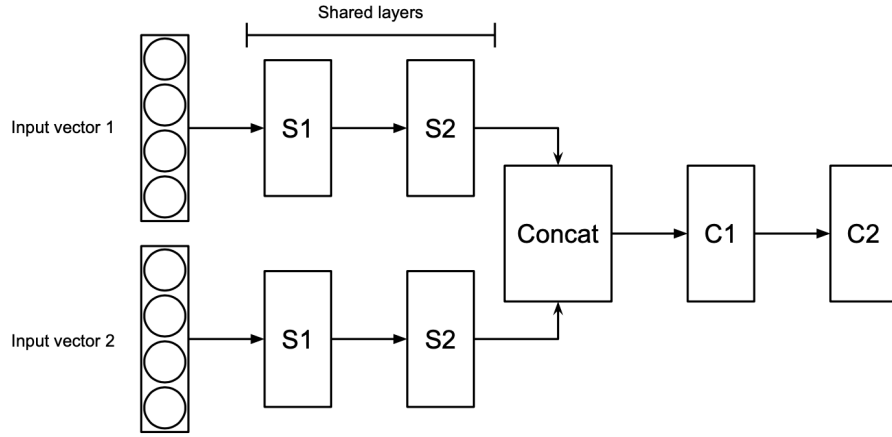


Figure 19: Siamese network structure.

the comparison function (similarity function in our scenario). We will be exploring two options for those final layers: a small MLP network and the exponential negative Manhattan distance (ENMD).

The small MLP as a final layer can, theoretically, learn the comparison function between the two processed inputs in a similar way a regular MLP would for initial inputs.

The Manhattan distance between two vectors is computed by summing up the absolute distances for each dimension. The exponential negative Manhattan distance, Equation (3.19) (with F^1 and F^2 being the two vectors extracted from the inputs processed by the shared layers), is just the exponential of the negative of that value. When the Manhattan distance is 0, the result for the *ENMD* will be 1 and when it is very big, the result will be closer to 0, thus allowing us to interpret it as a similarity function between the two processed inputs.

$$ENMD = \exp\left(-\sum_{i=1}^n |F_i^1 - F_i^2|\right) \quad (3.19)$$

3.4 Architectures

In this section, we present the several architectures that were tested, with each being a combination of the models and techniques presented in section 3.3.

We can distinguish our architectures by looking at 4 different selections of patterns: the text representation, the way to process both inputs (one for each text of the pair of

documents being compared), the base model used for calculation and, if we are using an SNN, a type of output to merge the two processed inputs. Those are represented in Figure 20, with each of the 4 "columns" being the selection patterns and the gray boxes being the options. Those gray boxes are also grouped (by white boxes) to better organize the types of options. The arrows indicate which way one can choose an option (from left to right) to build one of our 28 architectures.

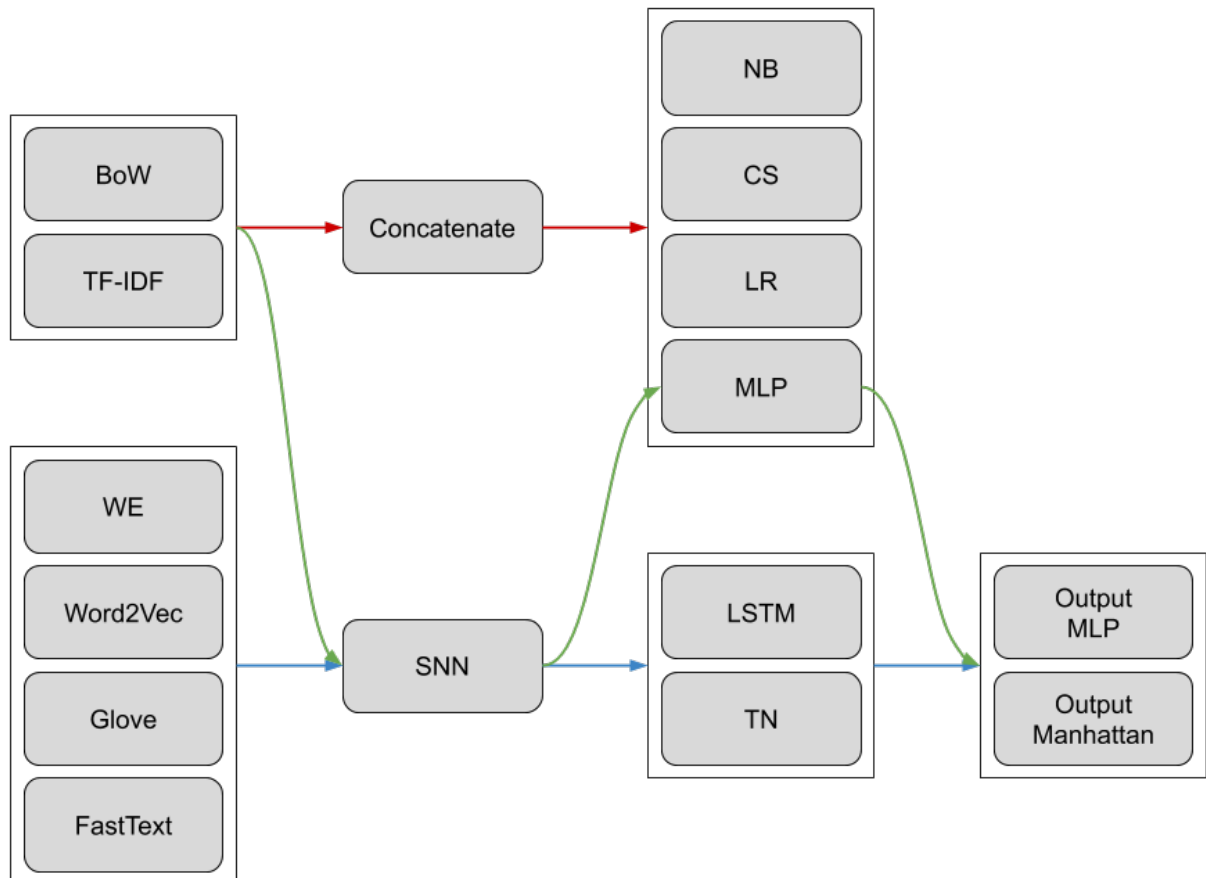


Figure 20: Architecture combinations.

Starting with the red arrows' path, we have the 2 possible single vector per document representations: BoW or TF-IDF. The output of one of those for each piece of text (from the pair) can then be concatenated into a single vector to be finally processed into our similarity result by one of the 4 base models: NB, CS, LR or MLP. Resulting in 8 architectures.

- BoW + NB
- BoW + CS

- BoW + LR
- BoW + MLP
- TF-IDF + NB
- TF-IDF + CS
- TF-IDF + LR
- TF-IDF + MLP

The NB based architectures used a multinomial Naive Bayes classifier with a smoothing parameter of 1.

For the CS based architectures, the threshold was chosen on a 0.1 interval based on the best accuracy, which ended up with 0.3 for BoW + CS and 0.1 for TF-IDF + CS.

The LR based architectures use a single sigmoid unit on the concatenated inputs.

The MLP based architectures use 2 internal layers of 256 neurons with ReLU as activation function with a single sigmoid neuron as output layer.

For the green arrows' path, instead of concatenating the 2 vectors representing the inputs pair, we process them through the same MLP model as before, but instead of having an output layer with a single sigmoid neuron, we replace it with 256 neurons (still using sigmoid as activation function), generating another 2 vectors that can be merge by one of the output options for the SNN: Output MLP or Output Manhattan (as explained in Section 3.3.7). The Output MLP concatenates the inputs, passes concatenated vector through 2 layers of 256 neurons using the ReLU as activation function and one final layer with a single neuron using sigmoid as activation function (similar to how BoW + MLP does). Resulting in another 4 architectures.

- BoW + SNN + MLP + Output MLP
- BoW + SNN + MLP + Output Manhattan
- TF-IDF + SNN + MLP + Output MLP
- TF-IDF + SNN + MLP + Output Manhattan

Finally, for the blue arrows's path, we start with the single vector per word representations: We, Word2Vec, Glove or FastText. The inputs are again processed by an SNN style architecture using one of the 2 available base models, LSTM or TN, as feature extractors, so they can then be merged by one of the 2 types of outputs as mentioned in the previous path: Output MLP or Output Manhattan. Resulting in yet another 16 architectures.

- WE + SNN + LSTM + Output MLP
- WE + SNN + LSTM + Output Manhattan
- WE + SNN + TN + Output MLP
- WE + SNN + TN + Output Manhattan
- Word2Vec + SNN + LSTM + Output MLP
- Word2Vec + SNN + LSTM + Output Manhattan
- Word2Vec + SNN + TN + Output MLP
- Word2Vec + SNN + TN + Output Manhattan
- Glove + SNN + LSTM + Output MLP
- Glove + SNN + LSTM + Output Manhattan
- Glove + SNN + TN + Output MLP
- Glove + SNN + TN + Output Manhattan
- FastText + SNN + LSTM + Output MLP
- FastText + SNN + LSTM + Output Manhattan
- FastText + SNN + TN + Output MLP
- FastText + SNN + TN + Output Manhattan

The WEs for the 2 LSTMs had an output dimension of 256. The WEs for the TN networks were trained with 50 as output dimensions instead, because with 256 the model was just not learning anything (accuracy stuck at 50%).

The models with pre-trained embeddings (Word2Vec, Glove and FastText) were all trained with 300 as output dimension (closest we could find to the 256 from WE and that were already trained in Portuguese) for the LSTM networks. Due to the same reason as before, we chose an output dimension of 50 for the TN based networks. They were all of the Skip Gram type.

All the pre-trained embeddings were downloaded from the NILC repository ⁵.

The LSTM based architectures assumed a maximum sequence length of 1k.

The TN based architectures had 2 layers, layer output dimension of 50, 2 heads, inner layer dimension of 16 and dropout rate of 0.1. The reason for not using the same parameters as the base model from (VASWANI et al., 2017) was due to a lack of better computational resources.

For all the architectures, when necessary, the models were trained with a vocabulary size of 10k, training batch size of 16, 2k steps per epoch and a start of 10 epochs. After the initial 10 epochs, each model was trained for another 10 and the loss history was analyzed: if the latest 10 epochs produced a smaller loss than the previous 10, the model was trained for another 10 epochs; if not, we stopped training at those initial 20 epochs.

For the NB based models, a total of 10 epochs was considered (20k total pairs) with no further "training", as it does not make sense for these kinds of models as explained in section 3.3.1.

It is worth noting that we did not consider some of the more advanced architectures such as using a pre-trained BERT as a feature extractor for an SNN, because it would take too long to train considering our computational resources and it did not seem like it would yield better results. As an example, we trained a simple Output MLP (similar to other models, concatenating the two inputs) with the inputs being the texts processed by a Brazilian Portuguese pre-trained BERT from NeuralMind ⁶ (the BERT itself was not trained at all, we just used it as a text processing function, similar to BoW and the others). We trained for 10 epochs and achieved the results from Figures 21 and 22, taking over 10h of training. Because of the long training time and the fact that the

⁵<http://www.nilc.icmc.usp.br/embeddings>

⁶<https://huggingface.co/neuralmind/bert-base-portuguese-cased>

accuracy did not seem like it would increase to the point of even reaching the results of our other models, we decided to forgo further exploring that architecture.

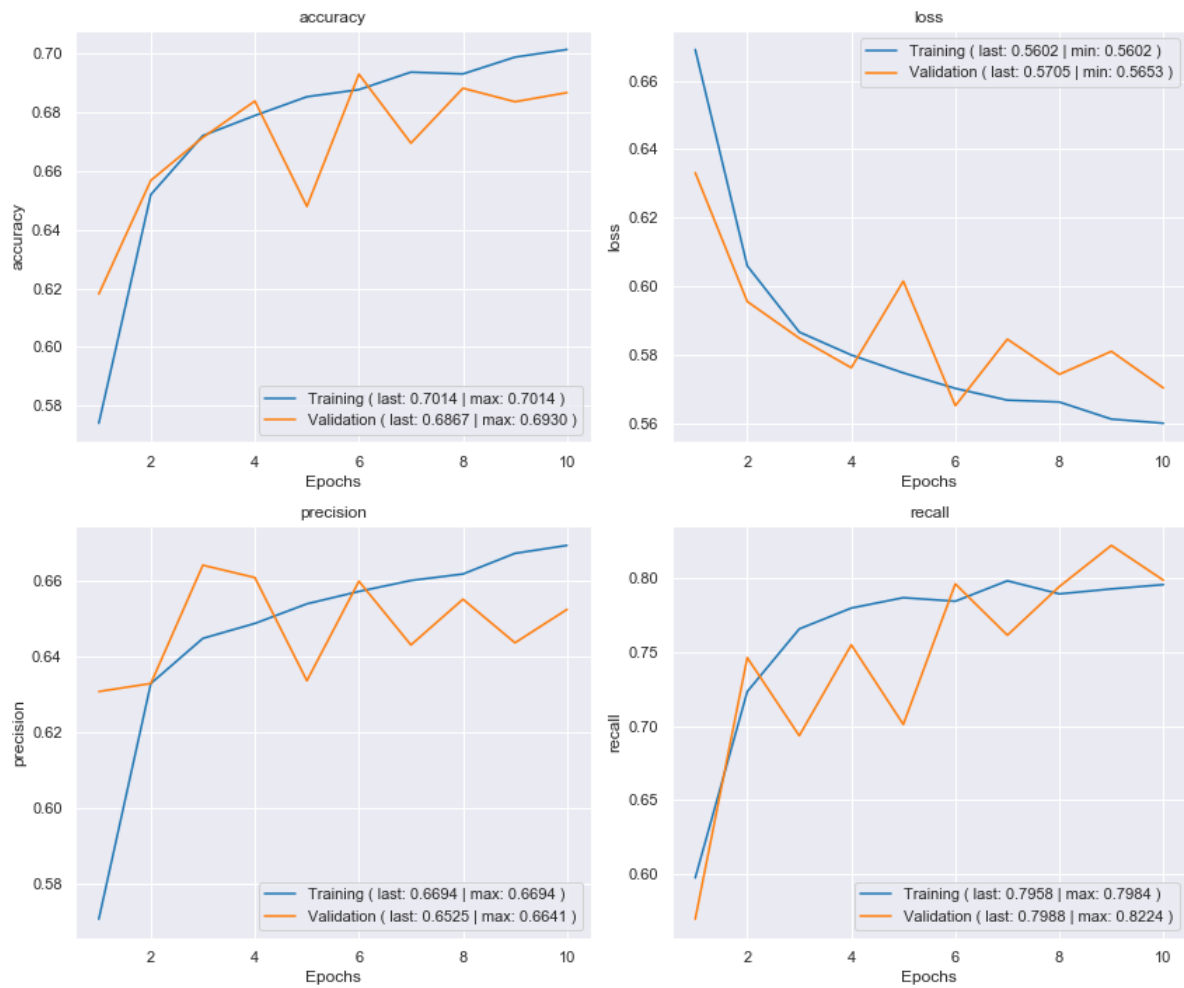


Figure 21: Accuracy, precision and recall for BERT + SNN + Output MLP.

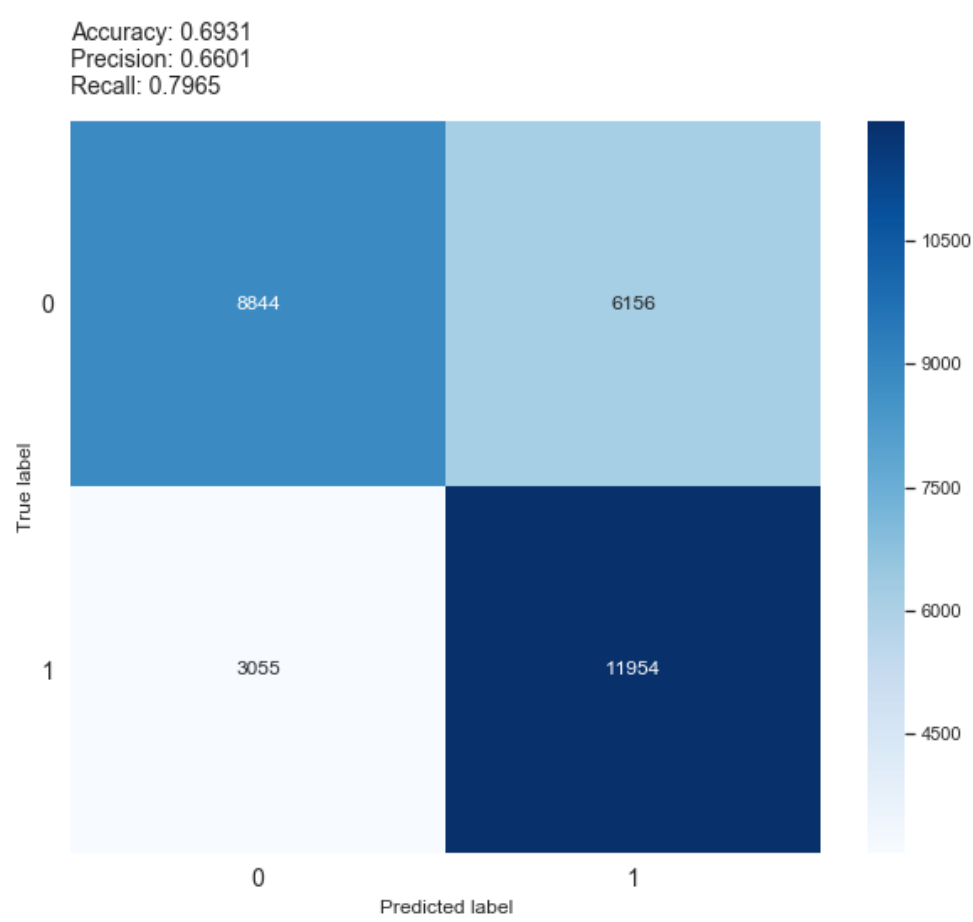


Figure 22: Validation confusion matrix for BERT + SNN + Output MLP.

4 EXPERIMENTS AND RESULTS

This section will present all the experiments and results for the 28 different architectures on the STJ dataset, along with the final model chosen with added regularization.

All the experiments were constructed using the following libraries: scikit-learn¹, Keras² and NLTK³. The first was mainly used for NB and CS based models, the second for all the others (SNN, MLP, LSTM, TN) and the third for pre-trained embeddings (Word2Vec, GloVe, FastText).

All the Keras models (all except NB and CS) were built using the binary cross-entropy loss with the library's standard parameters for the Adam optimizer (*learning rate* = 0.001, *beta*₁ = 0.9, *beta*₂ = 0.999).

All experiments were conducted on a Windows PC with the following specs:

- Processor: Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
- RAM: 32.0 GB
- System type: 64-bit
- GPU: NVIDIA GeForce RTX 2080 Ti

The code used for data extraction from the STJ website, data processing and text pairs generation, training and evaluation of the models, is available for non-commercial research purposes under the license CC-BY-NC-ND at the following repository: <https://github.com/rodrigorui/jurisprudence-research/>.

Before presenting the results, let us define accuracy, precision and recall. Consider the confusion matrix from Figure 23 with the following definitions:

- TP = true positives.
- TN = true negatives.

¹<https://scikit-learn.org/>

²<https://keras.io/>

³<https://www.nltk.org/>

- FP = false positives.
- FN = false negatives.

True label	0	TN	FP
	1	FN	TP
		0	1
		Predicted label	

Figure 23: Confusion matrix.

The true labels are the actual values from the dataset and the predicted labels are the values predicted by a model (if those are equal, then the model correctly predicted the values). Because the actual labels can be 0 or 1 and the model can predict values 0 or 1 (in our case it predicted values from 0 to 1 and we rounded it), we have 4 possible combinations for actual value vs predicted value, generating the numbers TP, TN, FP and FN.

Now accuracy, precision and recall can be defined by Equations (4.1), (4.2) and (4.3):

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.1)$$

$$precision = \frac{TP}{TP + FP} \quad (4.2)$$

$$recall = \frac{TP}{TP + FN} \quad (4.3)$$

Finally, to better compare the results of each best model, we can look at Table 1 - the raw results can be found in Appendix B.

Table 1: STJ results on validation dataset

Input	Model	Output	Accuracy (%)	Precision (%)	Recall (%)	F1 score (%)
BoW		NB	50.61	50.60	51.71	51.15
		CS	73.82	83.09	59.82	69.56
		LR	47.28	47.50	51.67	49.50
		MLP	83.67	83.44	84.02	83.73
TF-IDF		NB	48.78	48.93	55.91	52.19
		CS	80.61	86.32	72.76	78.96
		LR	48.90	49.04	56.52	52.51
		MLP	83.75	84.92	82.08	83.48
BoW	SNN + MLP	MLP	83.47	81.68	86.28	83.92
		Manhattan	81.81	82.44	80.84	81.63
TF-IDF		MLP	84.75	83.83	86.12	84.96
		Manhattan	83.52	85.09	81.29	83.15
WE	SNN + LSTM	MLP	81.29	79.46	84.39	81.85
		Manhattan	82.10	82.97	80.77	81.86
	SNN + TN	MLP	81.29	79.59	84.16	81.81
		Manhattan	59.03	59.66	55.77	57.65
Word2Vec	SNN + LSTM	MLP	81.76	80.38	84.02	82.16
		Manhattan	81.81	82.00	81.52	81.76
	SNN + TN	MLP	75.09	72.83	80.04	76.26
		Manhattan	63.87	63.70	64.49	64.09
GloVe	SNN + LSTM	MLP	80.61	78.34	84.62	81.36
		Manhattan	82.56	82.83	82.13	82.48
	SNN + TN	MLP	78.13	75.40	83.49	79.24
		Manhattan	64.23	62.46	71.31	66.59
FastText	SNN + LSTM	MLP	80.67	77.92	85.60	81.58
		Manhattan	82.71	82.22	83.47	82.84
	SNN + TN	MLP	77.24	74.13	83.68	78.62
		Manhattan	67.15	66.81	68.19	67.49

The NB and LR models did not do very well in comparison to the best model, most likely due to their lack of abstracting higher patterns in the data. Though the CS models did considerably better, which seems counter-intuitive since it is also a very simple model (that does not even need training).

All the MLP, SNN + MLP and SNN + LSTM models achieved very close results, with TF-IDF + SNN + MLP + Output MLP being the best one in terms of accuracy and F1 score. TF-IDF is a known technique for encoding large documents, while SNN can be used to compare two pieces of text that should be interpreted in the same manner with MLP being a common way of further processing our encoded input and flexible enough to fit a similarity function for the extracted features of each text, all which could explain why this was the best model.

The TN models achieved a broader range of results, with the ones using Output MLP being better than the Output Manhattan ones. Also having the WE + SNN + TN + Output MLP achieving results close to the best model.

From those results, we chose the model with the highest accuracy to further work on, but it does not imply this is the best model amongst all. Possibly TN or LSTM models did not get the proper adjustment on their hyper-parameters, such as number of heads, layers or neuron units on each layer, needed to achieve better results. It could also be that the LSTM models were not able to properly fit their parameters for such large documents, since they are usually better at handling smaller pieces of text.

Besides the NB and LR models, which presented the worst accuracies, it seems the results are more closely related to the actual data than the models themselves, since a lot of them were very close to each other. Many models with different complexity presented similar results, leading us to believe our problem might not need the complexity of word embedding types of input, so BoW and TF-IDF worked just as well. Possibly for the same reason, the TN and LSTM models showed similar results to MLP models, with MLP models doing slightly better, maybe due to them being easier to train.

As a side note, even if the CS model was the best one, it would still not be chosen simply because it cannot be much further improved (since it does not have a learning process).

As for the output layer, Output MLP having more flexibility in fitting the data when

compared to Manhattan might have given a slight edge on the last stages of data processing to help fit the layers with less data (though not a big difference, since not all models performed better with Output MLP).

Also, in hindsight, since the CS models (especially the one paired with TF-IDF) performed fairly well (with the TF-IDF + CS being very close to the best one), it might indicate that the ratio of word occurrences carry enough information to verify similarity between text pairs built from our extracted dataset, since this is essentially what a CS model compares, the distance between the two vectors with each vector being directly related to the ratio of words (via BoW or TF-IDF). This might also indicate that in other domains where the word ratio is heavily correlated to the task result, the BoW and TF-IDF type of text encoding should produce good results.

Finally for the best model of the 28 tested, TF-IDF + SNN + MLP + Output MLP, we explored a few regularization methods: L1, L2, Dropout and increasing the batch size.

We added dropout layers between all the others, similar to what was done in the original paper (SRIVASTAVA et al., 2014) (after each input layer, after each dense layer in the feature model, after the two extracted and concatenated features, and after each dense layer of the final output processing, except for the very last layer which outputs the actual classification.) with a dropout rate of 0.3. The final architecture can better be seen in Figure 24.

In order to evaluate this final model, we then constructed the fixed test dataset with other 30k pairs generated from those 10k test cases in the way explained in Subsection 3.1. The results on the validation and test datasets can be seen in Table 2.

Table 2: Results for the final model

TF-IDF + SNN + MLP + Output MLP				
Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1 score (%)
Validation	86.58	84.30	89.91	87.01
Test	85.98	83.89	89.06	86.40

As we can see, the added regularization did not improve much on the validation dataset in the first place, further corroborating our intuition that the data was much

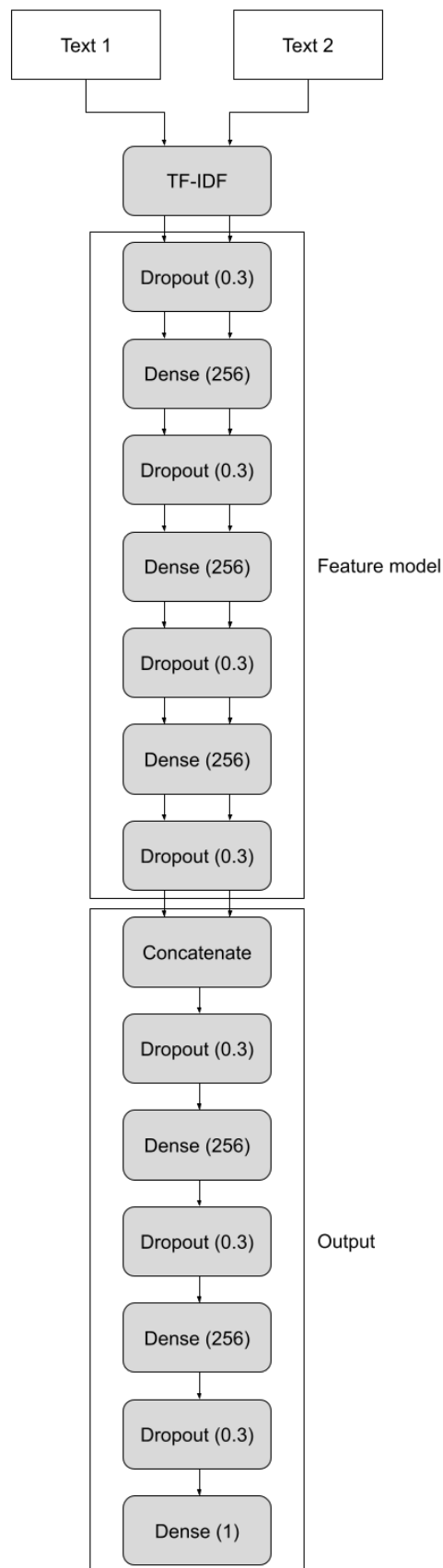


Figure 24: Final architecture.

more relevant in the results than the model itself. This might also indicate that we are reaching the limit of how precise the data is. Since it was not a manually classified dataset, we cannot know the exact accuracy of the generated pairs used to train and evaluate the models and we do know any model cannot do better than the data's accuracy it is being used to train on. In addition to that, it might also be the case that some pairs of texts should be considered similar, but were not done so because the past decision of the to be referenced case was not in accordance with what the lawyer wanted for their current case, which further contributes to decreasing the accuracy of our generated data.

As for the results on the test dataset, they were very close to the ones on the validation dataset (being slightly worse, as would be expected), meaning we managed to train a model without much bias towards our training (and validation) dataset.

So if our assumption that the extracted dataset from the original cases is good enough to represent what a manually classified dataset would be is true, then those results mean that, on average, for more than 8 every 10 cases a lawyer read from our model's recommendation will be similar to their current case, and that is considering we show the cases classified as similar in any order. If we order the cases by the model's confidence in the prediction, the assertiveness would be even higher. Thus effectively reducing the search time of the lawyer on this task.

The resulting model could also potentially be used, either as-is or as a starting point, to classify cases from other courts. Depending on how similar the structure of their text is, it might be possible to re-use the same models without further training.

5 CONCLUSION

During this work, we explored different NLP and ML techniques to address the task of determining whether two pieces of text were similar, more specifically, two facts' descriptions from juridical cases.

We started by extracting a dataset of past judgments from STJ since we did not have a manually classified dataset. For that, we presented an exploratory analysis that might help future studies on the same data. The data itself could also be used as a new benchmark for text interpretation of legal cases in Portuguese, comprising many possible tasks besides the jurisprudence classification done in this study, thus being one of our work's contributions.

Using that data we explored 28 different architectures (as described in section 3.4) and investigated with a more thorough parameter search the one with the best results, TF-IDF + SNN + MLP + Output MLP, leading to the final result of 85.98% accuracy in the test dataset. After all that testing, we can say we did not anticipate the results of many of those architectures being so similar to each other, a fact that could help future studies to find another architecture with a key difference that could improve our results. For now, we maintain our conclusion that the results we got were more dependent on the data than the models themselves, though we can suggest starting with TF-IDF as a document encoding technique when dealing with large pieces of text, due to Word Embeddings with LSTMs usually being harder to fit into problems with larger texts. We also suggest starting with an MLP feature extractor rather than a TN based one, since TNs are harder to train and, in our case, lead to worse results.

Also as another contribution, all the code for our work was made publicly available for research purposes (and can be found in the repository specified in Chapter 4).

Even so, due to the lack of even more powerful computers, not all of the latest techniques were evaluated, including bigger and more expensive networks like ELMO, ULMFit, BERT, and XLNet, leaving those for future work, along with procuring a manually labeled similarity dataset. Besides leveraging pre-trained models to achieve results in the specific task of jurisprudence similarity, transfer learning could also be done from

our models to other tasks, mainly ones related to long juridical texts in Portuguese, possibly taking the feature extractor part of our models and using it as an encoder for other tasks.

Overall, NLP and ML are vastly studied fields, even in specific domains such as the legal one. Even so, we expect our work to contribute to yet another branch of this ongoing study by not only providing a new dataset of legal cases in Portuguese, but also helping other researchers with a starting point on the jurisprudence similarity task of large legal texts.

REFERENCES

- AL-KOFAHI, K. et al. A machine learning approach to prior case retrieval. In: **Proceedings of the 8th international conference on Artificial intelligence and law**. [S.l.: s.n.], 2001. p. 88–93.
- BROMLEY, J. et al. Signature verification using a "siamese" time delay neural network. In: **Advances in neural information processing systems**. [S.l.: s.n.], 1994. p. 737–744.
- BUENO, T. C. D. et al. Jurisconsulto: retrieval in jurisprudential text bases using juridical terminology. In: ACM. **Proceedings of the 7th international conference on Artificial intelligence and law**. [S.l.], 1999. p. 147–155.
- CER, D. et al. Universal sentence encoder. **arXiv preprint arXiv:1803.11175**, 2018.
- CHANG, K.-W. et al. A dual coordinate descent method for large-scale linear svm. In: **Proc. Intl. Conf. on Machine Learning (ICML)**. [S.l.: s.n.], 2008.
- CHEN, Q. et al. Enhanced lstm for natural language inference. In: **ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)**. [S.l.: s.n.], 2017. v. 1, p. 1657–1668.
- COHEN, E. **How to predict Quora Question Pairs using Siamese Manhattan LSTM**. 2017. <<https://medium.com/mlreview/implementing-malstm-on-kaggles-quora-question-pairs-competition-8b31b0b16a07>>. [Online; accessed 19-December-2018].
- COLLOBERT, R.; WESTON, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In: ACM. **Proceedings of the 25th international conference on Machine learning**. [S.l.], 2008. p. 160–167.
- DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. **arXiv preprint arXiv:1810.04805**, 2018.
- HARRIS, Z. S. Distributional structure. **Word**, Taylor & Francis, v. 10, n. 2-3, p. 146–162, 1954.
- HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.
- JOULIN, A. et al. Bag of tricks for efficient text classification. **arXiv preprint arXiv:1607.01759**, 2016.
- KLEINBAUM, D. G. et al. **Logistic regression**. [S.l.]: Springer, 2002.
- LANG, K. Newsweeder: Learning to filter netnews. In: **Machine Learning Proceedings 1995**. [S.l.]: Elsevier, 1995. p. 331–339.

- LE, Q.; MIKOLOV, T. Distributed representations of sentences and documents. In: **International conference on machine learning**. [S.l.: s.n.], 2014. p. 1188–1196.
- LECUN, Y. et al. Object recognition with gradient-based learning. In: **Shape, Contour and Grouping in Computer Vision**. [S.l.]: Springer, 1999. p. 319–345.
- LIN, C.-Y. Rouge: A package for automatic evaluation of summaries. In: **Text summarization branches out**. [S.l.: s.n.], 2004. p. 74–81.
- LIU, Y.; YANG, X. A similar legal case retrieval system by multiple speech question and answer. In: **Proc. 18th Int. Conf. Electron. Bus.(ICEB)**. [S.l.: s.n.], 2018. p. 72–81.
- MAAS, A. L. et al. Learning word vectors for sentiment analysis. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1**. [S.l.], 2011. p. 142–150.
- MAHESHWARY, S.; MISRA, H. Matching resumes to jobs via deep siamese network. In: INTERNATIONAL WORLD WIDE WEB CONFERENCES STEERING COMMITTEE. **Companion of the The Web Conference 2018 on The Web Conference 2018**. [S.l.], 2018. p. 87–88.
- MIKOLOV, T. et al. Efficient estimation of word representations in vector space. In: **Proceedings of the International Conference on Learning Representations (ICLR 2013)**. [S.l.: s.n.], 2013. p. 1–12.
- MINAEE, S.; LIU, Z. Automatic question-answering using a deep similarity neural network. In: IEEE. **2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)**. [S.l.], 2017. p. 923–927.
- MUELLER, J.; THYAGARAJAN, A. Siamese recurrent architectures for learning sentence similarity. In: **Thirtieth AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2016.
- NECULOIU, P.; VERSTEEGH, M.; ROTARU, M. Learning text similarity with siamese recurrent networks. In: **Proceedings of the 1st Workshop on Representation Learning for NLP**. [S.l.: s.n.], 2016. p. 148–157.
- NGUYEN, H. V.; BAI, L. Cosine similarity metric learning for face verification. In: SPRINGER. **Asian conference on computer vision**. [S.l.], 2010. p. 709–720.
- PENNINGTON, J.; SOCHER, R.; MANNING, C. Glove: Global vectors for word representation. In: **Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)**. [S.l.: s.n.], 2014. p. 1532–1543.
- PLANK, B.; SØGAARD, A.; GOLDBERG, Y. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In: **54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Short Papers**. [S.l.: s.n.], 2016. p. 412–418.
- RAMOS, J. et al. Using tf-idf to determine word relevance in document queries. In: PISCATAWAY, NJ. **Proceedings of the first instructional conference on machine learning**. [S.l.], 2003. v. 242, p. 133–142.

RISH, I. An empirical study of the naive bayes classifier. In: **IJCAI 2001 workshop on empirical methods in artificial intelligence**. [S.l.: s.n.], 2001. v. 3, p. 41–46.

RISSLAND, E. L.; SKALAK, D. B.; FRIEDMAN, M. T. Case retrieval through multiple indexing and heuristic search. In: **IJCAI**. [S.l.: s.n.], 1993. v. 93, p. 902–908.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958.

RUIZ, R.; BONA, G. D. Jurisprudence search based on facts similarity using nlp techniques. In: **VIII Workshop de Pós-Graduação - Engenharia de Computação - WPGEC 2019**. [S.l.: s.n.], 2019.

SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. Facenet: A unified embedding for face recognition and clustering. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 815–823.

SHIH, C. et al. Investigating siamese lstm networks for text categorization. In: **IEEE. 2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)**. [S.l.], 2017. p. 641–646.

SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. **The journal of machine learning research**, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

TRAN, V.; NGUYEN, M. L.; SATOH, K. Automatic catchphrase extraction from legal case documents via scoring using deep neural networks. **arXiv preprint arXiv:1809.05219**, 2018.

TRAN, V.; NGUYEN, S. T.; NGUYEN, M. L. Jnlp group: Legal information retrieval with summary and logical structure analysis. In: **Twelfth International Workshop on Juris-informatics (JURISIN)**. [S.l.: s.n.], 2018.

TRAPPEY, C. V.; TRAPPEY, A. J.; LIU, B.-H. Identify trademark legal case precedents-using machine learning to enable semantic analysis of judgments. **World Patent Information**, Elsevier, v. 62, p. 101980, 2020.

VASWANI, A. et al. Attention is all you need. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2017. p. 5998–6008.

WANG, Y. et al. Attention-based lstm for aspect-level sentiment classification. In: **Proceedings of the 2016 conference on empirical methods in natural language processing**. [S.l.: s.n.], 2016. p. 606–615.

ZHONG, H. et al. How does nlp benefit legal system: A summary of legal artificial intelligence. In: **Proceedings of ACL**. [S.l.: s.n.], 2020.

APPENDIX A – STJ SEARCH AND EXTRACTION

Figure 25 shows an example of the initial search by the generic term "acórdão" (the manifestation of a collegiate judicial body that reveals a legal position), to find all past cases from STJ.

Figure 26 presents an example of the report section of a case.

Figure 27 displays the part of the report where the jurisprudence reference is presented.

The screenshot shows the STJ (Superior Tribunal de Justiça) website search results for the term "acórdão". The browser address bar shows the URL "scon.stj.jus.br/SCON/pesquisar.jsp". The website header includes navigation links like "Institucional", "Processos", "Jurisprudência", "Precedentes (Repetitivos)", "Comunicação", "Leis e normas", "Sob medida", and "Contato e ajuda".

The search results are displayed under the heading "Jurisprudência do STJ". The search term "acórdão" is entered in the search bar. The results show 737.126 acórdãos encontrados. The results are filtered by "REPETITIVOS" (843), "IACS" (7), and "ACÓRDÃOS DE AFETAÇÃO" (316). The results are also filtered by "Órgãos Julgadores" and "Ministros".

The results are displayed in a table with columns: Súmulas (9), Acórdãos (737.126), Decisões Monocráticas (4.861.918), and Informativos e outros produtos. The results are sorted by "1 ~ 10" and "documentos por página".

The first result is "Documento 1". The details of the document are as follows:

- Processo:** AgRg no HC 668477 / SC, AGRADO REGIMENTAL NO HABEAS CORPUS, 2021/0156805-7
- Relator(a):** Ministro OLINDO MENEZES (DESEMBARGADOR CONVOCADO DO TRF 1ª REGIÃO) (1180)
- Órgão Julgador:** T6 - SEXTA TURMA
- Data do Julgamento:** 14/09/2021
- Data da Publicação/Fonte:** DJe 17/09/2021
- Ementa:** AGRADO REGIMENTAL NO HABEAS CORPUS. FURTO SIMPLES. PRINCÍPIO DA INSIGNIFICÂNCIA. ATIPICIDADE MATERIAL. INAPLICABILIDADE. MULTIRREINCIDÊNCIA E HABITUALIDADE DELITIVA VERIFICADAS. 1. A incidência do princípio da insignificância pressupõe a concomitância de quatro vetores: a) mínima ofensividade da conduta do agente; b) nenhuma periculosidade social da ação; c) reduzidíssimo grau de reprovabilidade do comportamento; e d) inexpressividade da lesão jurídica provocada.

Figure 25: STJ website search by "acórdão".

Superior Tribunal de Justiça

AgInt no AGRAVO EM RECURSO ESPECIAL Nº 1.491.095 - SP (2019/0113870-3)

RELATOR : **MINISTRO LUIS FELIPE SALOMÃO**
AGRAVANTE : MARCELO HENRIQUE DE OLIVEIRA
AGRAVANTE : PEDRO HENRIQUE DE OLIVEIRA
ADVOGADO : ROQUE RODRIGUES - SP231255
AGRAVADO : LUCAS FERNANDO PONTALTI KRASUCKI
ADVOGADOS : TERUO TAGUCHI MIYASHIRO E OUTRO(S) - SP086111
 MARISA REGINA AMARO MIYASHIRO - SP121739
 JOSÉ HENRIQUE CASTELO BRANCO NEVES DA SILVA E
 OUTRO(S) - DF046240

RELATÓRIO

O SENHOR MINISTRO LUIS FELIPE SALOMÃO (Relator):

1. Cuida-se de agravo interno interposto em face da decisão monocrática de fls. 482-485, que conheceu do AREsp para dar provimento ao recurso especial, ao fundamento de que, muito embora a jurisprudência do STJ reconheça a legitimidade do filho para suscitar em embargos de terceiro a impenhorabilidade do bem de família em que reside, isso não pode ser usado para, por via transversa, desconstituir a coisa julgada material, proferida em demanda a envolver os próprios proprietários do bem.

Nas razões recursais, aduzem os recorrentes que, no caso, não se pode presumir que o mútuo contraído com o banco com pacto adjeto de hipoteca foi pra beneficiar a entidade familiar.

Ponderam que o imóvel é bem de família e que, em se tratando desse instituto, tem que se ter em mente que seu objetivo é proteger a habitação da família, resguardando o direito universal de moradia.

Obtemperam que a Corte local decidiu à luz das provas e das circunstâncias fáticas, e que a renúncia à impenhorabilidade em contratos é ineficaz, pois a dívida foi contraída por empresa, não ficando provado que reverteu em favor da família.

Dizem que a decisão tem capítulos autônomos não impugnados.

É o relatório.

Figure 26: Report section from a case's PDF.

Mutatis mutandis, merece também registro o seguinte precedente:

AGRAVO REGIMENTAL NO AGRAVO EM RECURSO ESPECIAL. EMBARGOS À ARREMATACÃO. BEM DE FAMÍLIA. COISA JULGADA MATERIAL. IMPOSSIBILIDADE DE REVISÃO. SÚMULA Nº 7/STJ.

1. **A alegação de impenhorabilidade foi analisada pelas instâncias de origem a partir da mesma relação jurídica e com base nos mesmos**

Documento: 1882266 - Inteiro Teor do Acórdão - Site certificado - DJe: 05/11/2019

Página 8 de 5

Superior Tribunal de Justiça

atos e provas, de modo que não é cabível rediscuti-la.

2. Infirmar a conclusão do aresto recorrido - para admitir que o imóvel do agravante de fato satisfazia os requisitos da Lei nº 8.009/90 - implicaria a revisão do contexto fático-probatório dos autos, o que é vedado pela Súmula nº 7/STJ, óbice que impede a admissão dos recursos interpostos com fundamento em qualquer das alíneas do permissivo constitucional.

3. Agravo regimental **não provido**.

(AgRg no AREsp 625.704/SP, Rel. Ministro RICARDO VILLAS BÔAS CUEVA, TERCEIRA TURMA, julgado em 18/06/2015, DJe 06/08/2015)

Figure 27: Reference from a case's PDF.

APPENDIX B – STJ RESULTS

The raw results for the investigation on each of the 28 architectures trained with the STJ dataset can be seen in Figures 28 to 81, displaying the loss, accuracy, precision and recall curves during training along with the confusion matrix of the best of each model (found during training) applied to the validation data. NB and CS models are the exceptions, because there is no training in either, as explained before in sections 3.3.1 and 3.3.2 respectively.

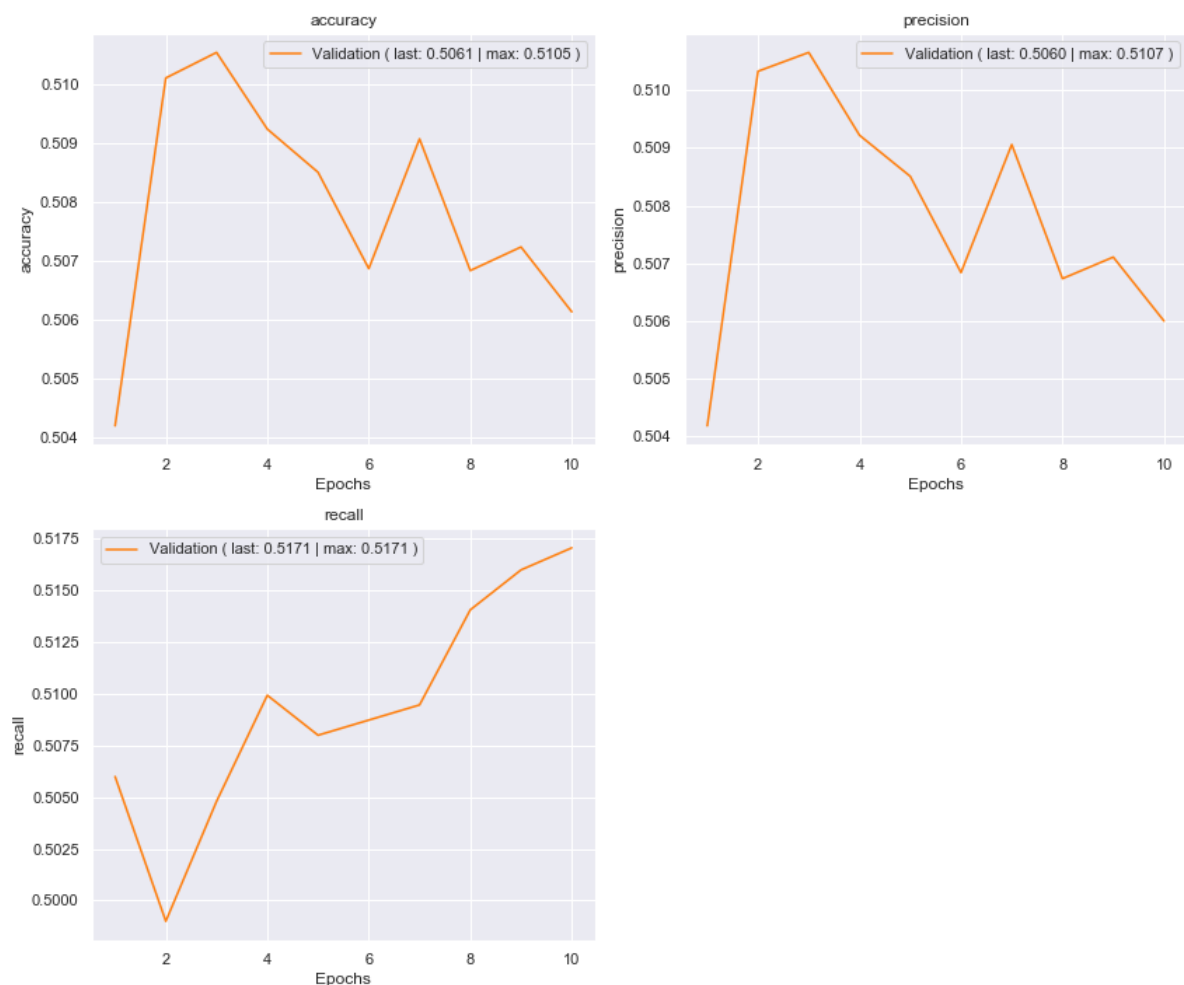


Figure 28: Accuracy, precision and recall for BoW + NB.

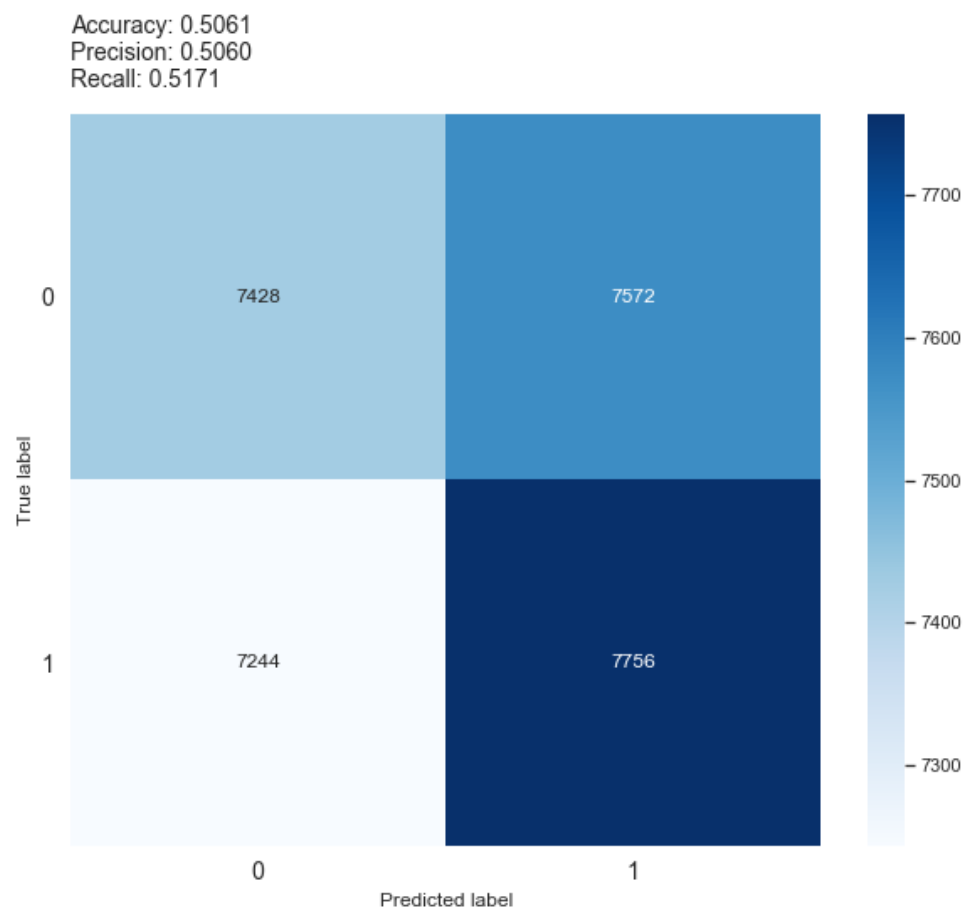


Figure 29: Validation confusion matrix for BoW + NB.

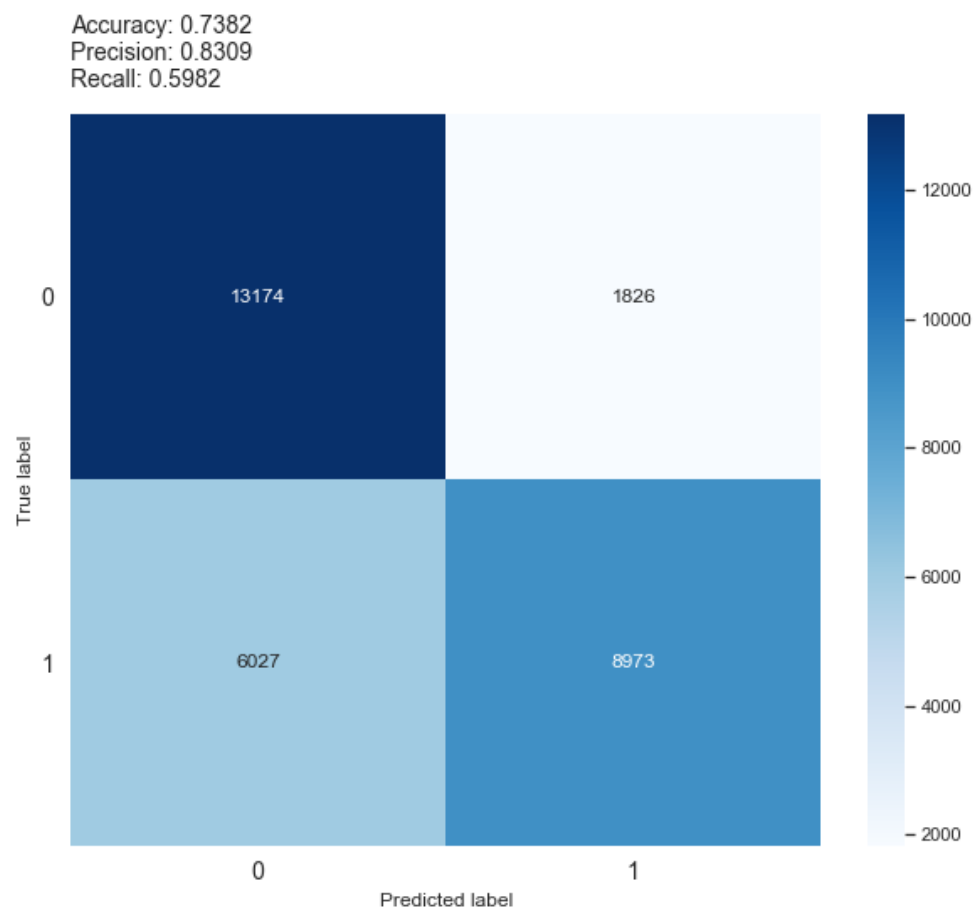


Figure 30: Validation confusion matrix for BoW + CS.

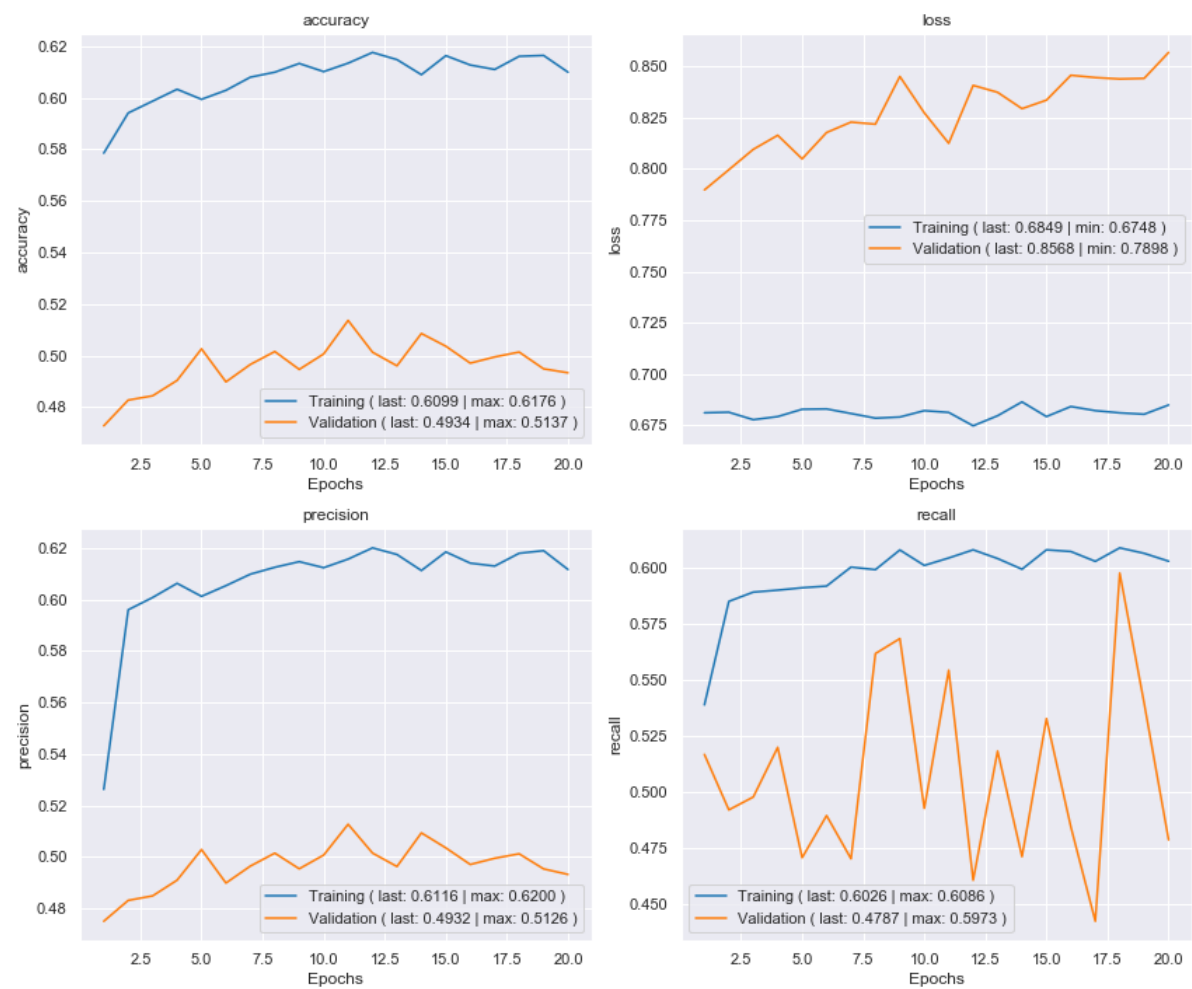


Figure 31: Accuracy, precision and recall for BoW + LR.

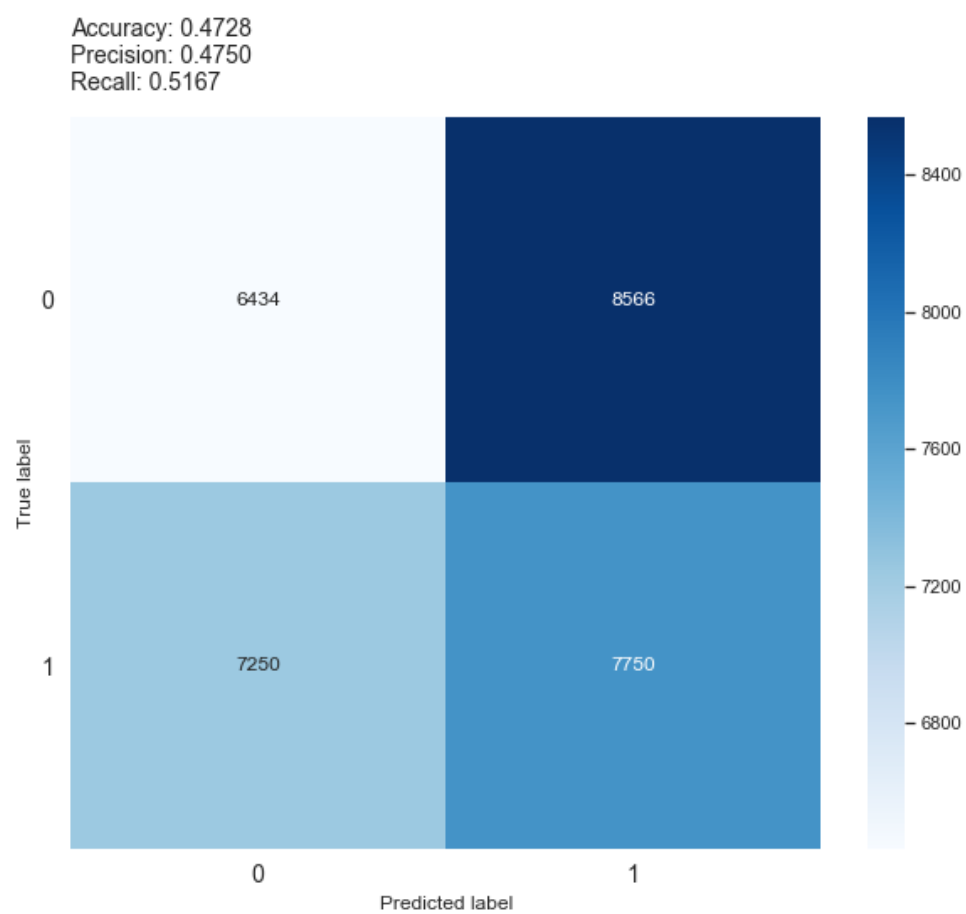


Figure 32: Validation confusion matrix for BoW + LR.

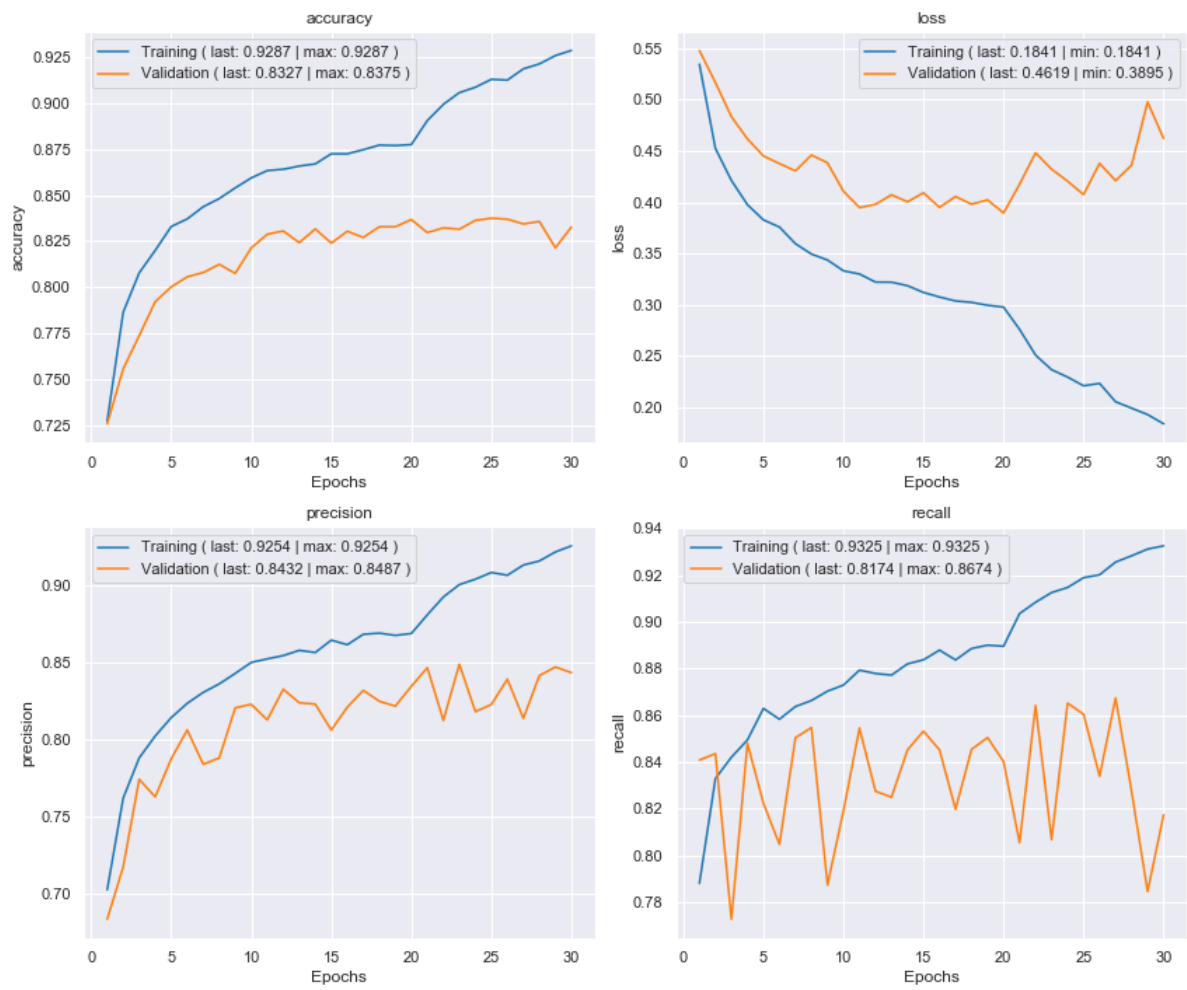


Figure 33: Accuracy, precision and recall for BoW + MLP.

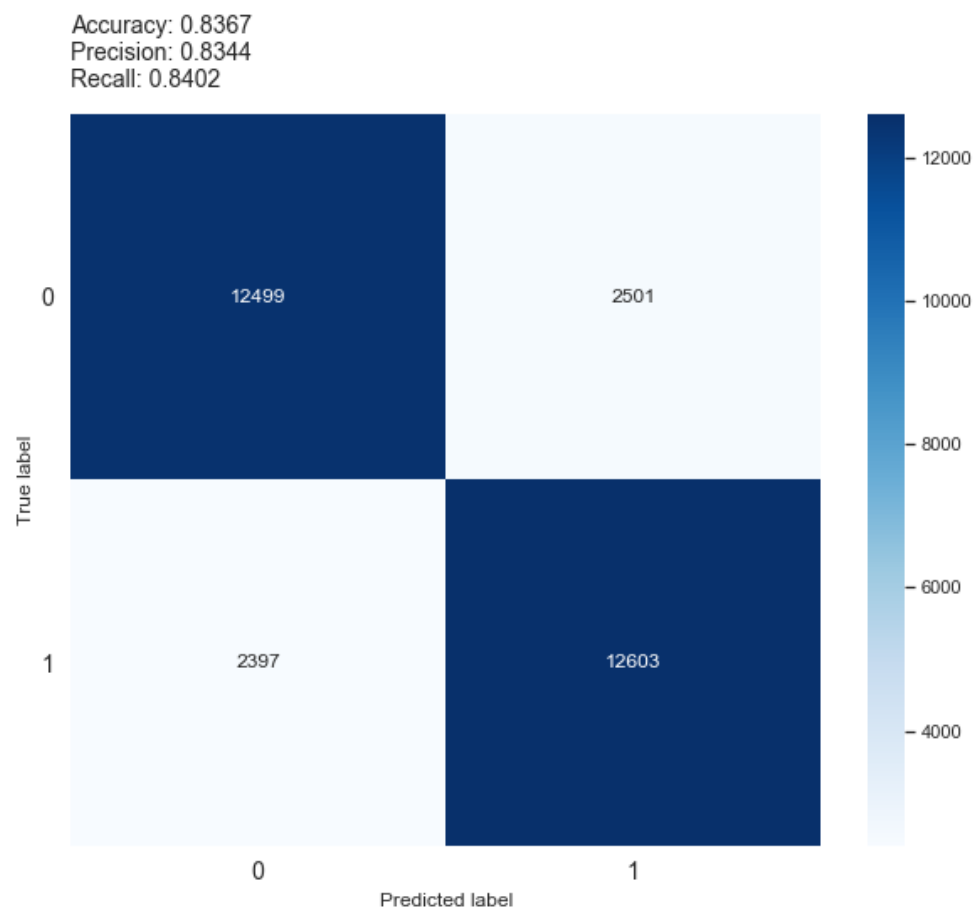


Figure 34: Validation confusion matrix for BoW + MLP.

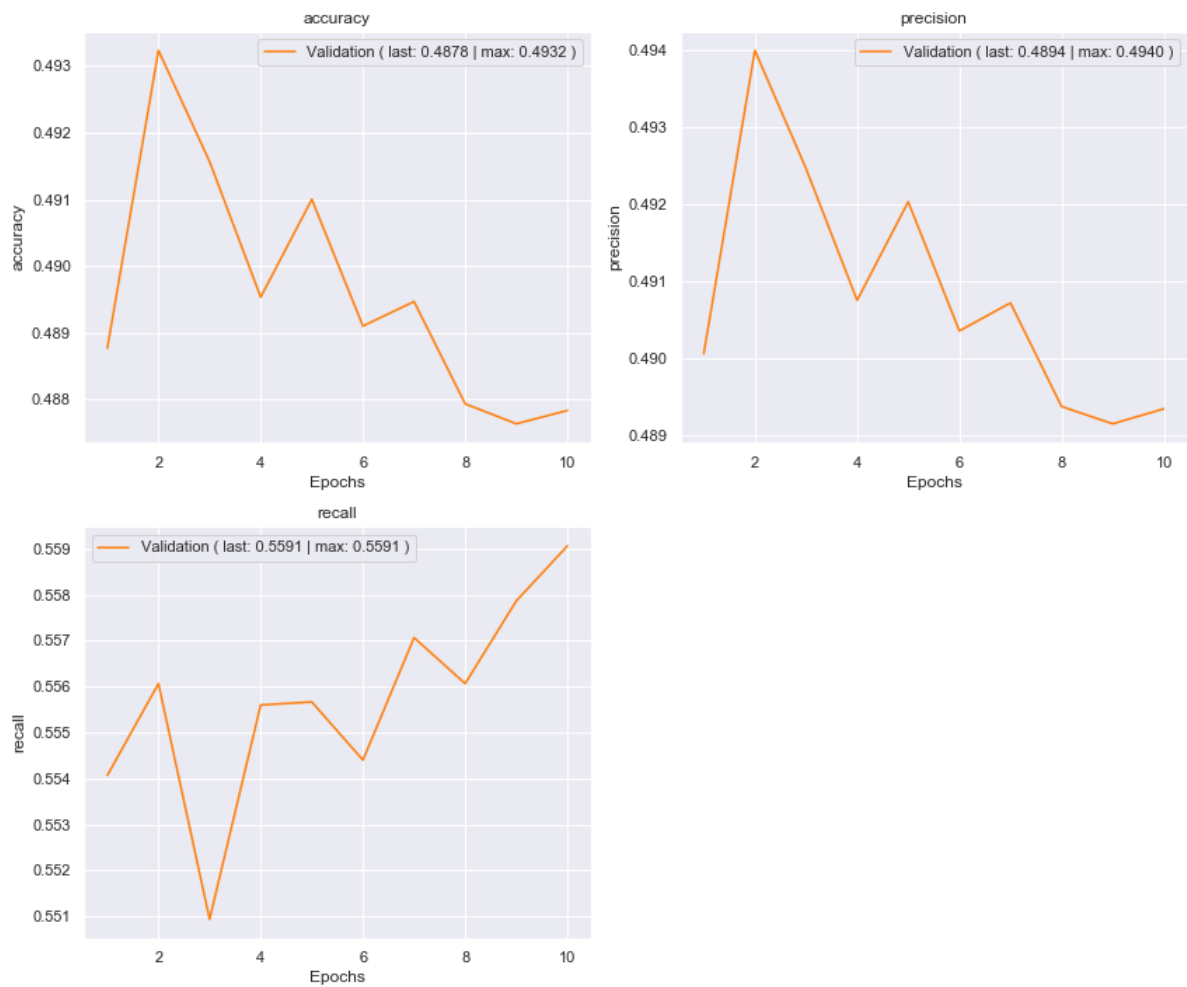


Figure 35: Accuracy, precision and recall for TF-IDF + NB.

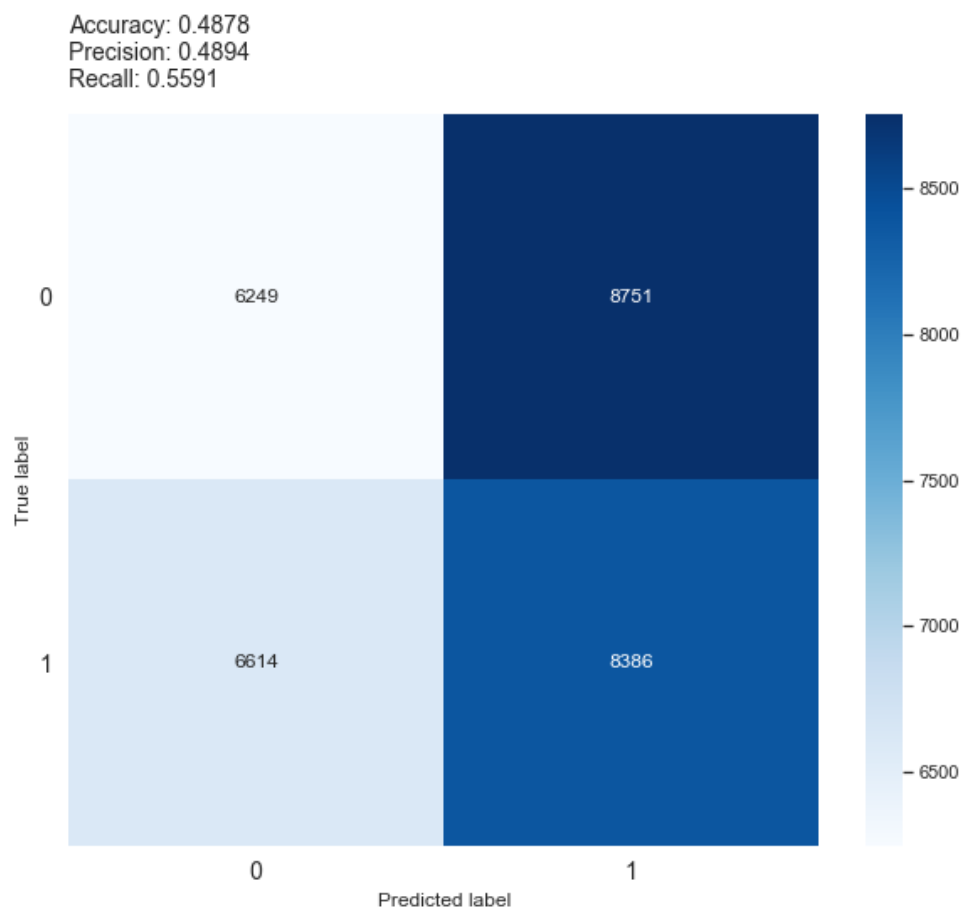


Figure 36: Validation confusion matrix for TF-IDF + NB.

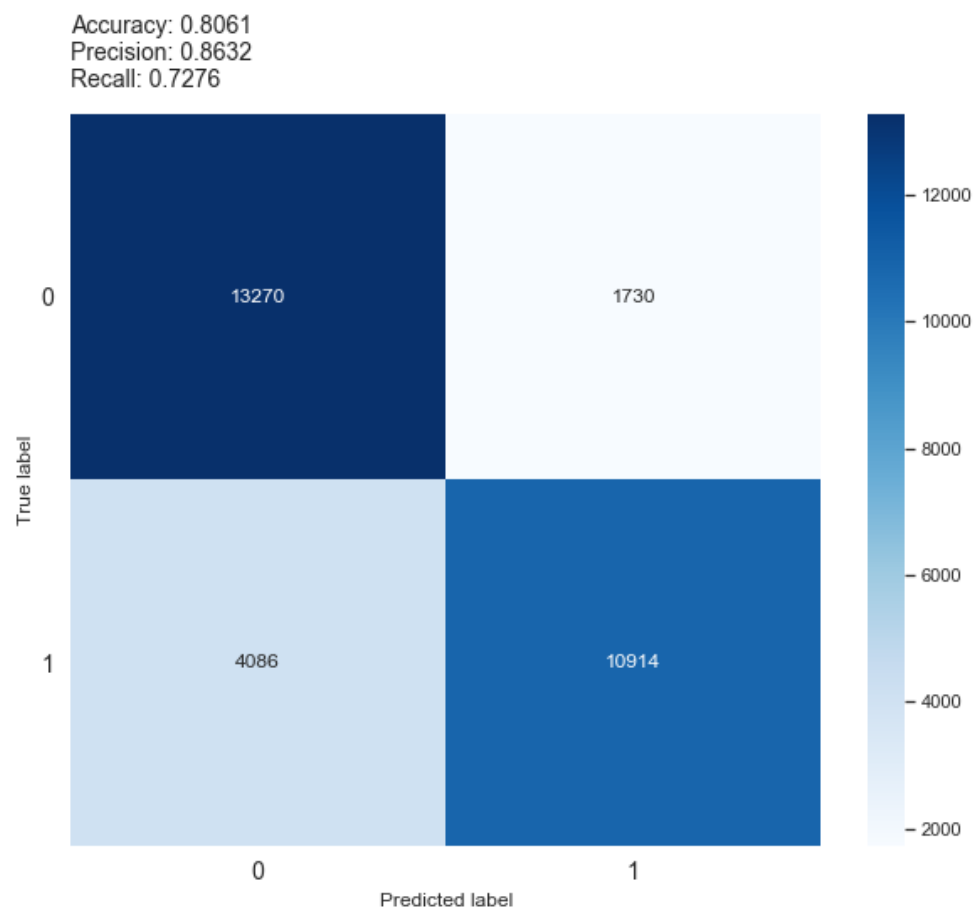


Figure 37: Validation confusion matrix for TF-IDF + CS.

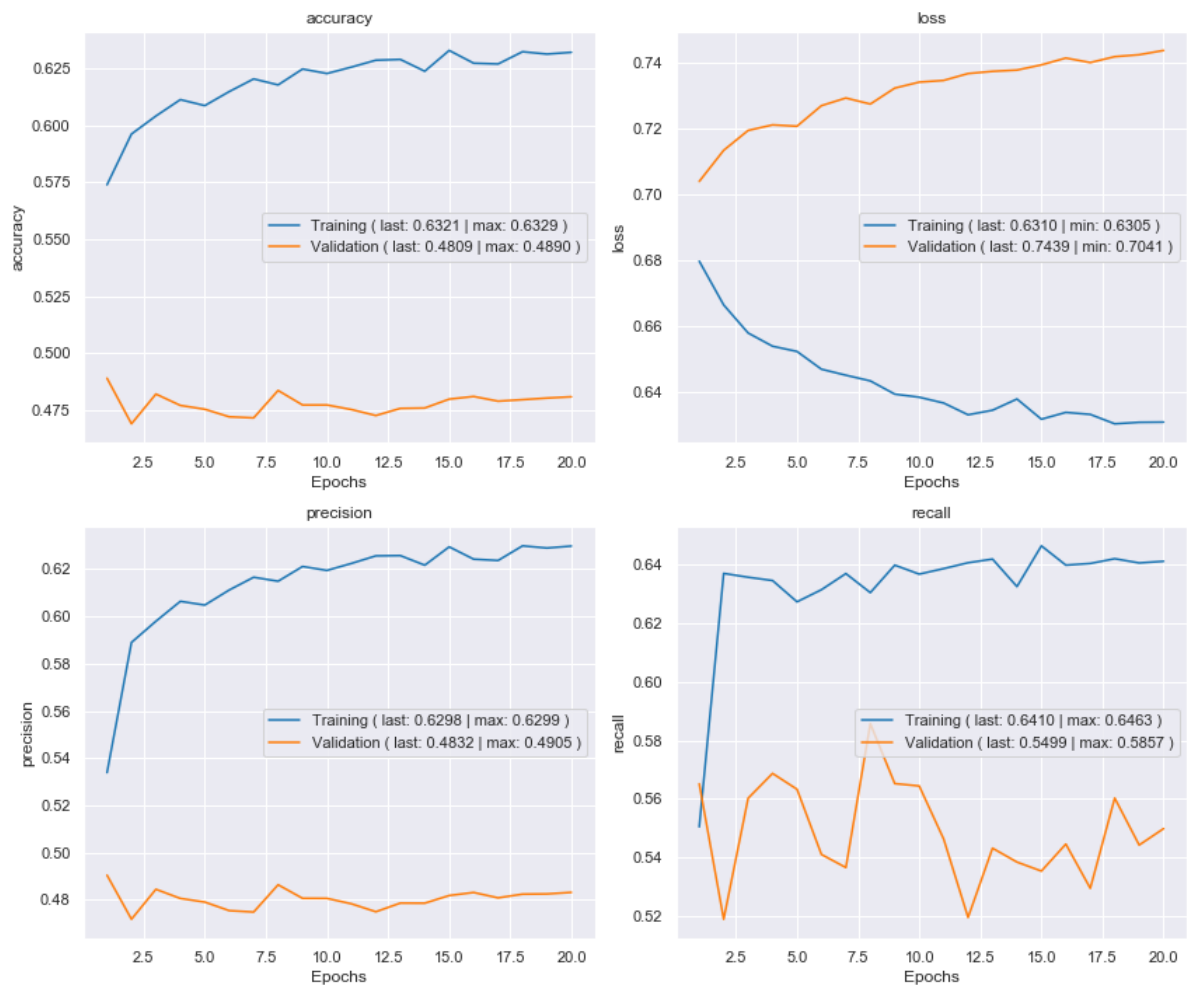


Figure 38: Accuracy, precision and recall for TF-IDF + LR.

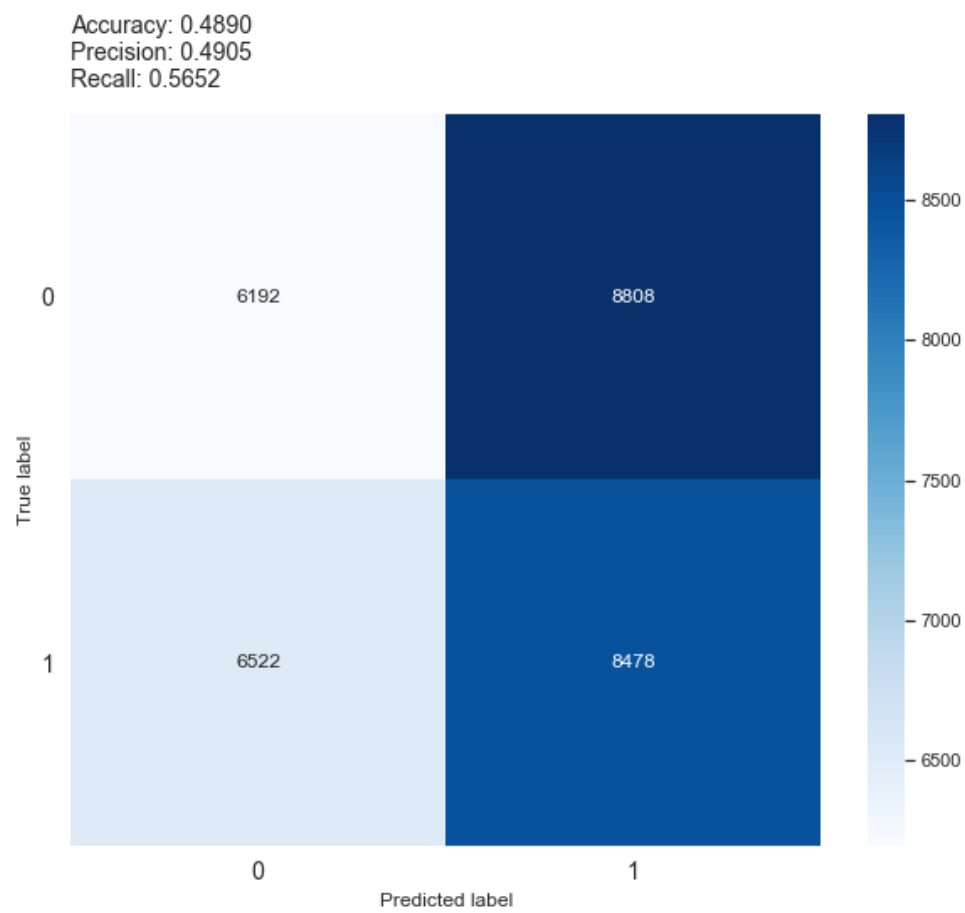


Figure 39: Validation confusion matrix for TF-IDF + LR.

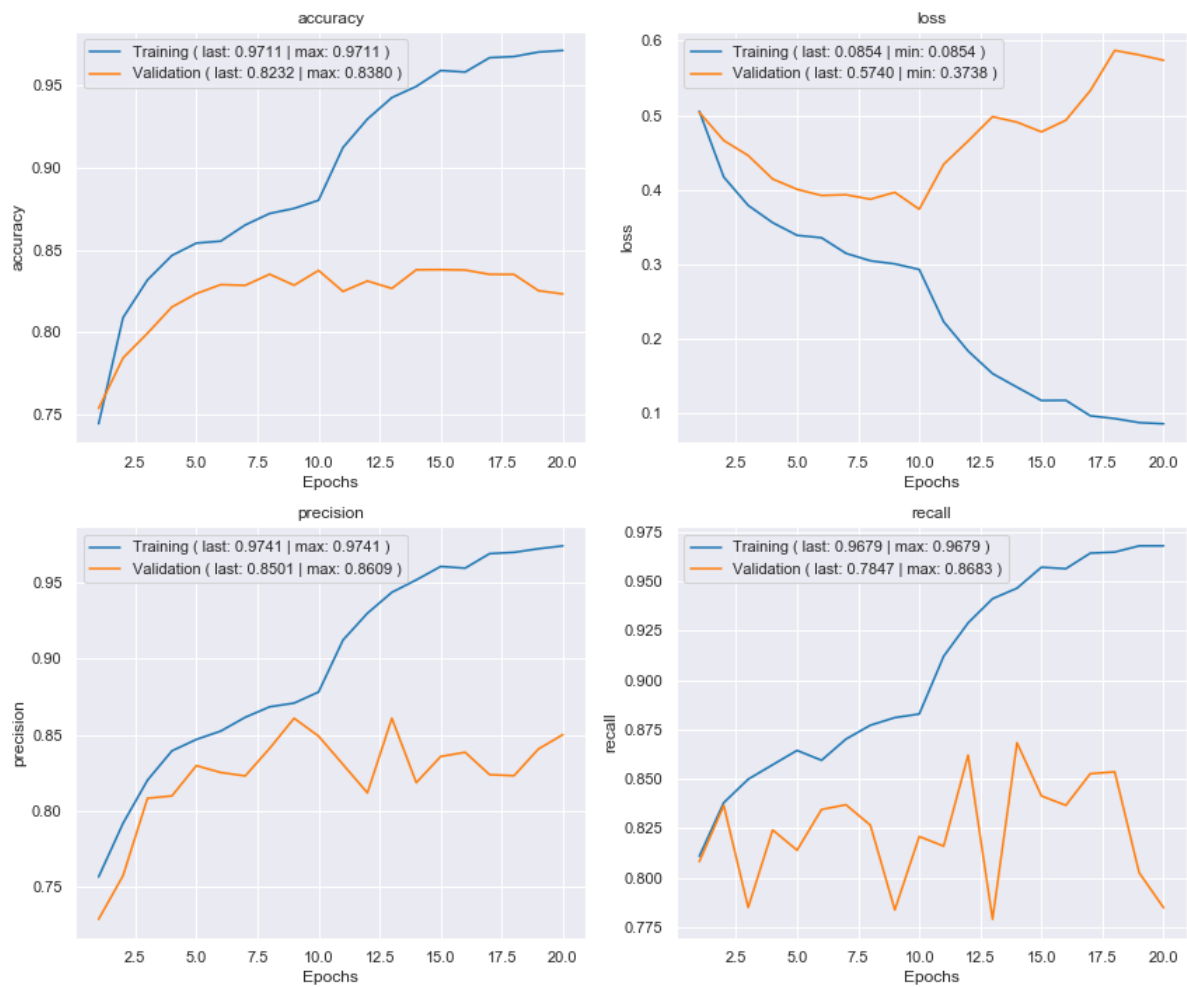


Figure 40: Accuracy, precision and recall for TF-IDF + MLP.

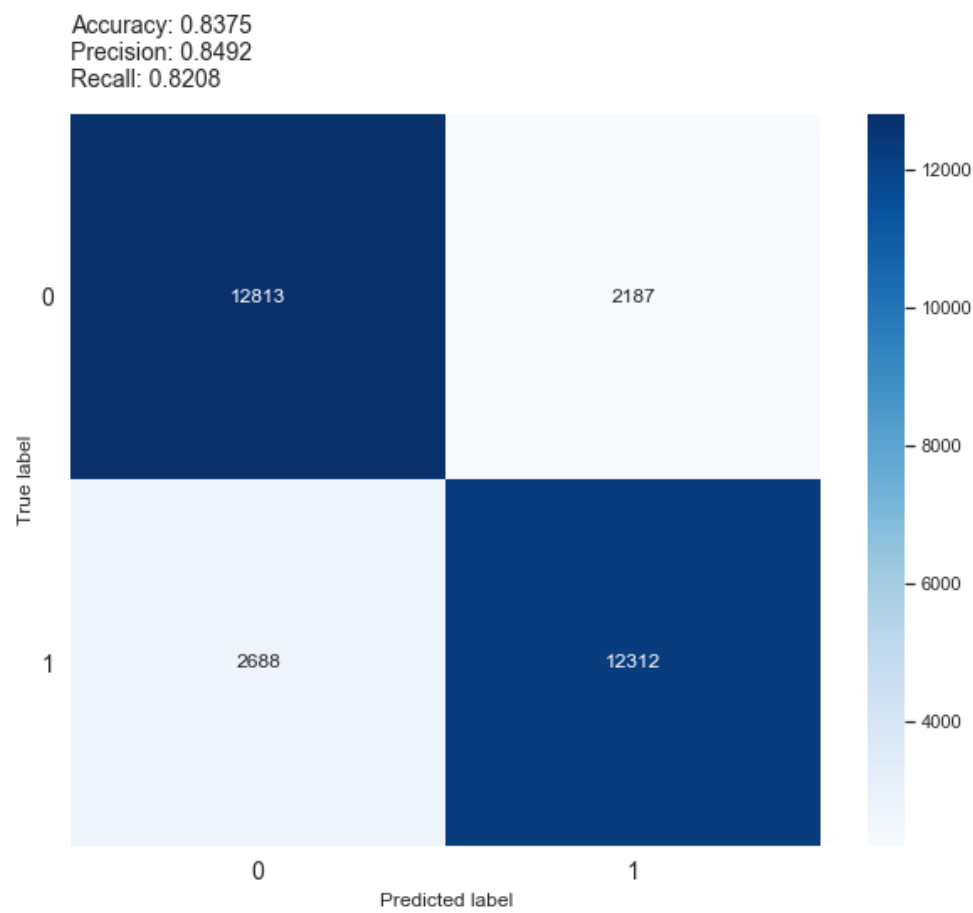


Figure 41: Validation confusion matrix for TF-IDF + MLP.

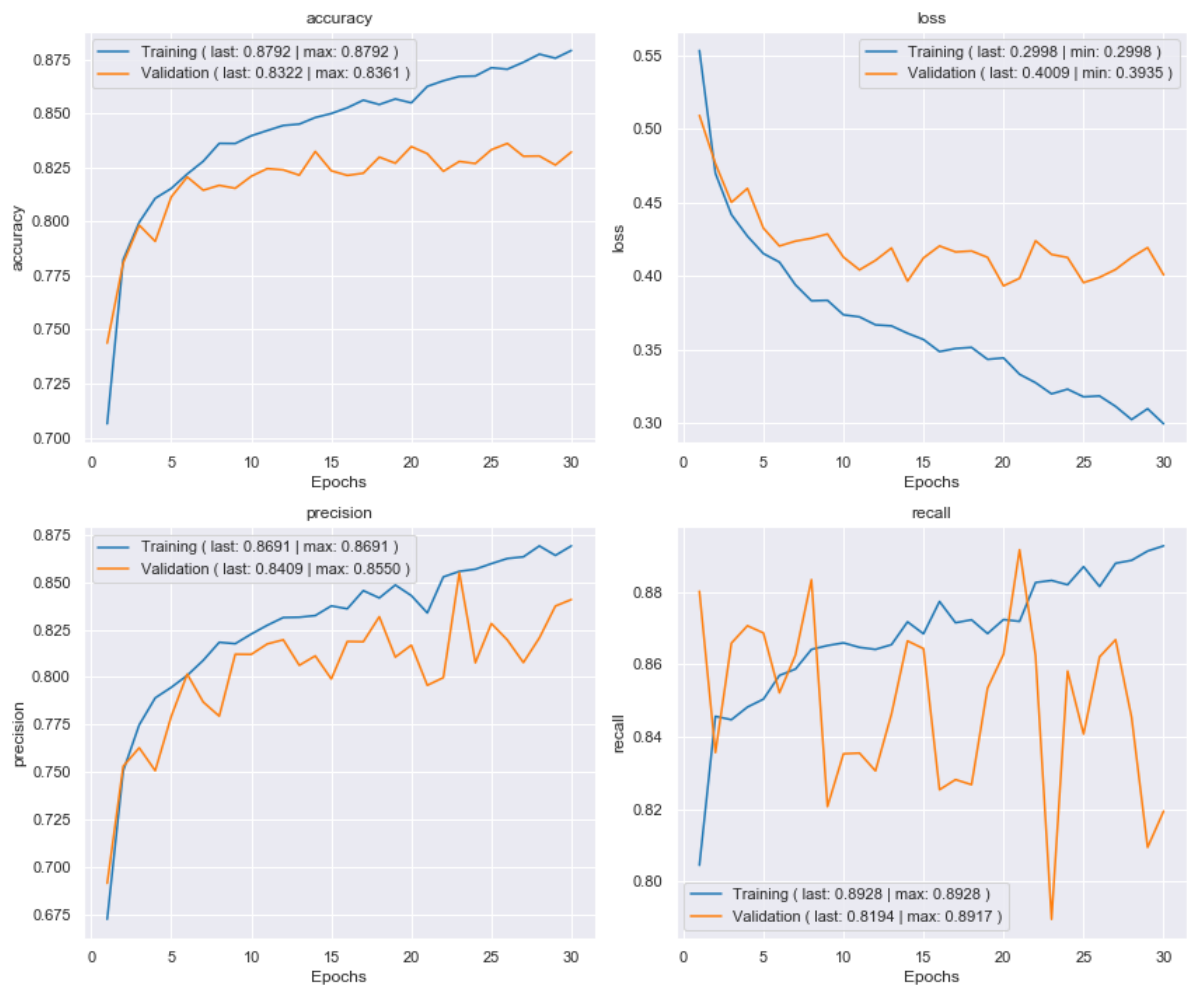


Figure 42: Accuracy, precision and recall for BoW + SNN + MLP + Output MLP.

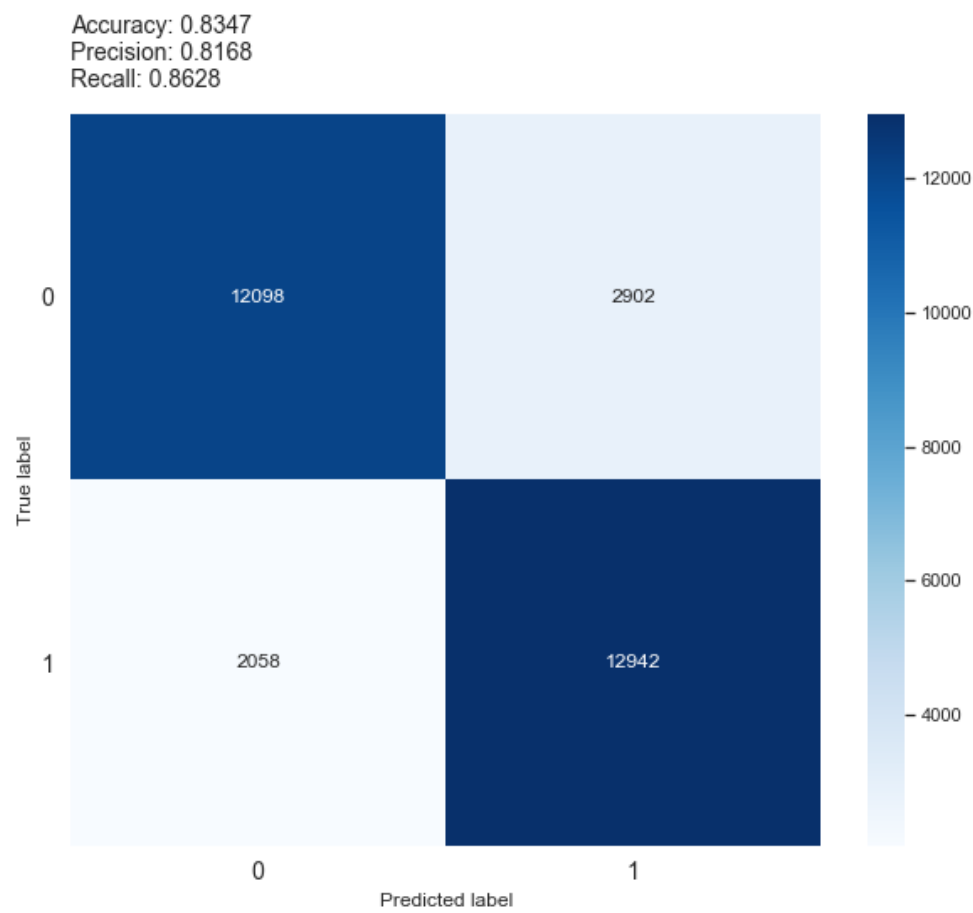


Figure 43: Validation confusion matrix for BoW + SNN + MLP + Output MLP.

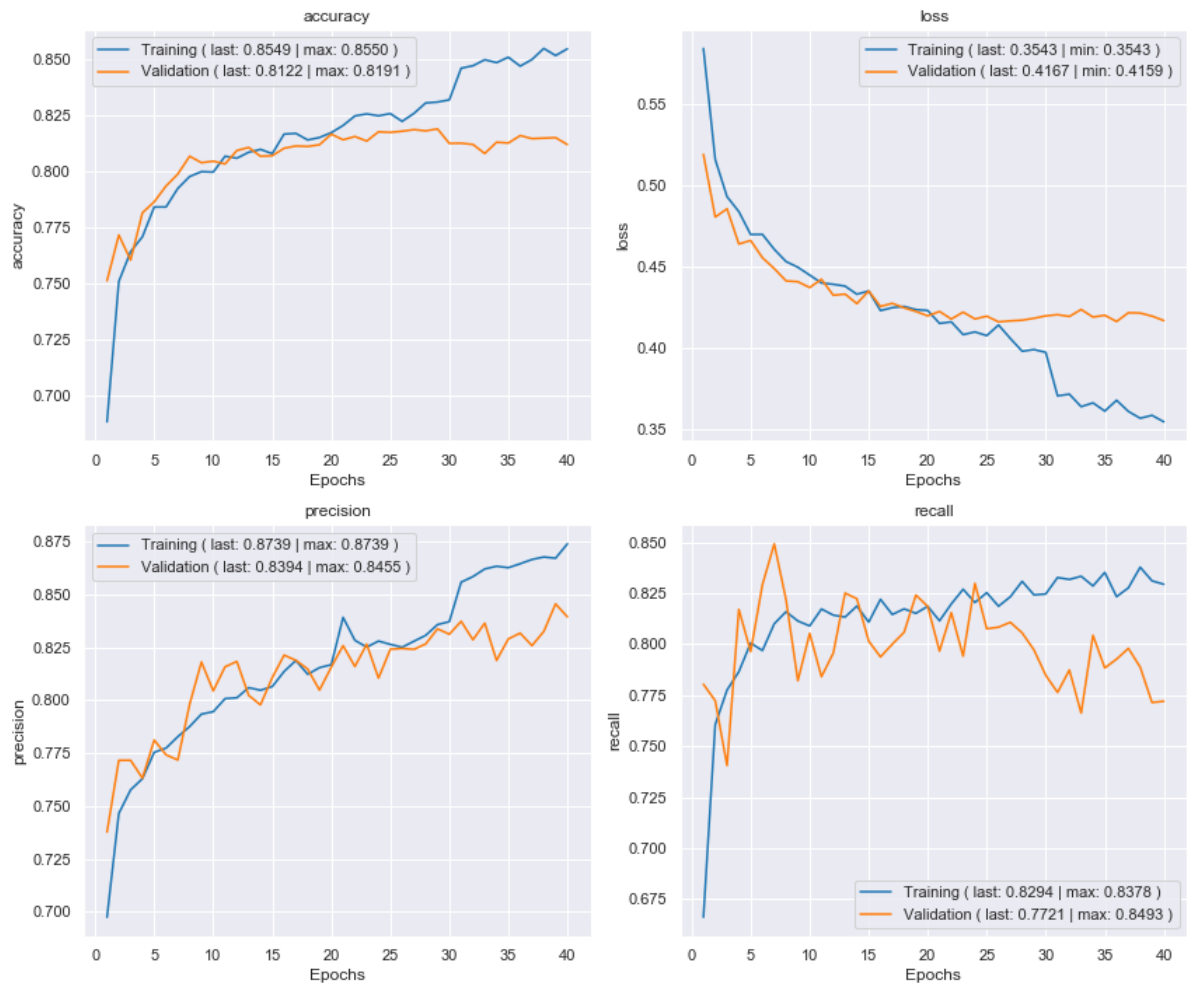


Figure 44: Accuracy, precision and recall for BoW + SNN + MLP + Output Manhattan.

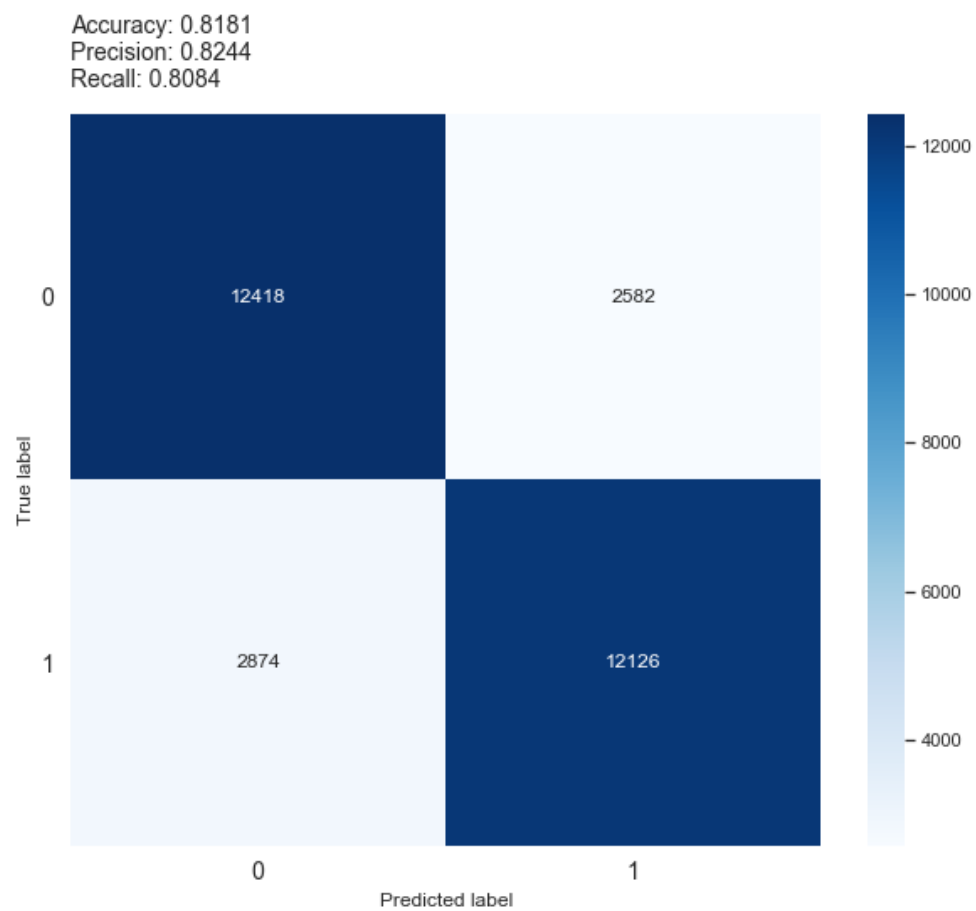


Figure 45: Validation confusion matrix for BoW + SNN + MLP + Output Manhattan.

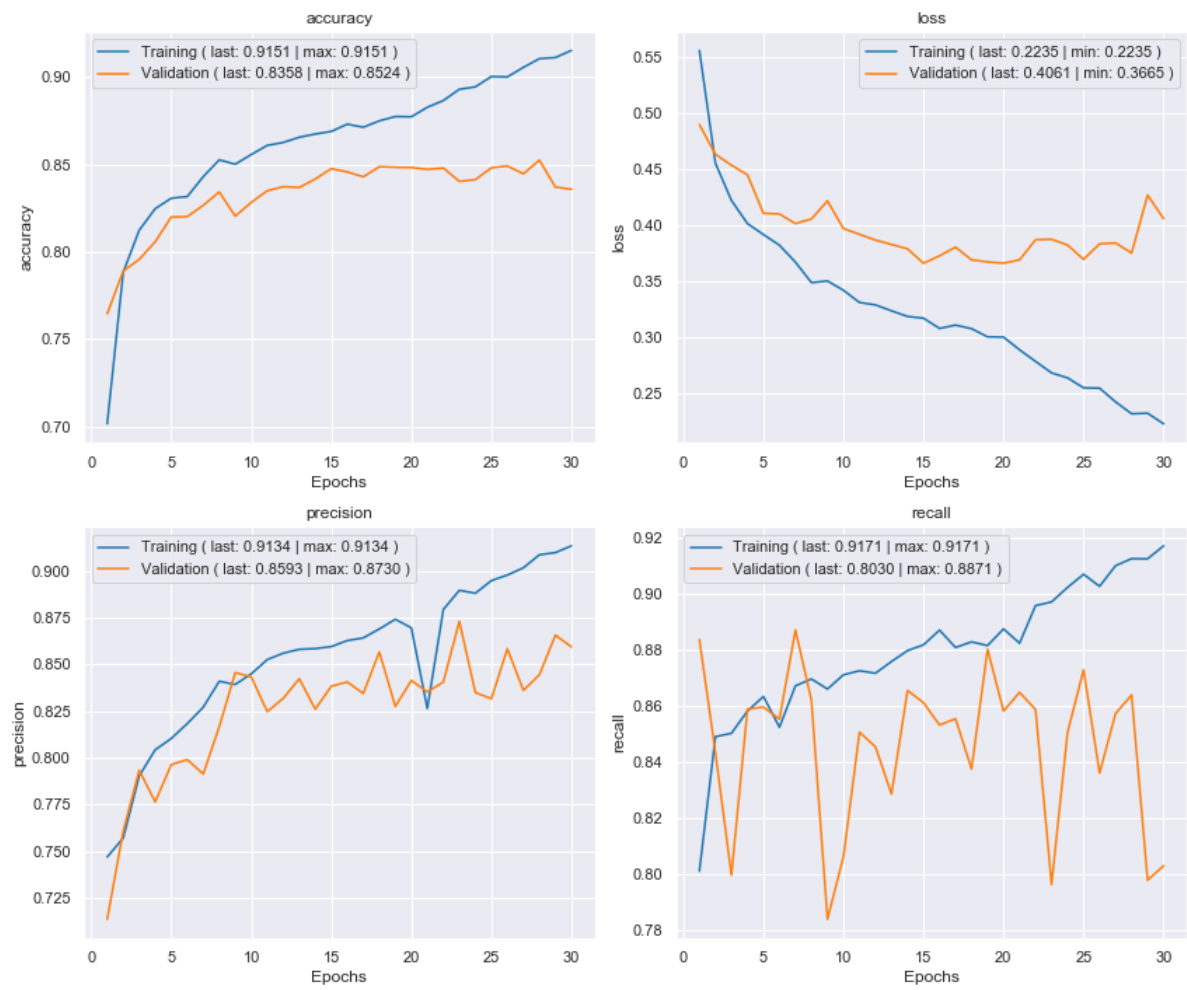


Figure 46: Accuracy, precision and recall for TF-IDF + SNN + MLP + Output MLP.

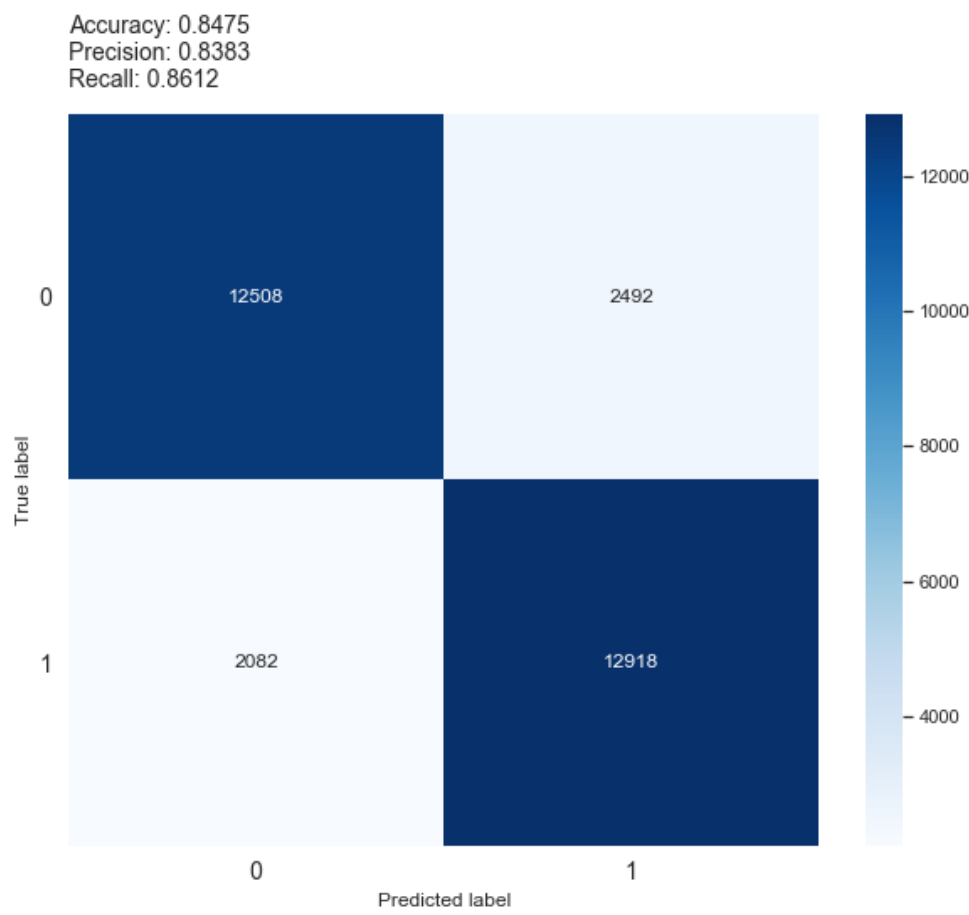


Figure 47: Validation confusion matrix for TF-IDF + SNN + MLP + Output MLP.



Figure 48: Accuracy, precision and recall for TF-IDF + SNN + MLP + Output Manhattan.

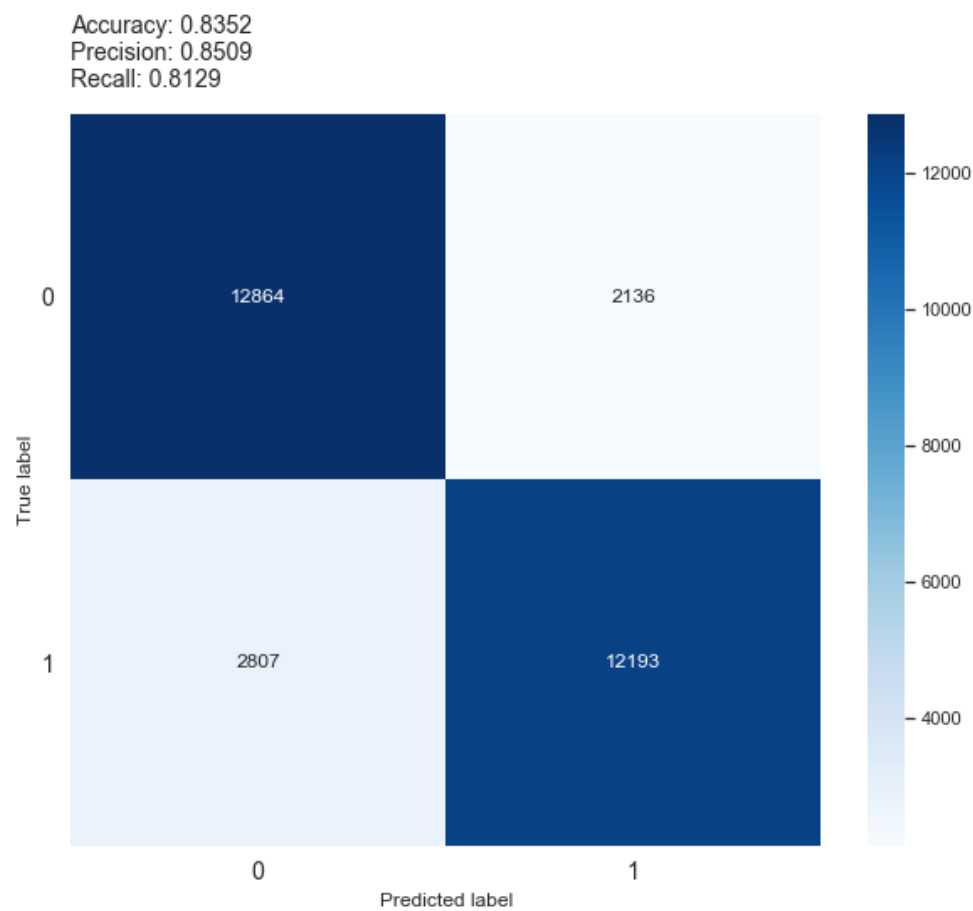


Figure 49: Validation confusion matrix for TF-IDF + SNN + MLP + Output Manhattan.

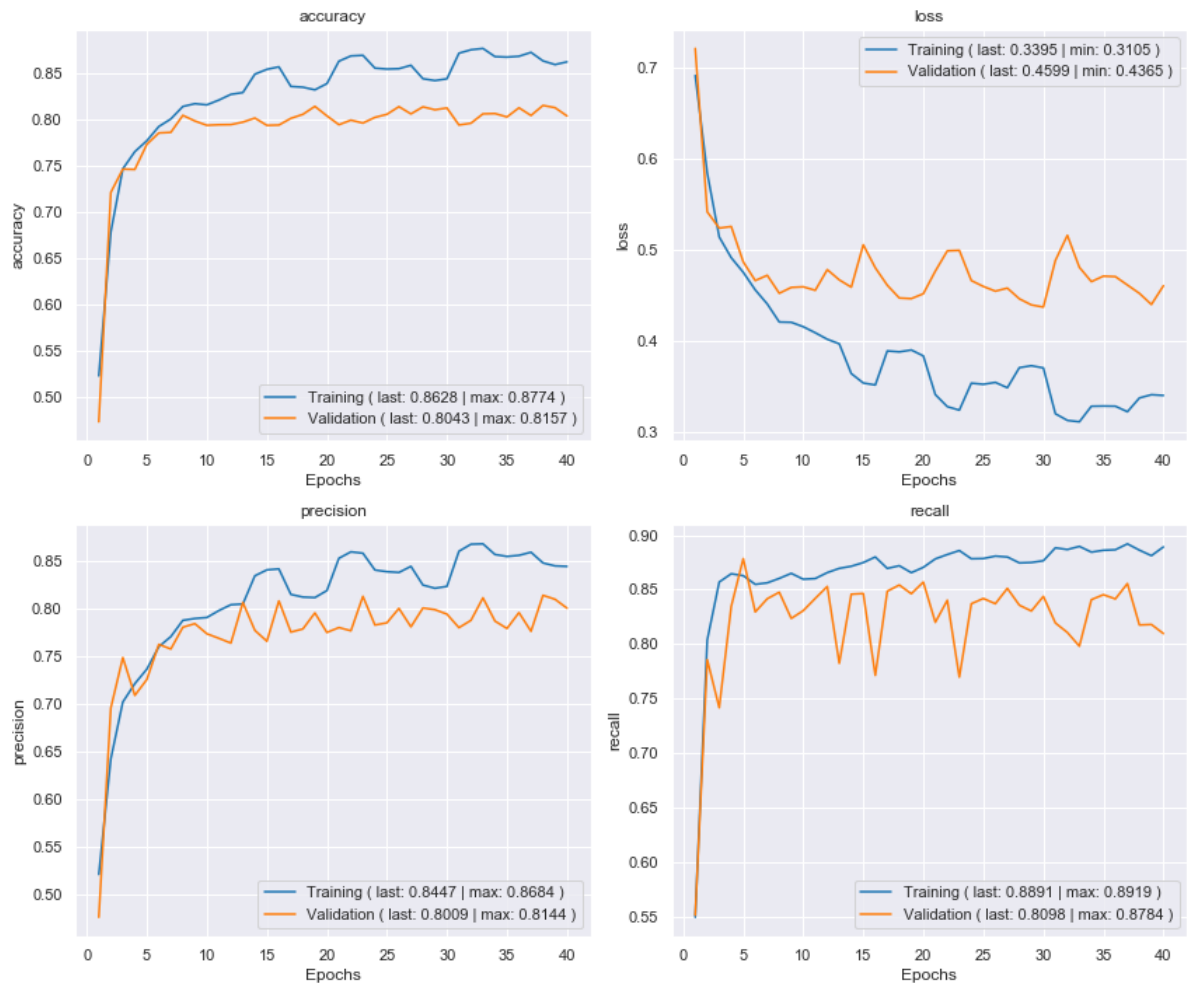


Figure 50: Accuracy, precision and recall for WE + SNN + LSTM + Output MLP.

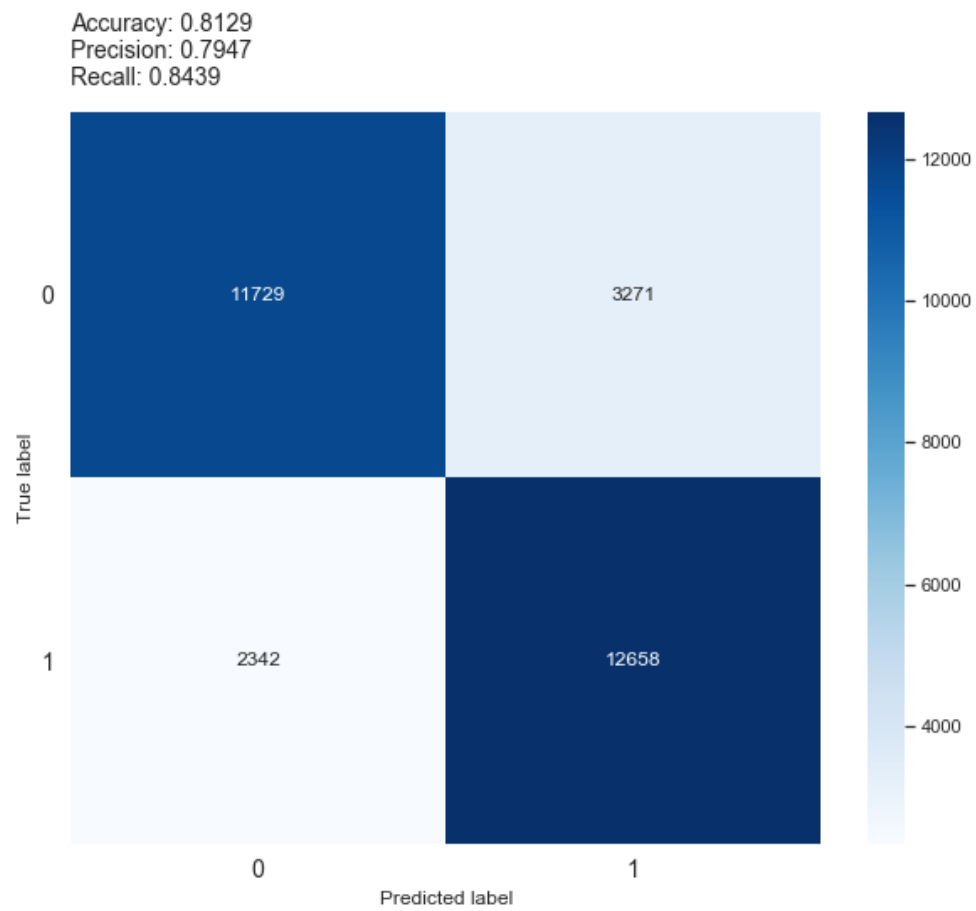


Figure 51: Validation confusion matrix for WE + SNN + LSTM + Output MLP.

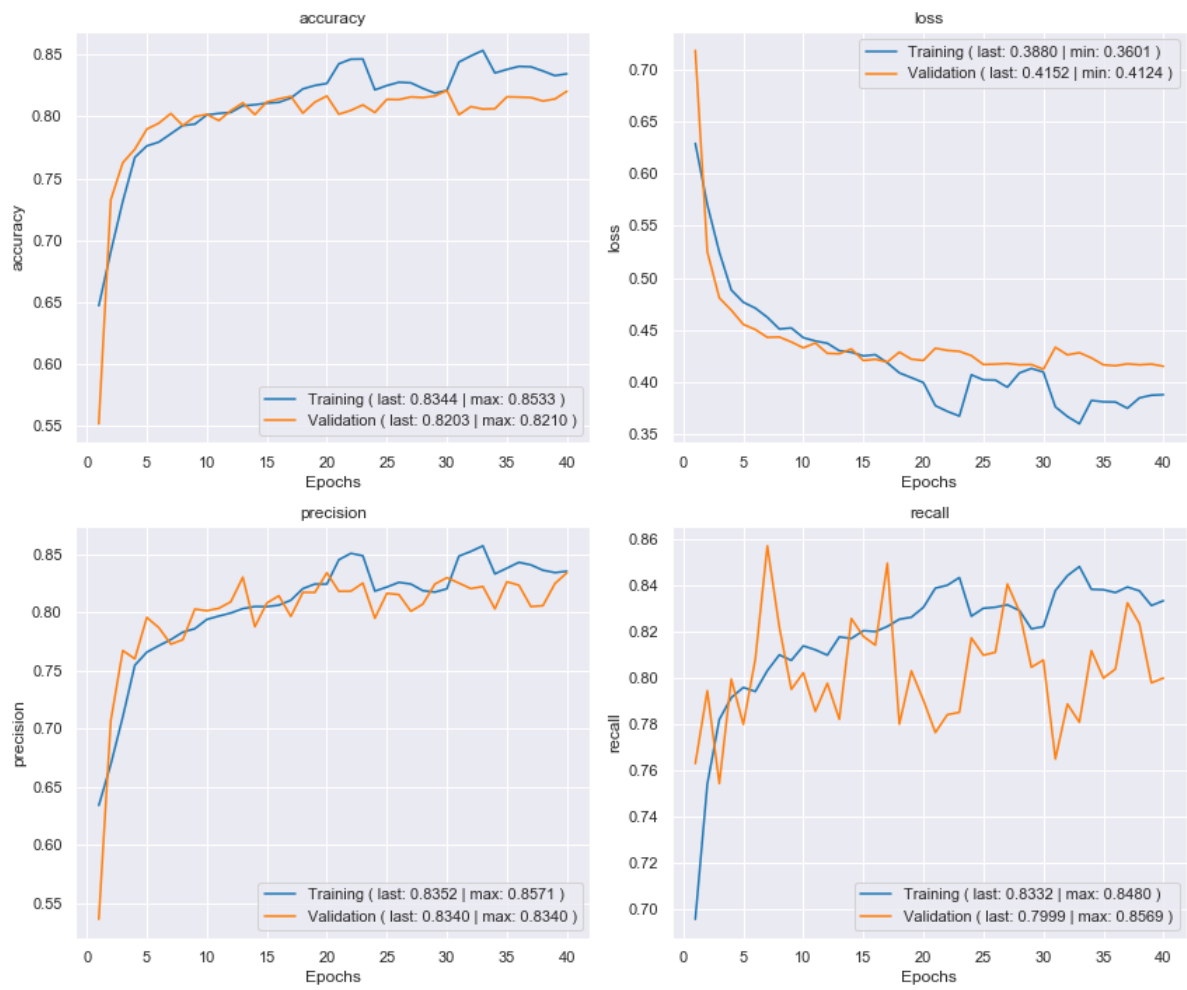


Figure 52: Accuracy, precision and recall for WE + SNN + LSTM + Output Manhattan.

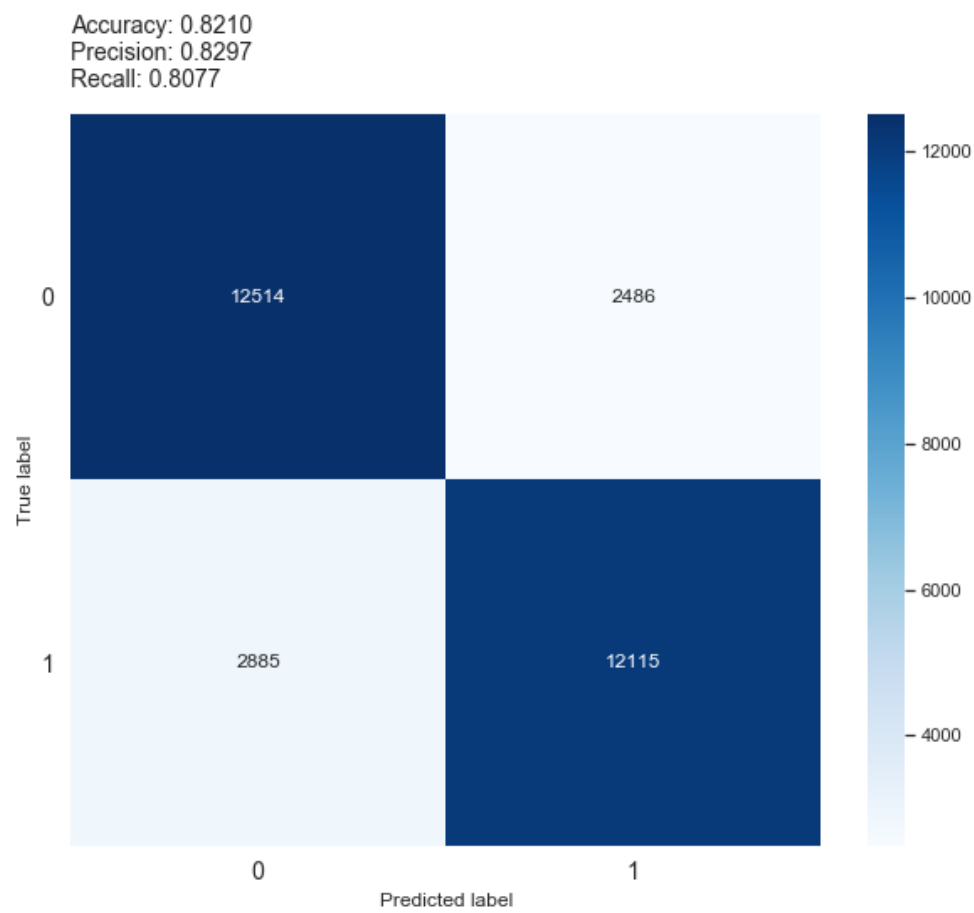


Figure 53: Validation confusion matrix for WE + SNN + LSTM + Output Manhattan.

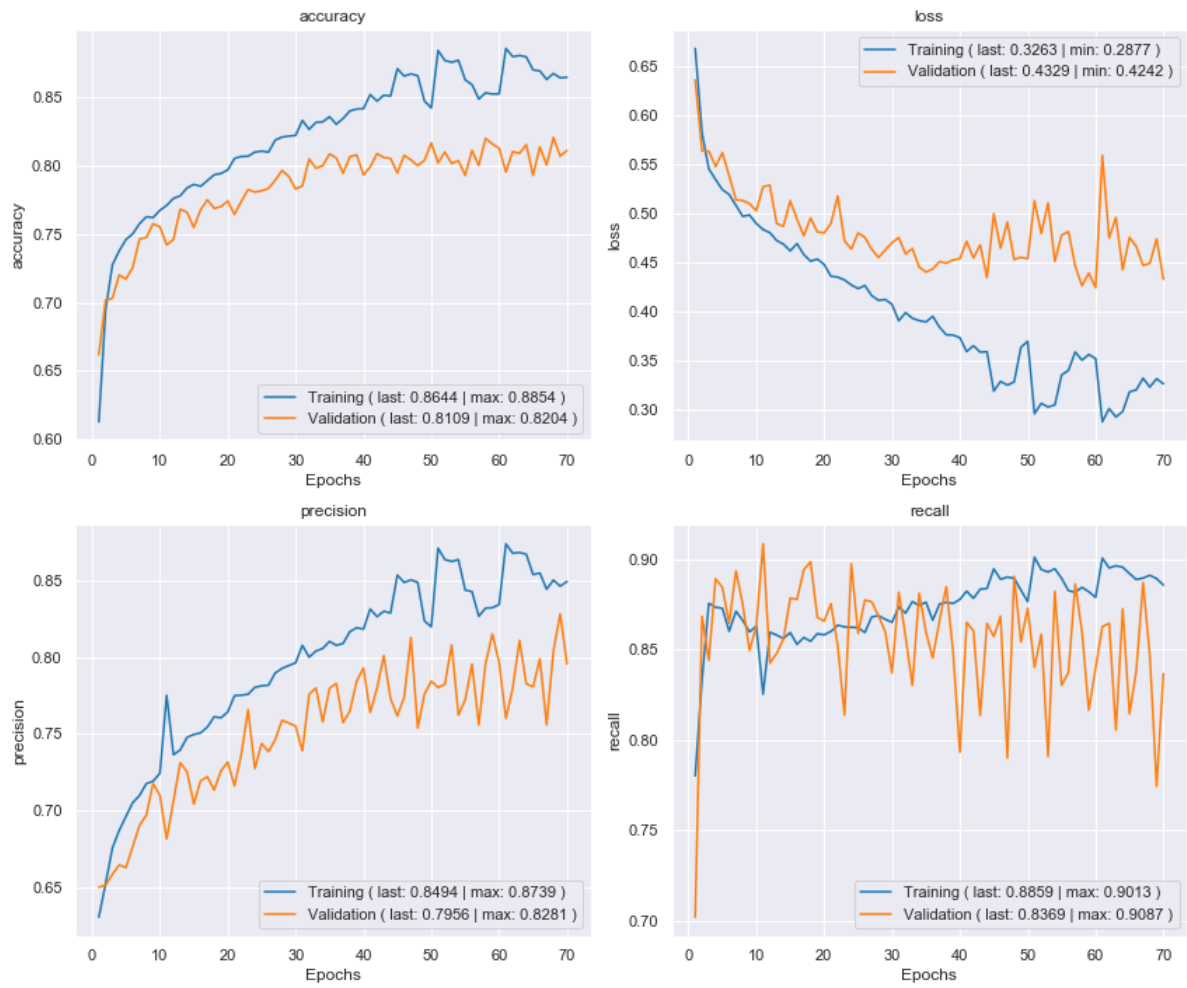


Figure 54: Accuracy, precision and recall for WE + SNN + TN + Output MLP.

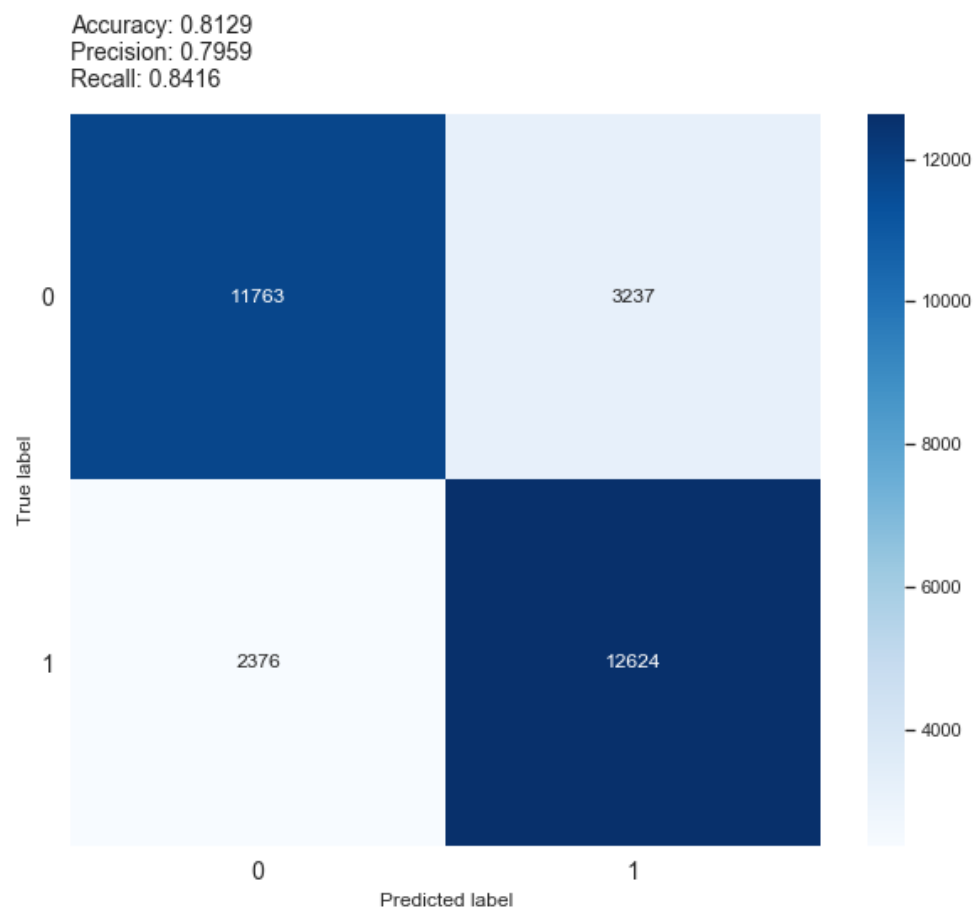


Figure 55: Validation confusion matrix for WE + SNN + TN + Output MLP.

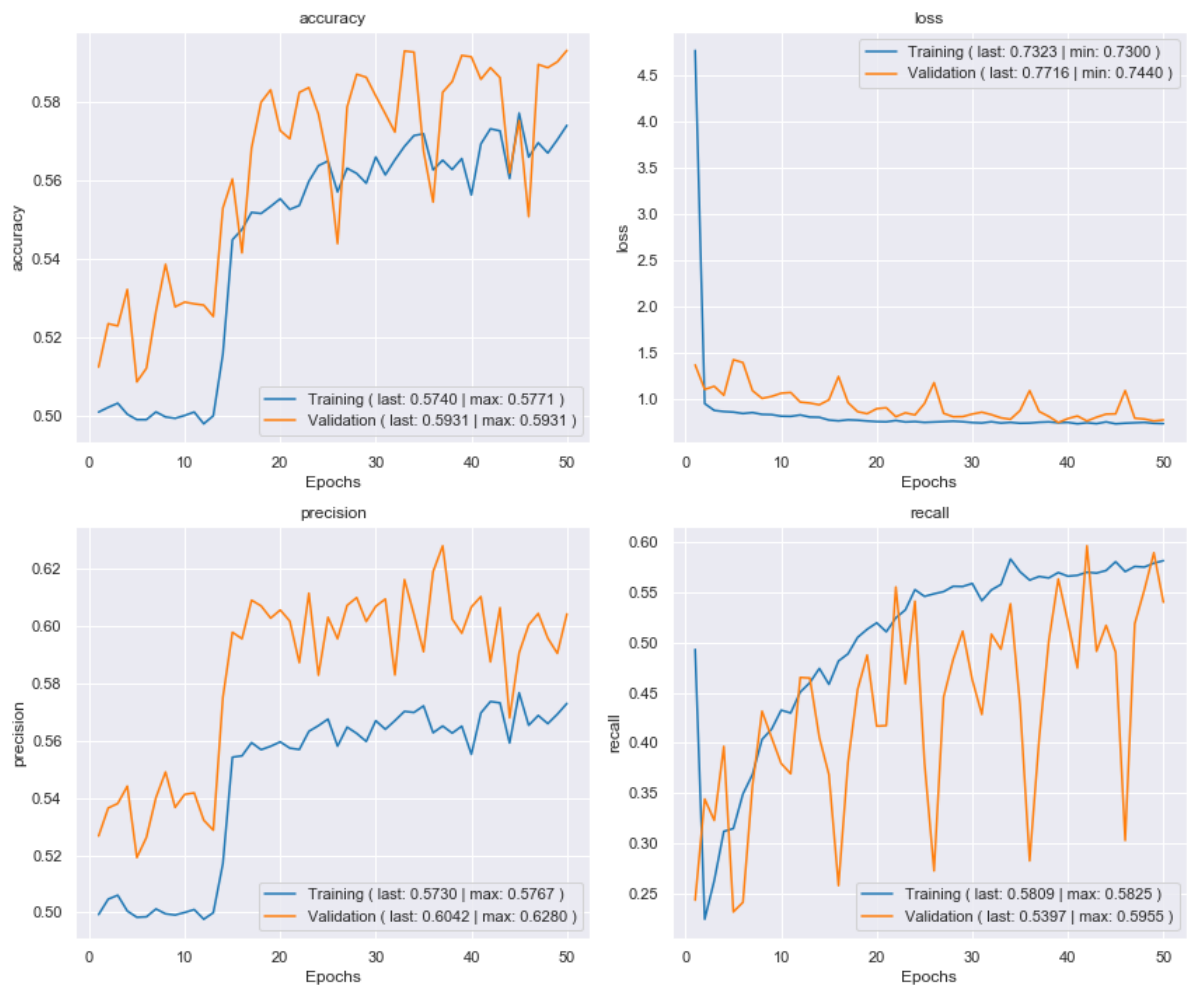


Figure 56: Accuracy, precision and recall for WE + SNN + TN + Output Manhattan.

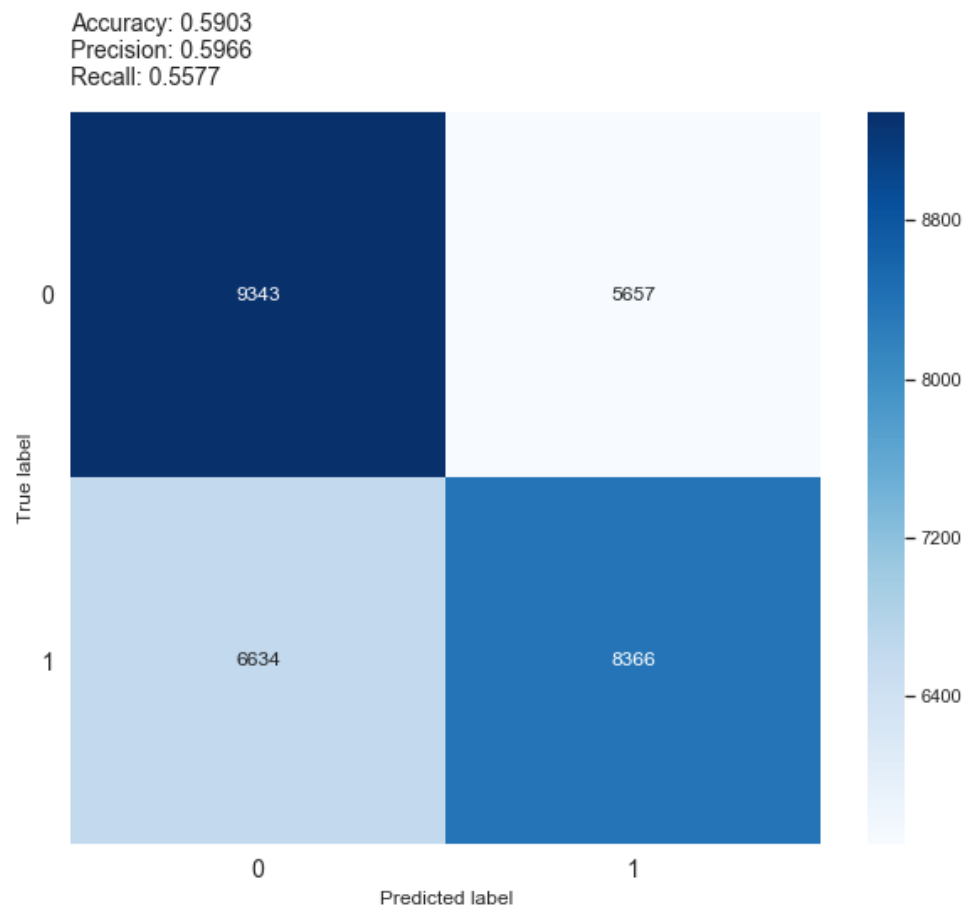


Figure 57: Validation confusion matrix for WE + SNN + TN + Output Manhattan.

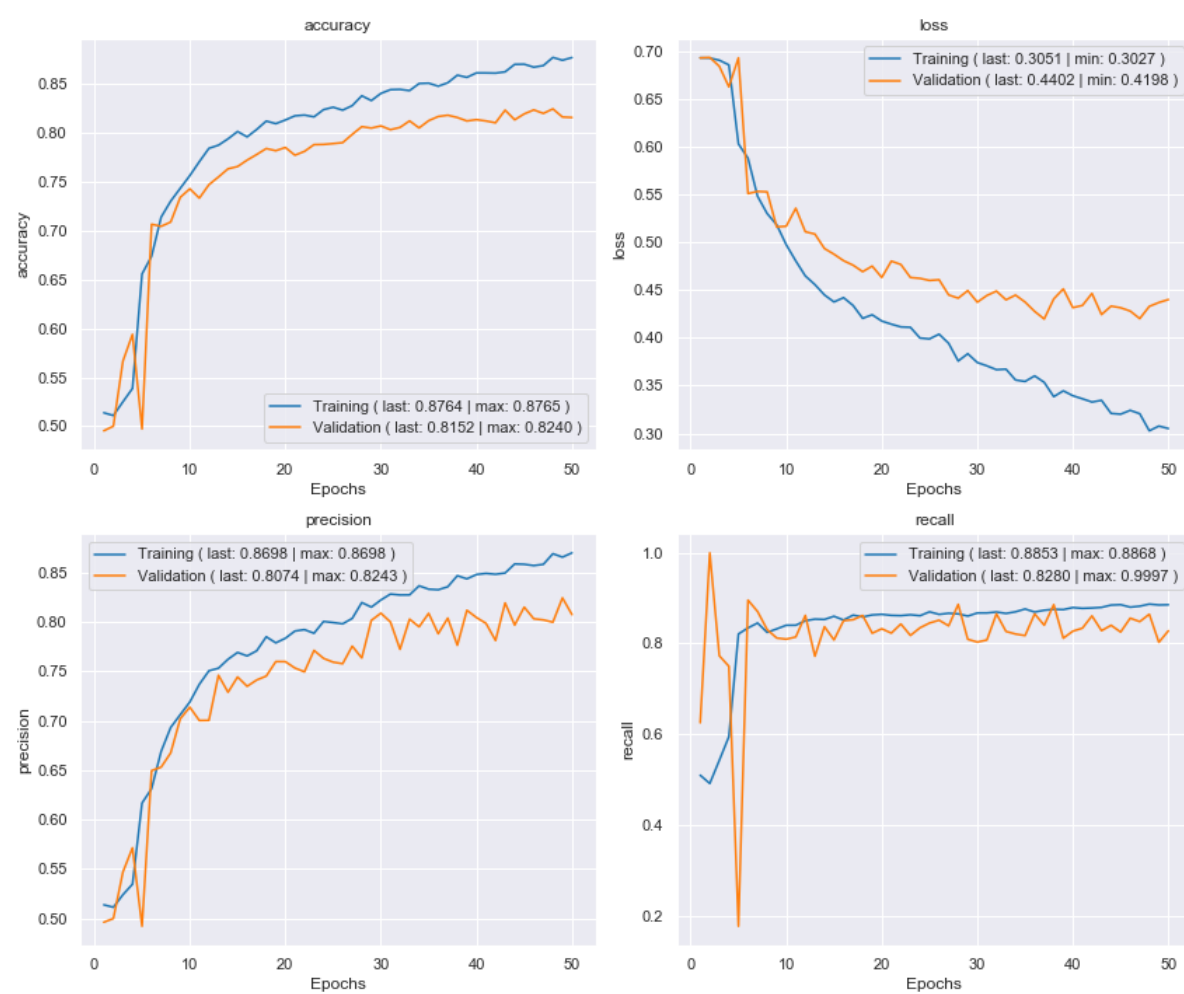


Figure 58: Accuracy, precision and recall for Word2Vec + SNN + LSTM + Output MLP.

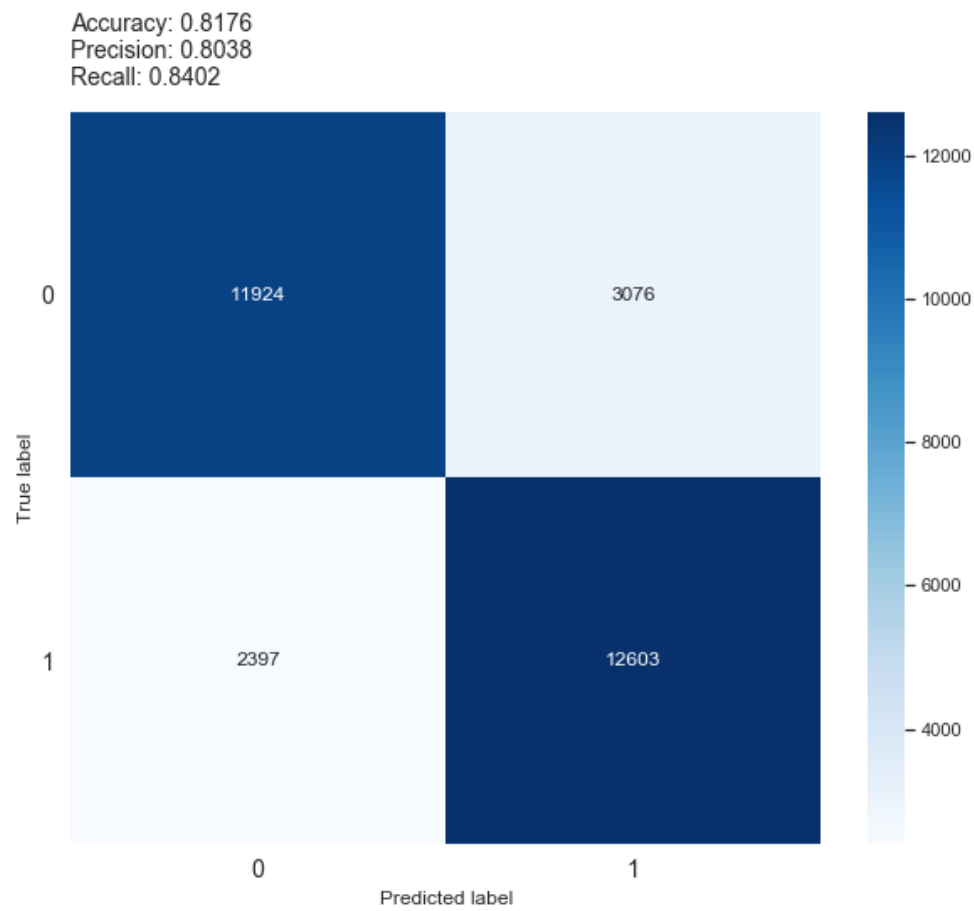


Figure 59: Validation confusion matrix for Word2Vec + SNN + LSTM + Output MLP.

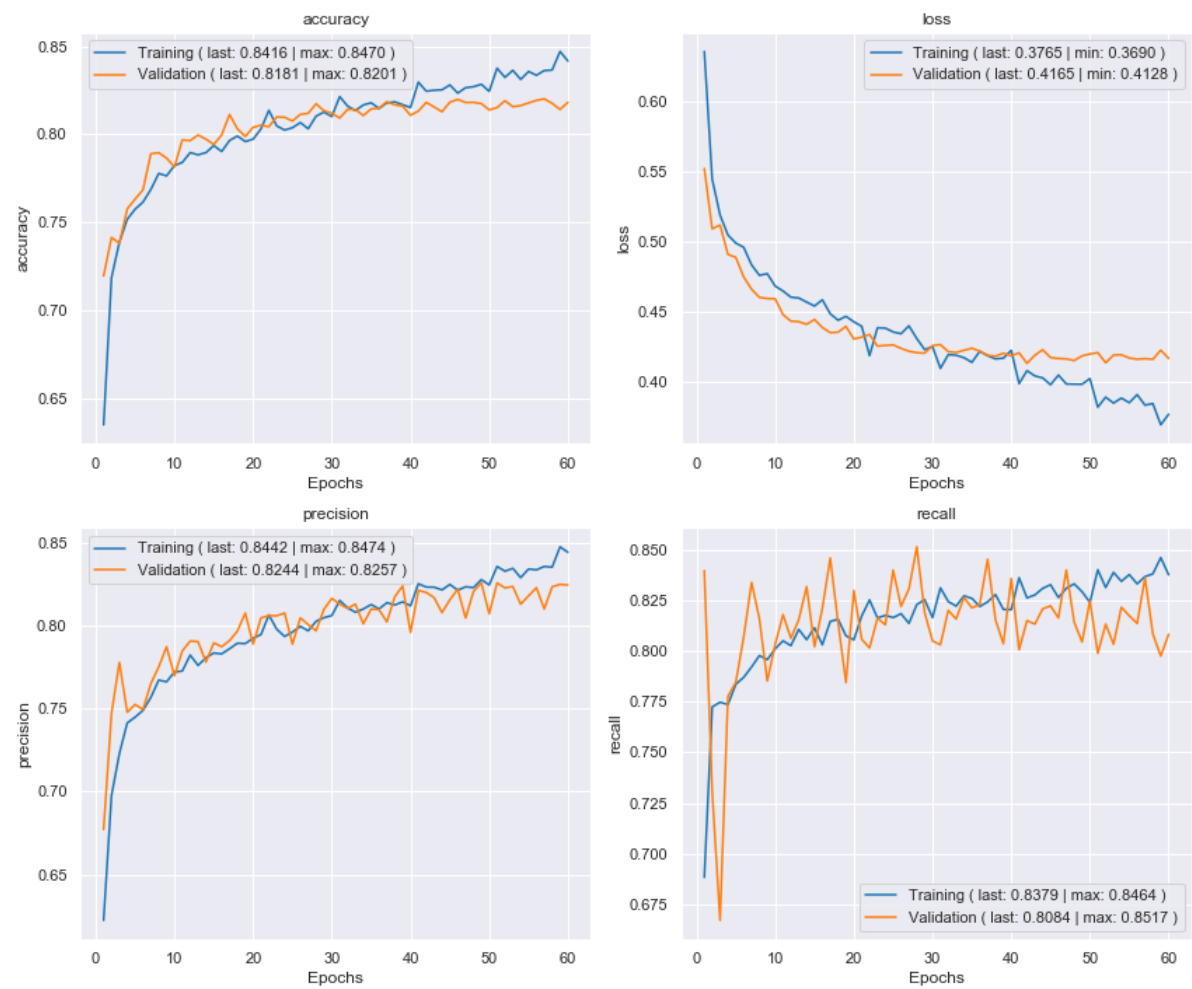


Figure 60: Accuracy, precision and recall for Word2Vec + SNN + LSTM + Output Manhattan.

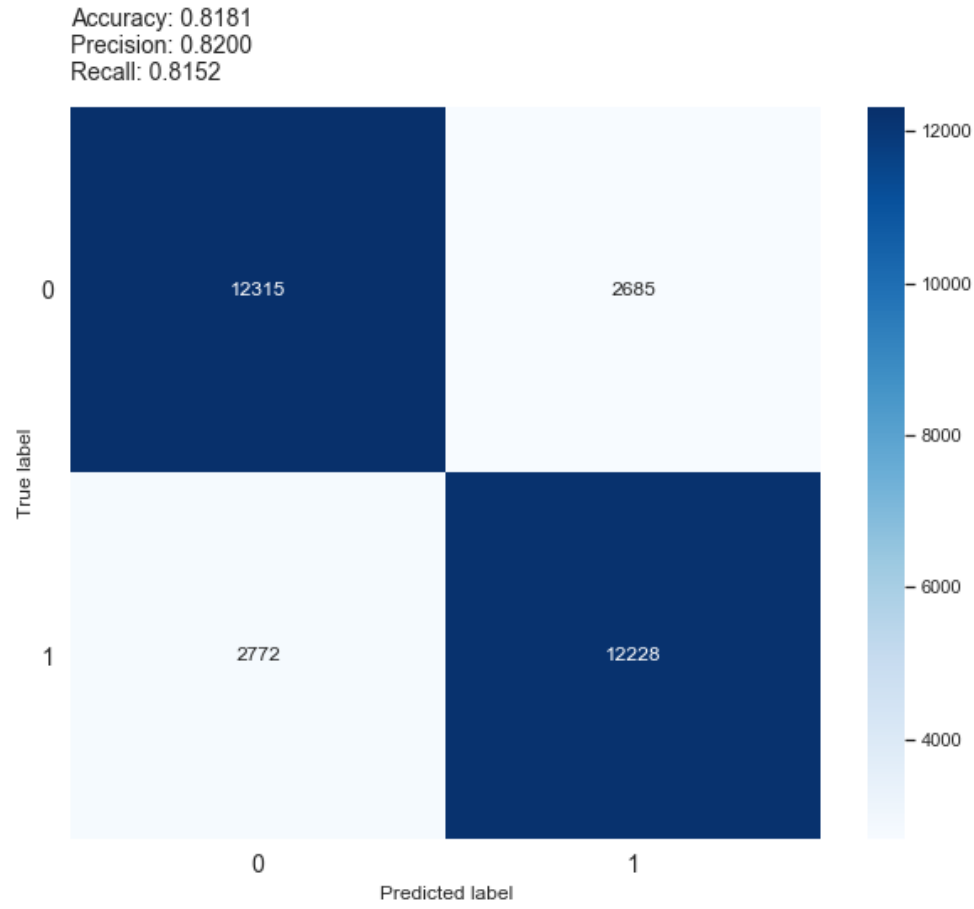


Figure 61: Validation confusion matrix for Word2Vec + SNN + LSTM + Output Manhattan.

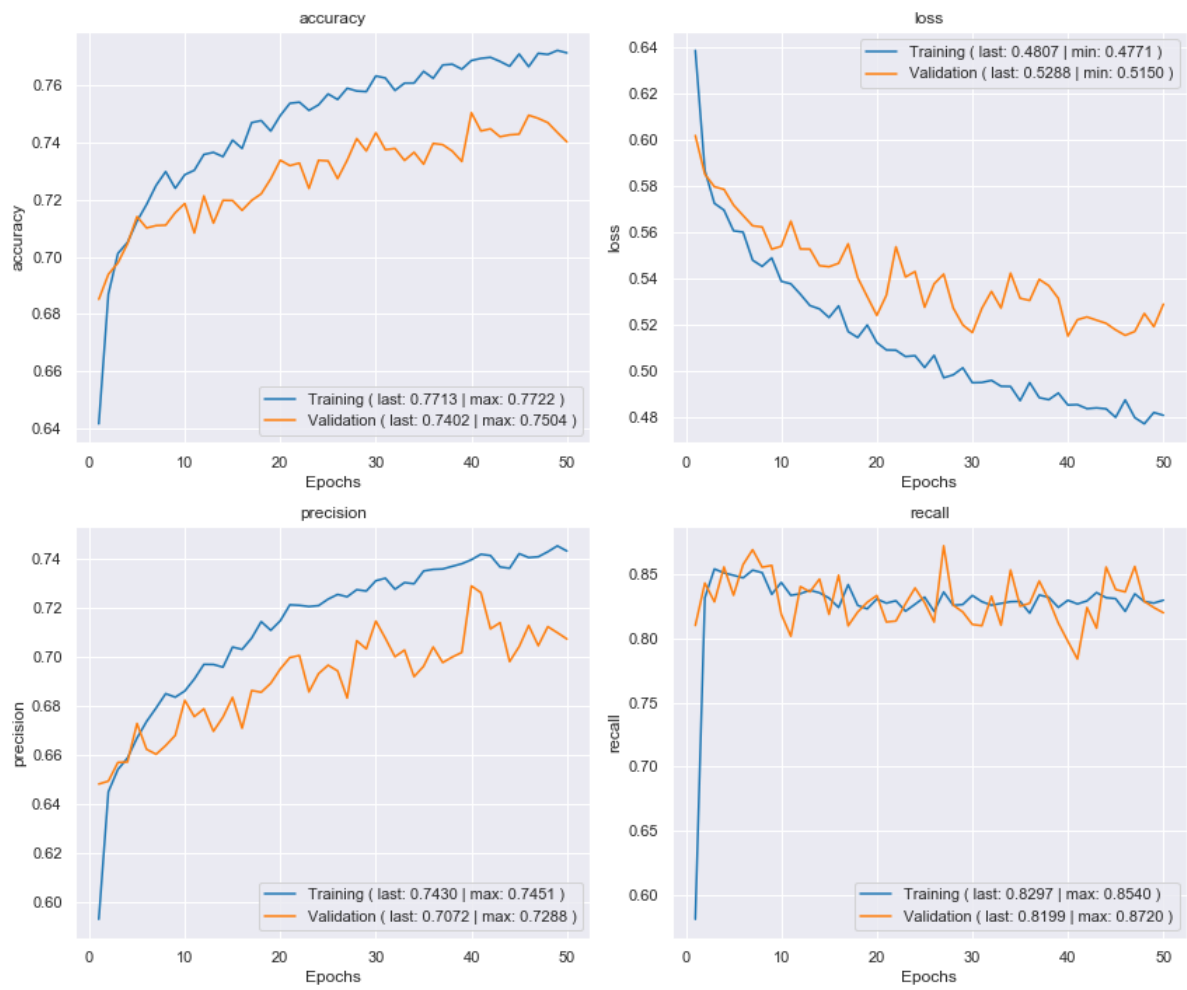


Figure 62: Accuracy, precision and recall for Word2Vec + SNN + TN + Output MLP.

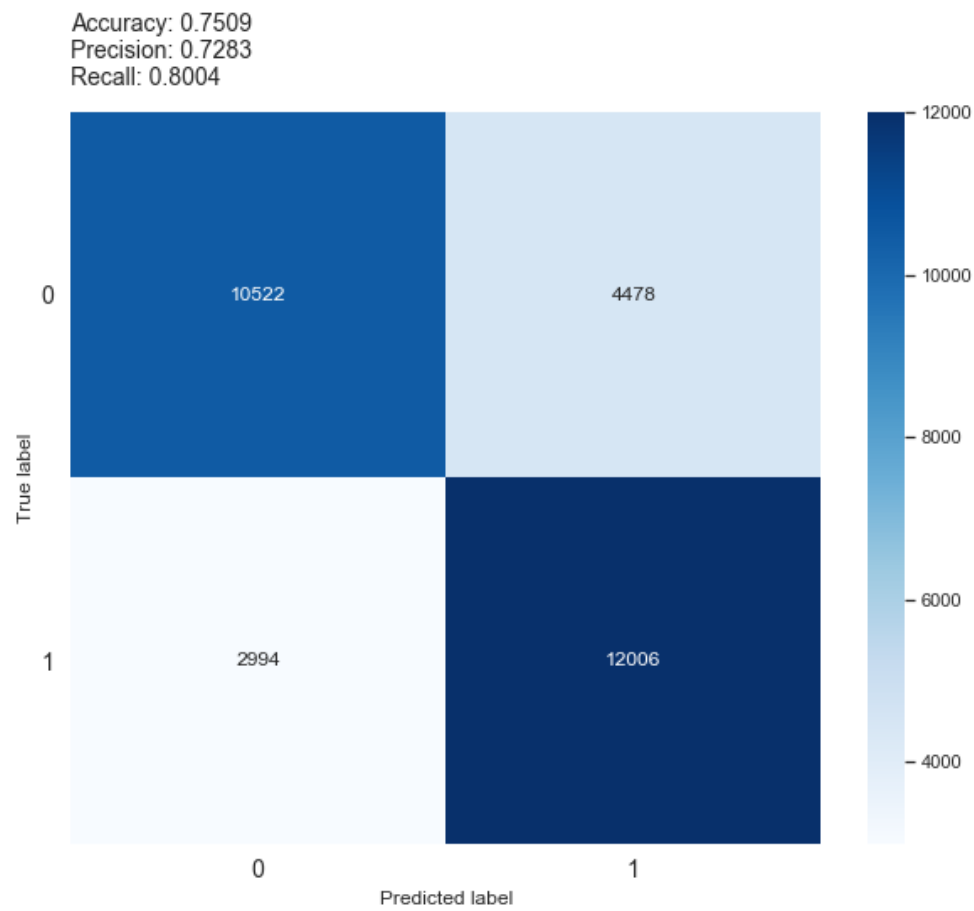


Figure 63: Validation confusion matrix for Word2Vec + SNN + TN + Output MLP.

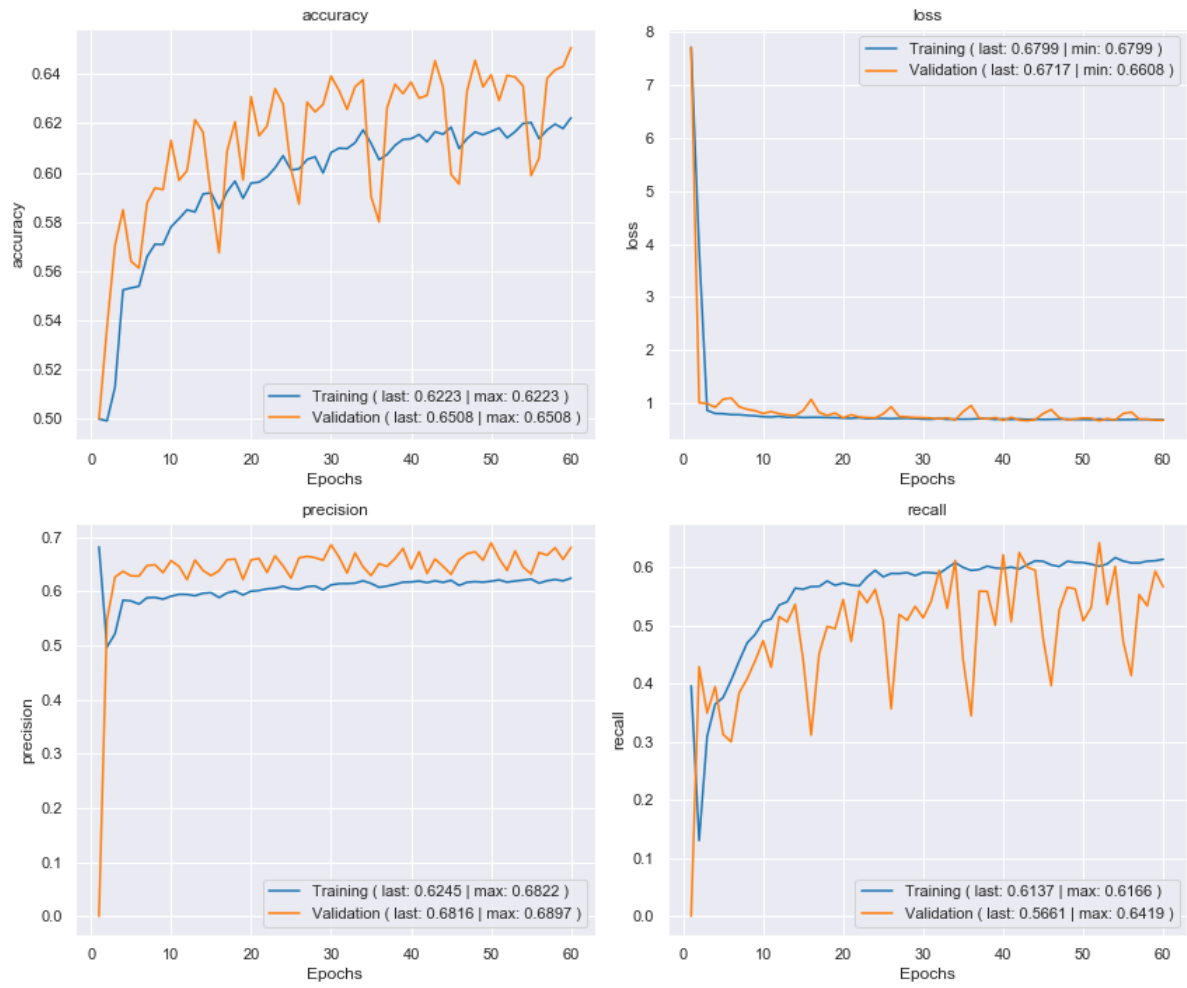


Figure 64: Accuracy, precision and recall for Word2Vec + SNN + TN + Output Manhattan.

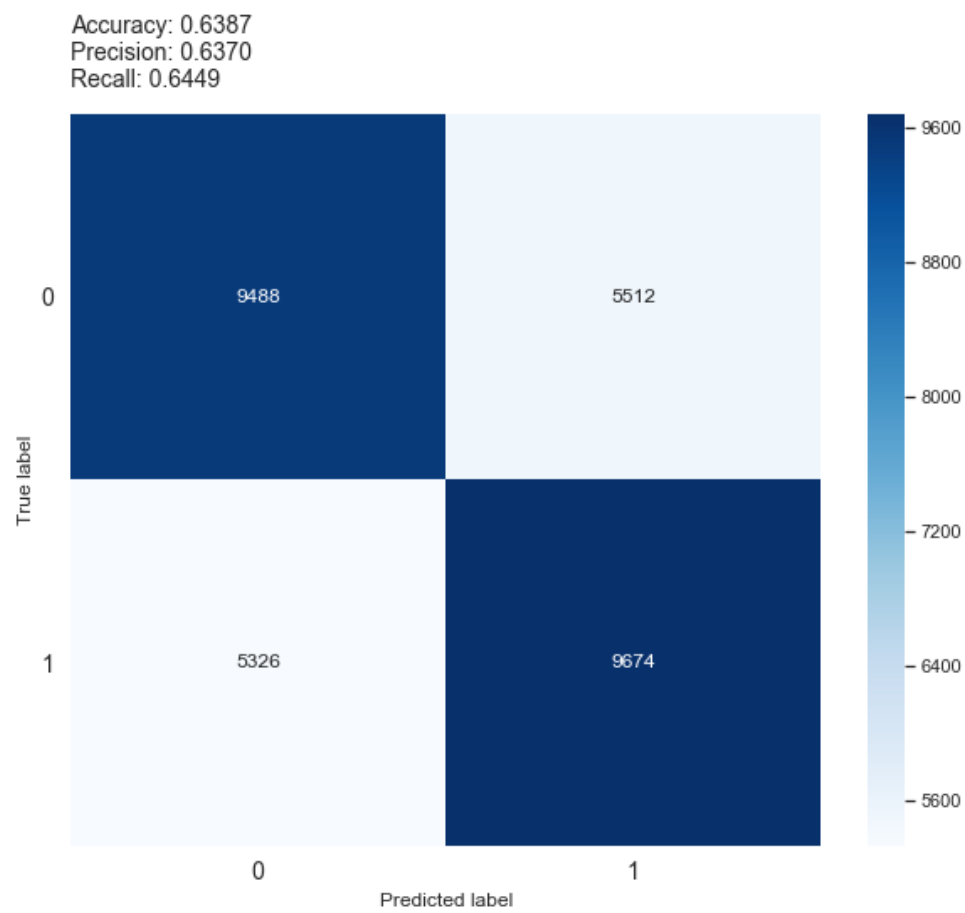


Figure 65: Validation confusion matrix for Word2Vec + SNN + TN + Output Manhattan.

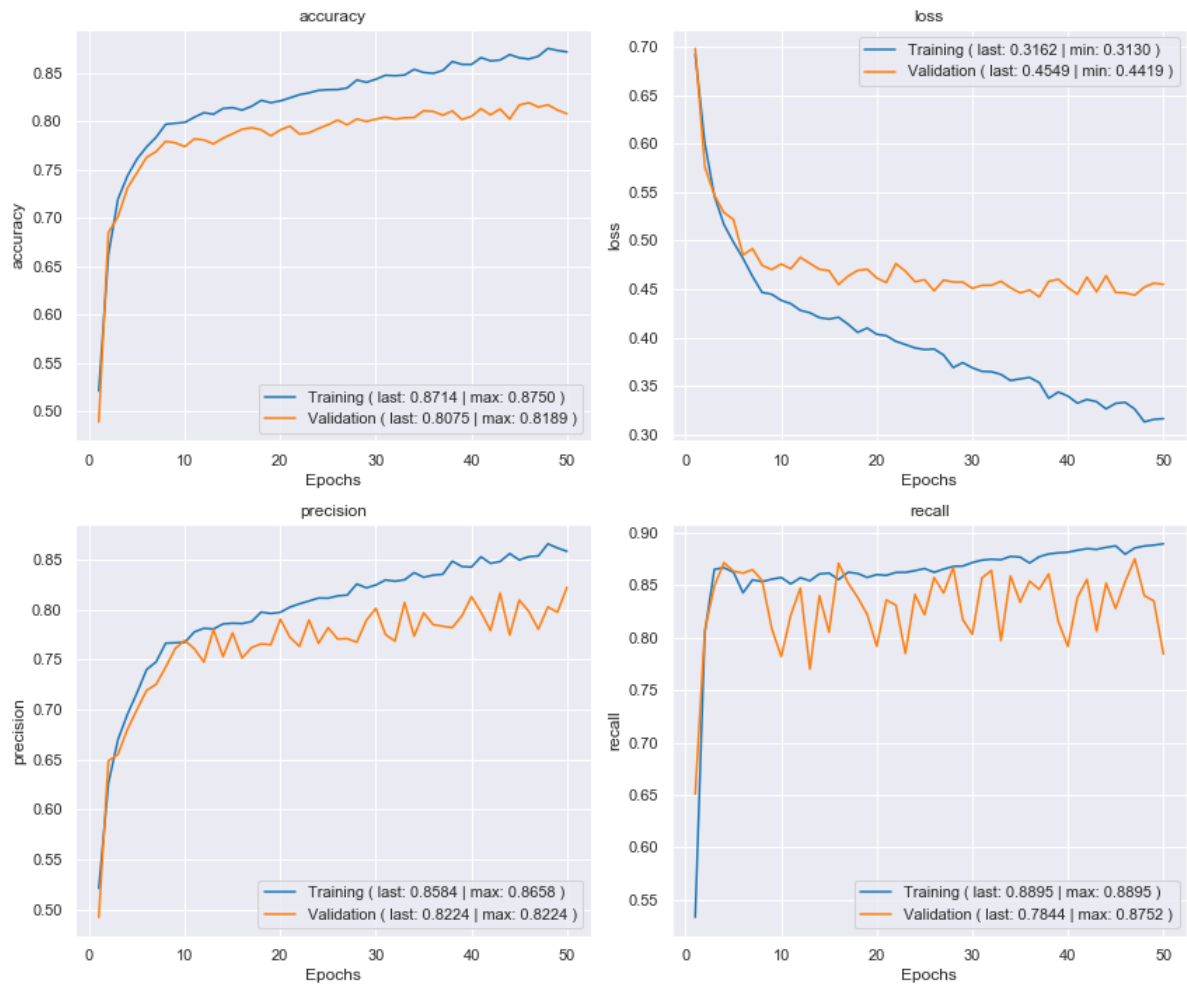


Figure 66: Accuracy, precision and recall for GloVe + SNN + LSTM + Output MLP.

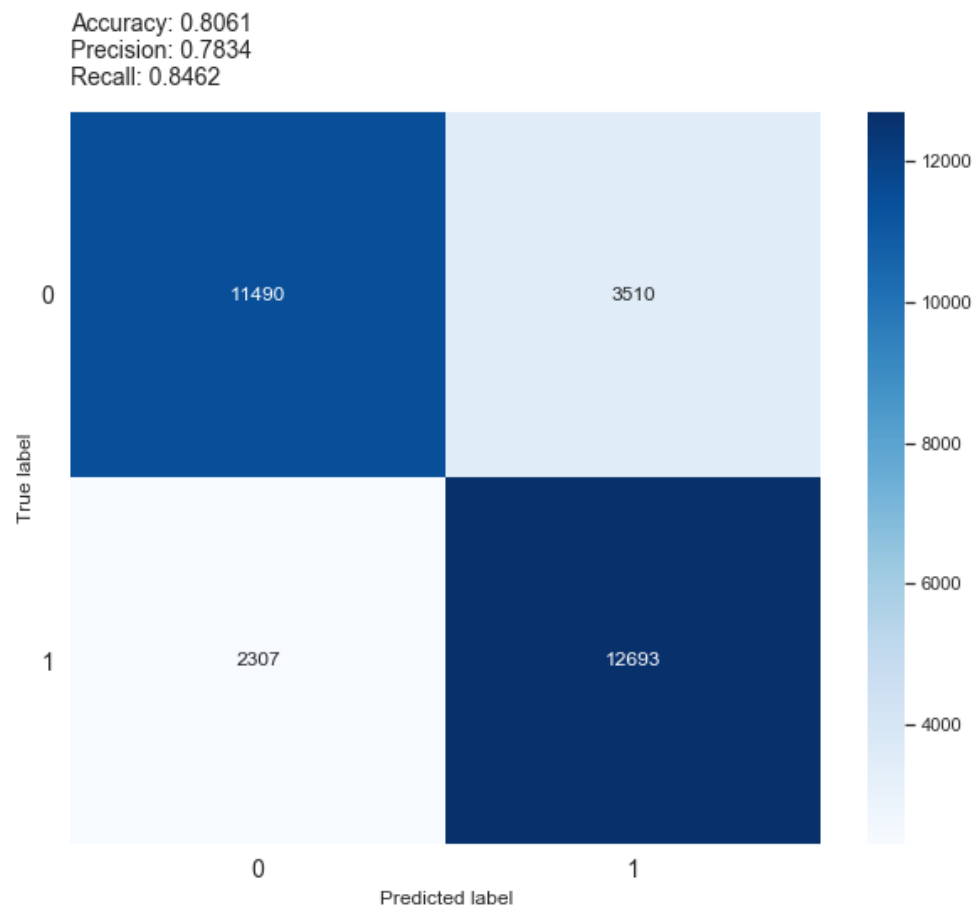


Figure 67: Validation confusion matrix for Glove + SNN + LSTM + Output MLP.

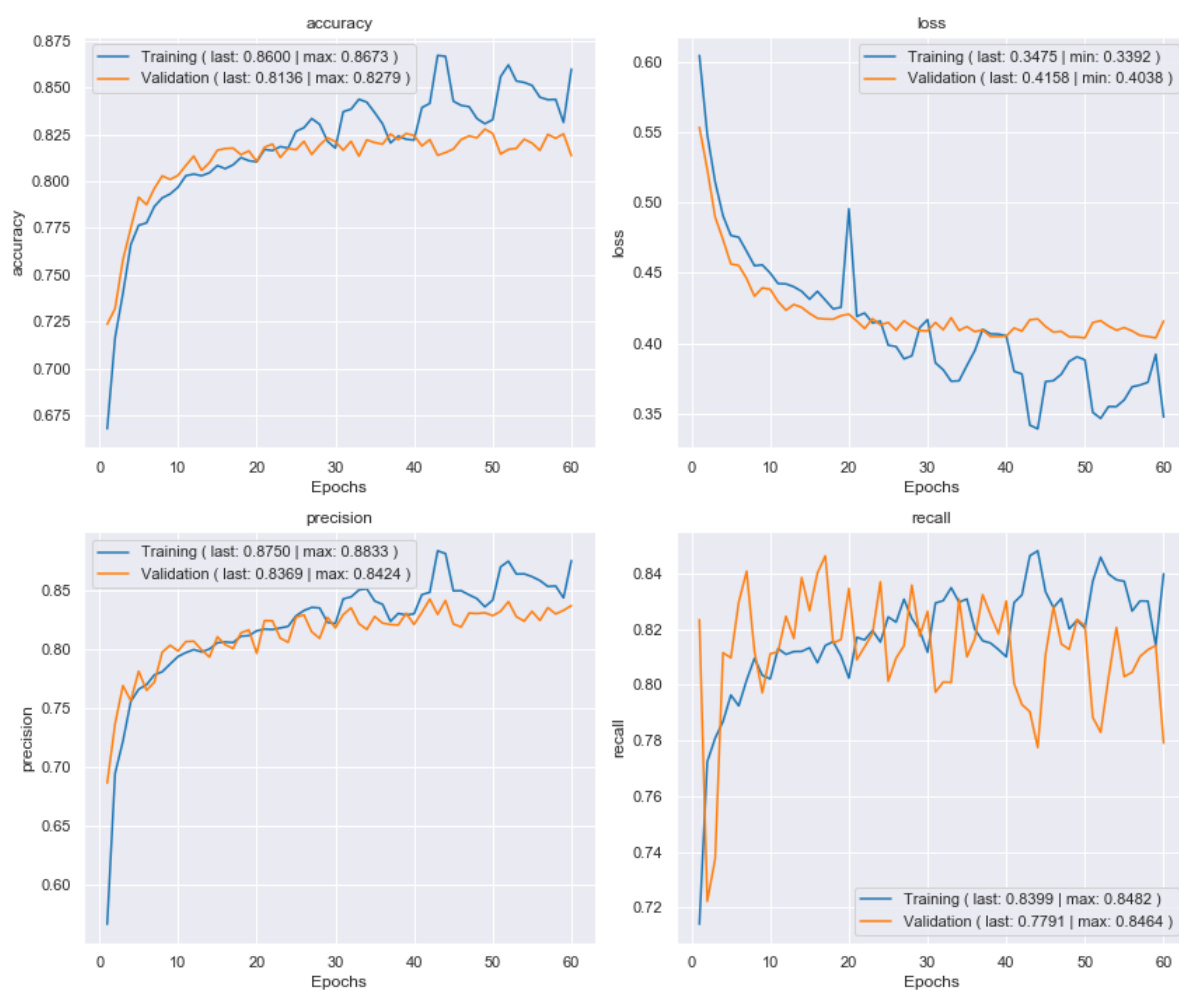


Figure 68: Accuracy, precision and recall for Glove + SNN + LSTM + Output Manhattan.

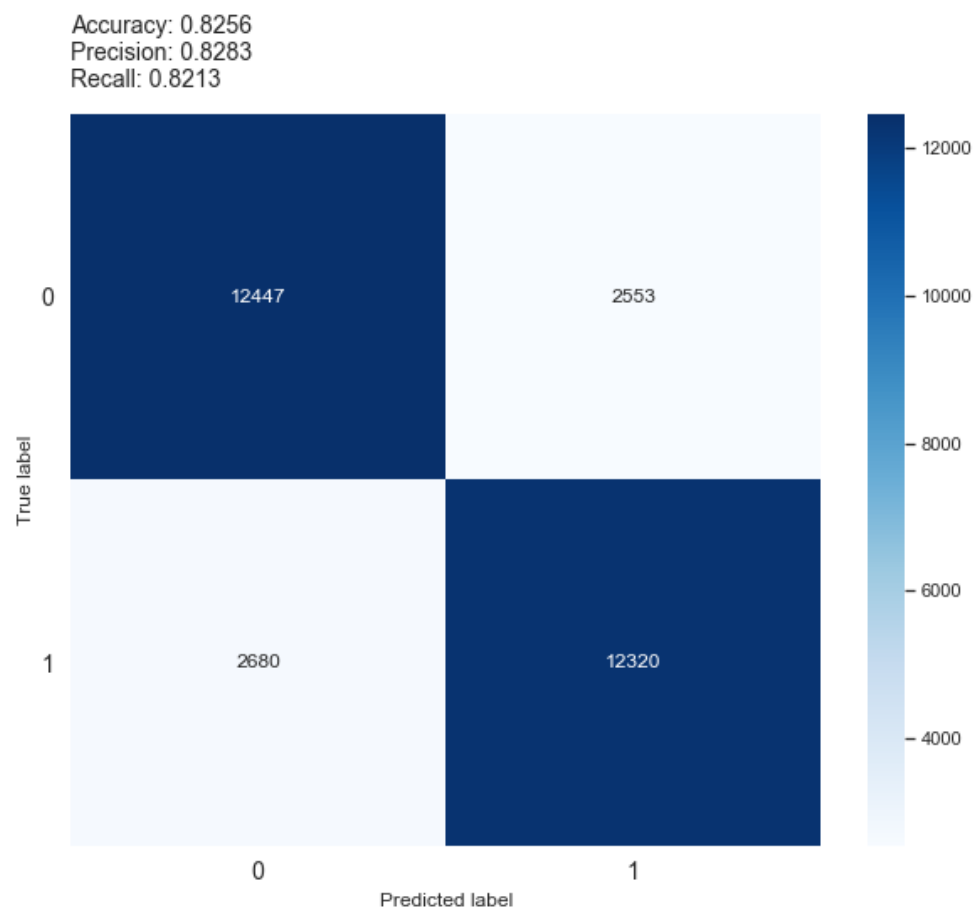


Figure 69: Validation confusion matrix for Glove + SNN + LSTM + Output Manhattan.

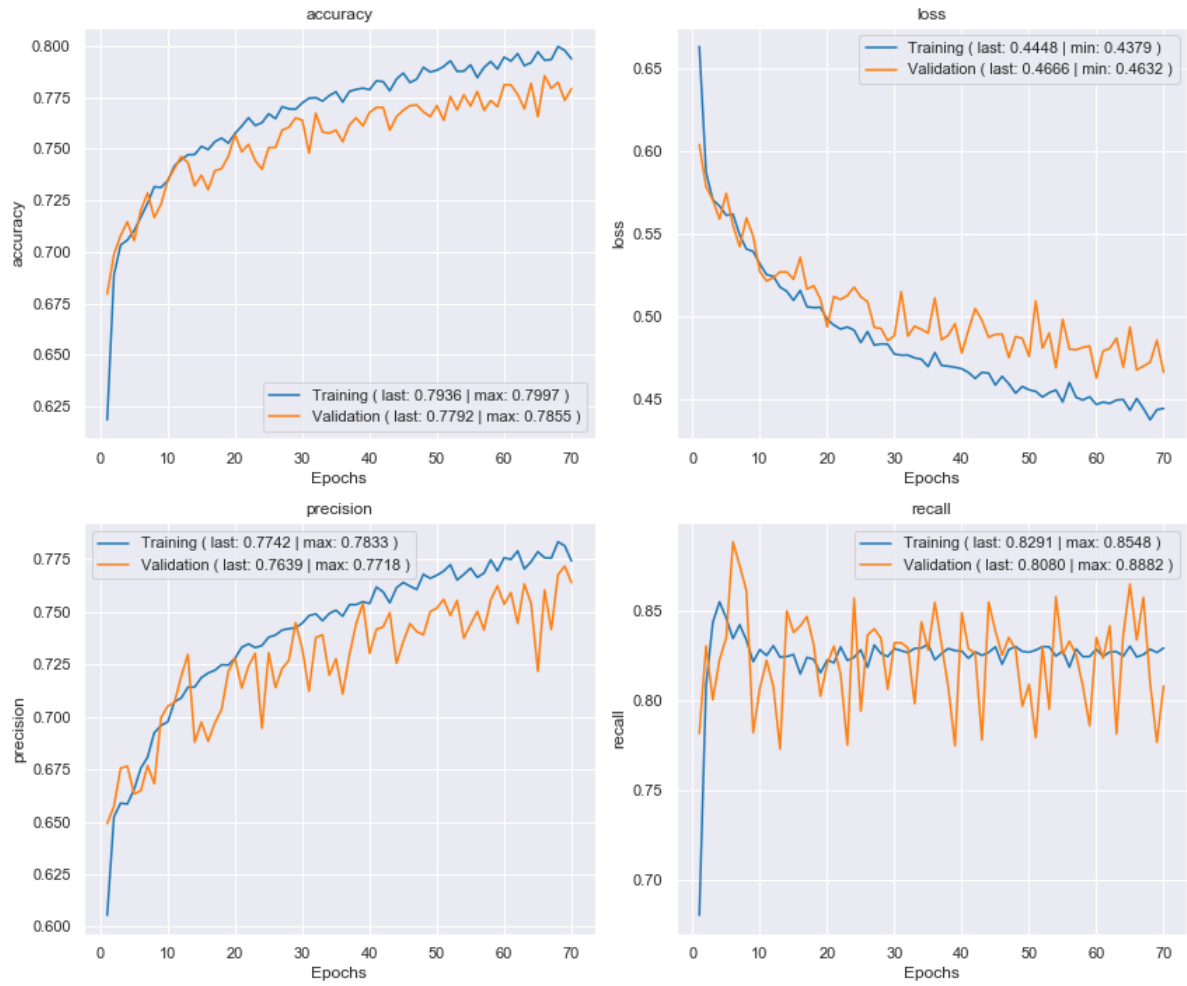


Figure 70: Accuracy, precision and recall for GloVe + SNN + TN + Output MLP.

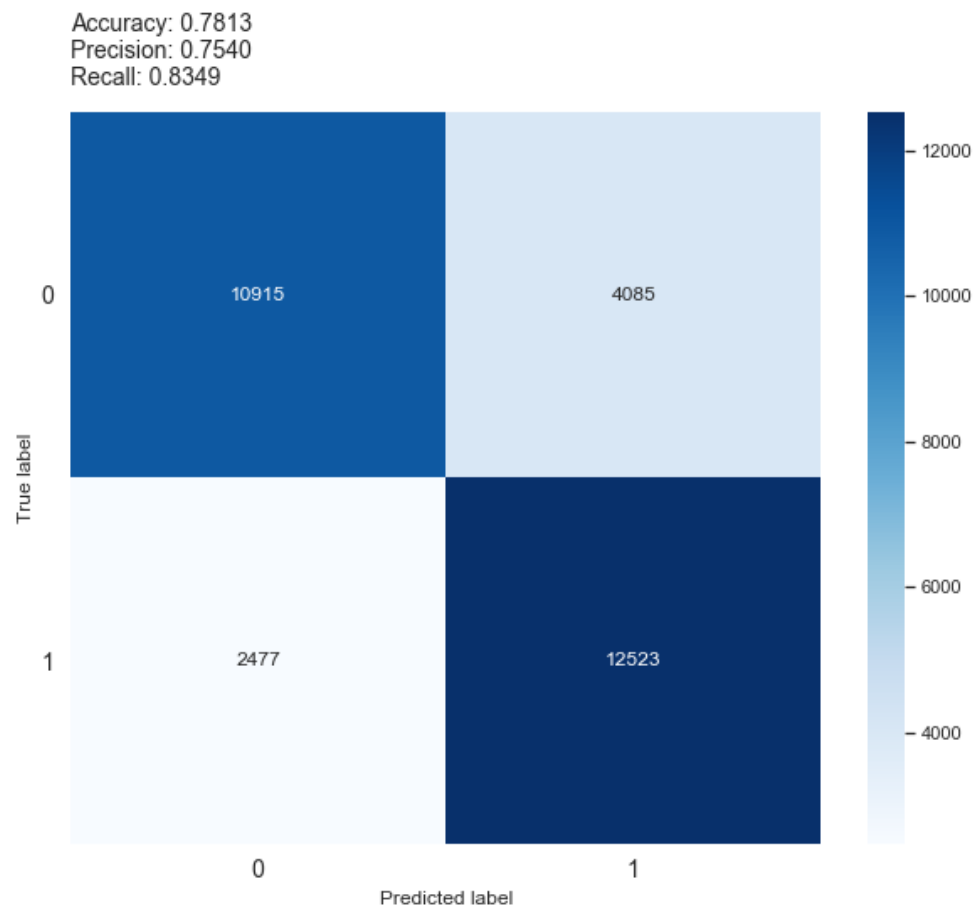


Figure 71: Validation confusion matrix for Glove + SNN + TN + Output MLP.

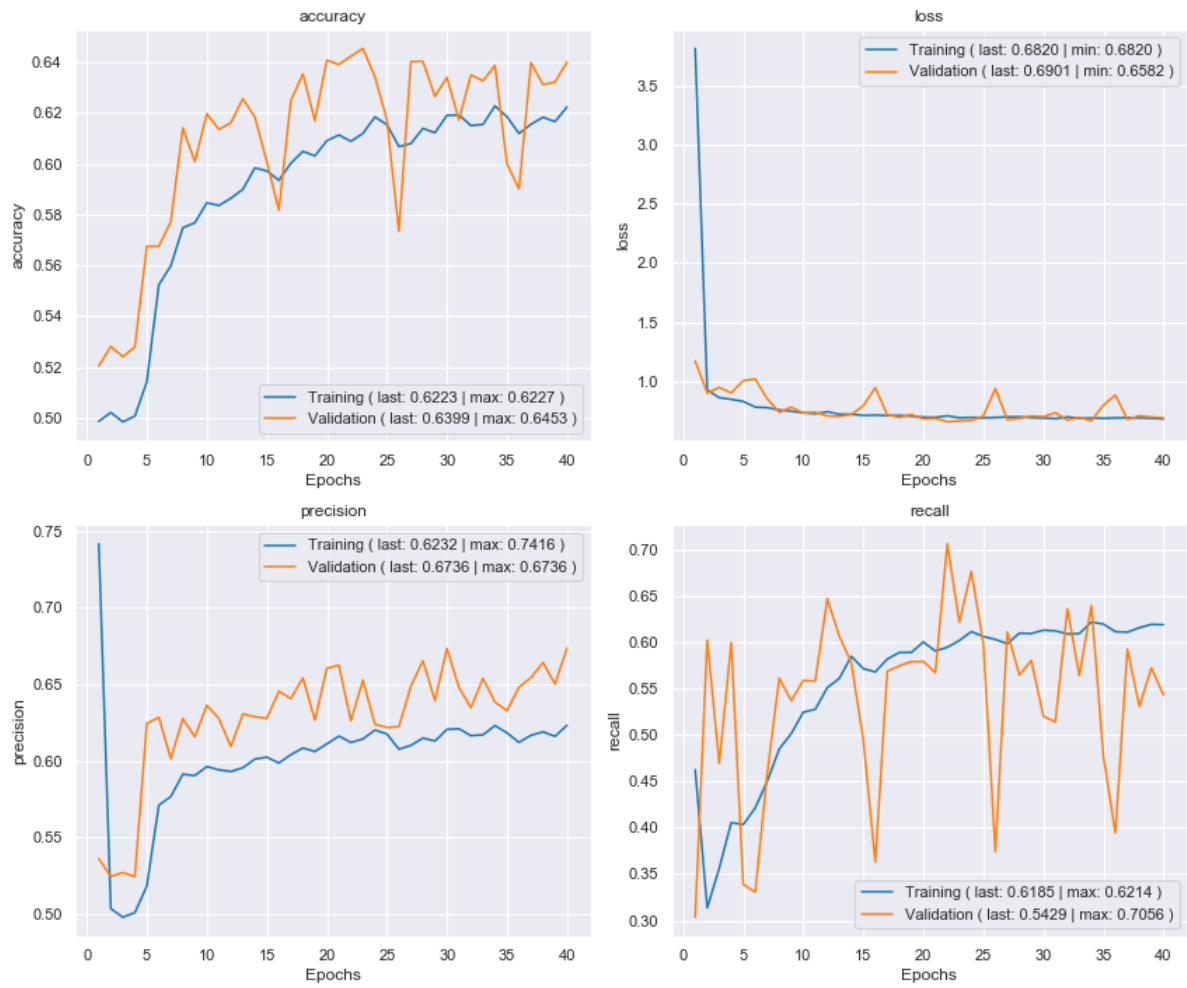


Figure 72: Accuracy, precision and recall for Glove + SNN + TN + Output Manhattan.

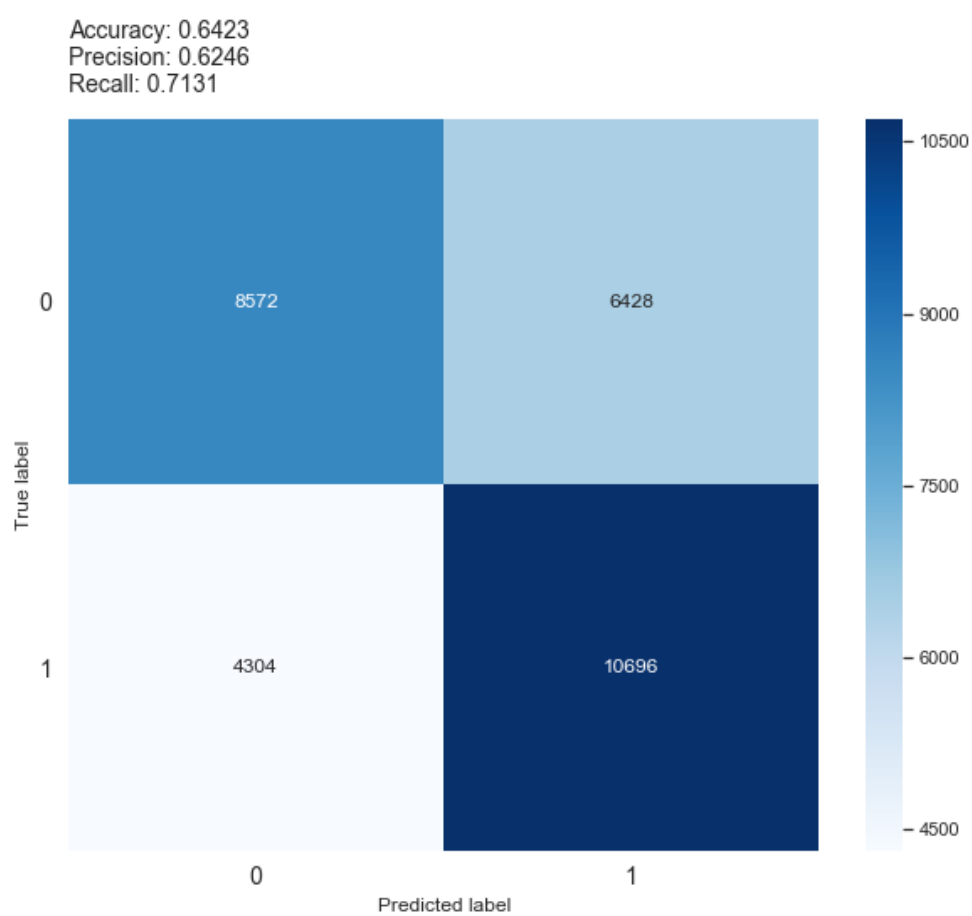


Figure 73: Validation confusion matrix for Glove + SNN + TN + Output Manhattan.

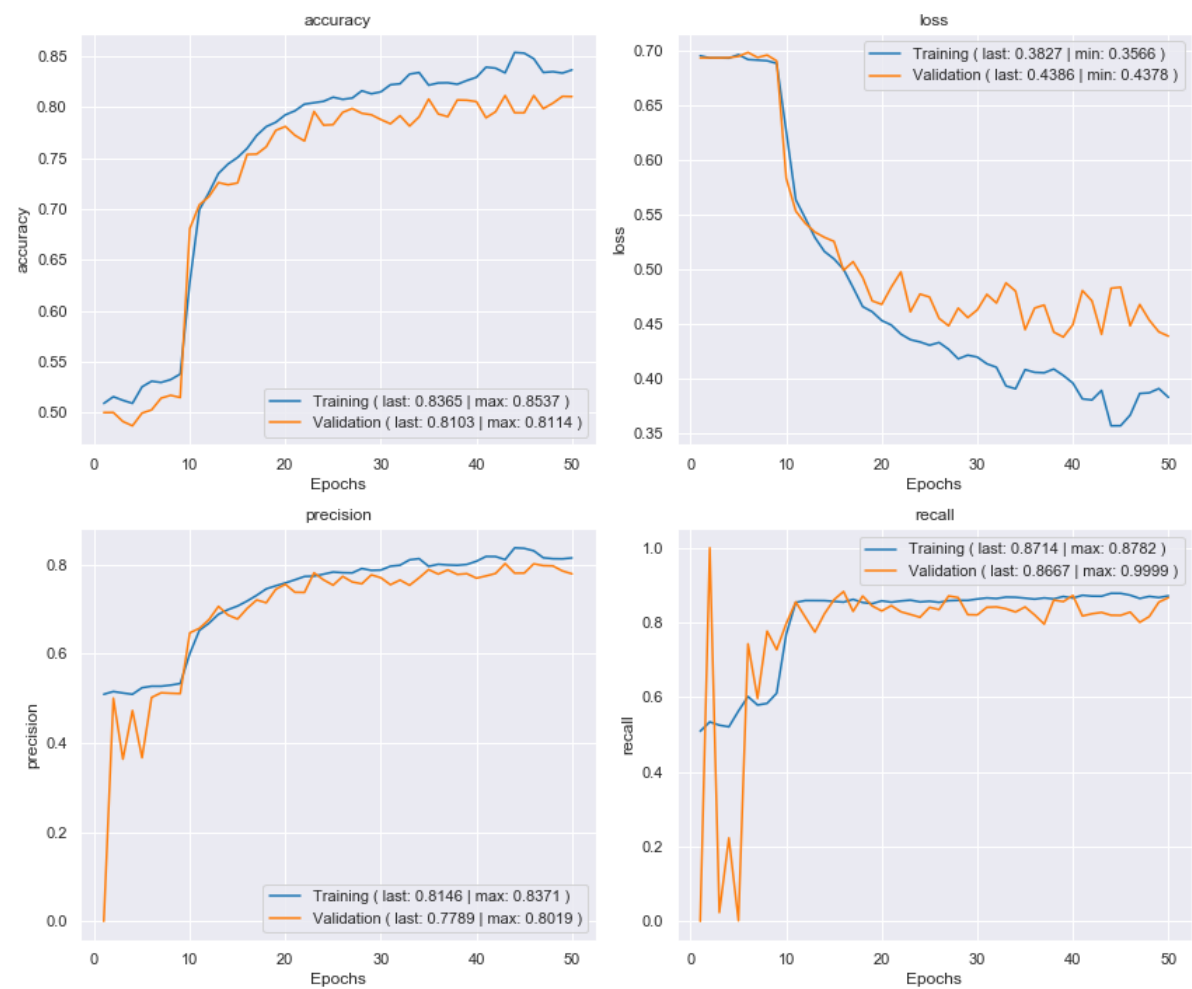


Figure 74: Accuracy, precision and recall for FastText + SNN + LSTM + Output MLP.

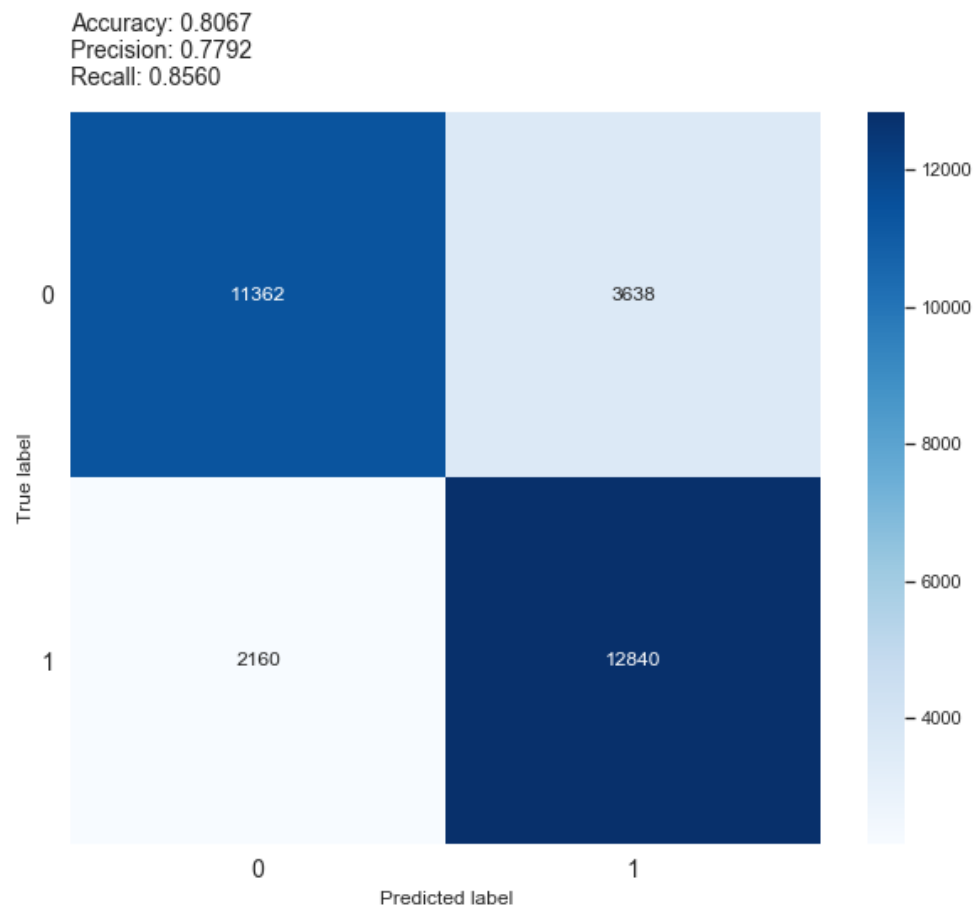


Figure 75: Validation confusion matrix for FastText + SNN + LSTM + Output MLP.

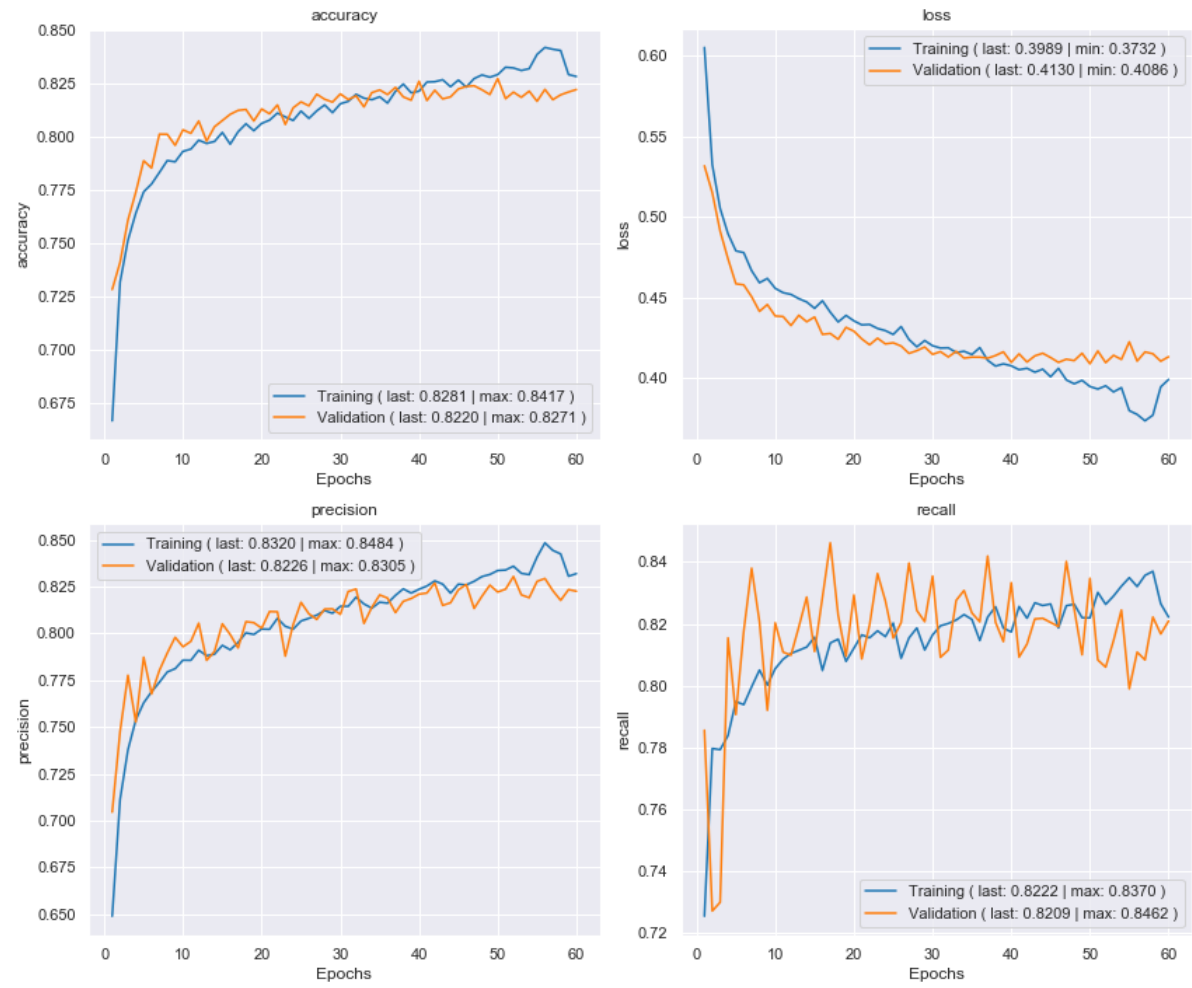


Figure 76: Accuracy, precision and recall for FastText + SNN + LSTM + Output Manhattan.

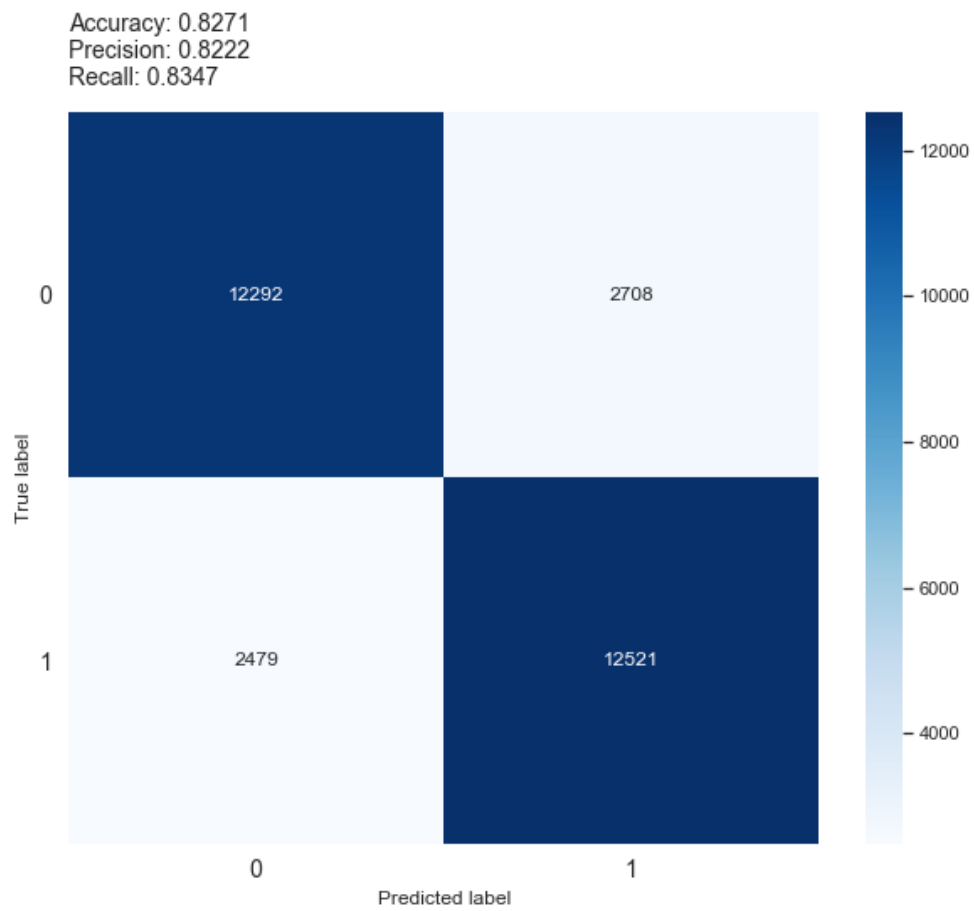


Figure 77: Validation confusion matrix for FastText + SNN + LSTM + Output Manhattan.

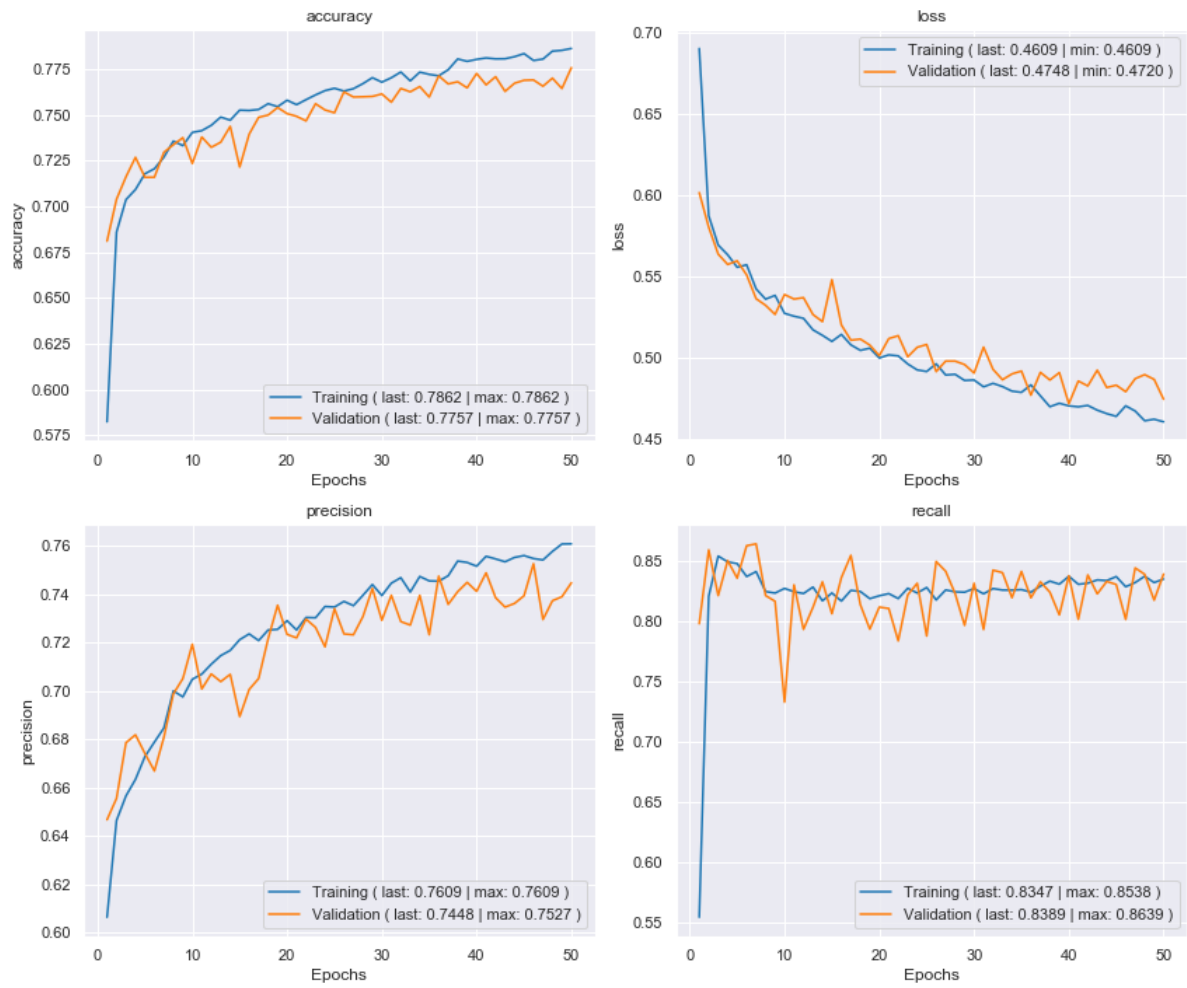


Figure 78: Accuracy, precision and recall for FastText + SNN + TN + Output MLP.

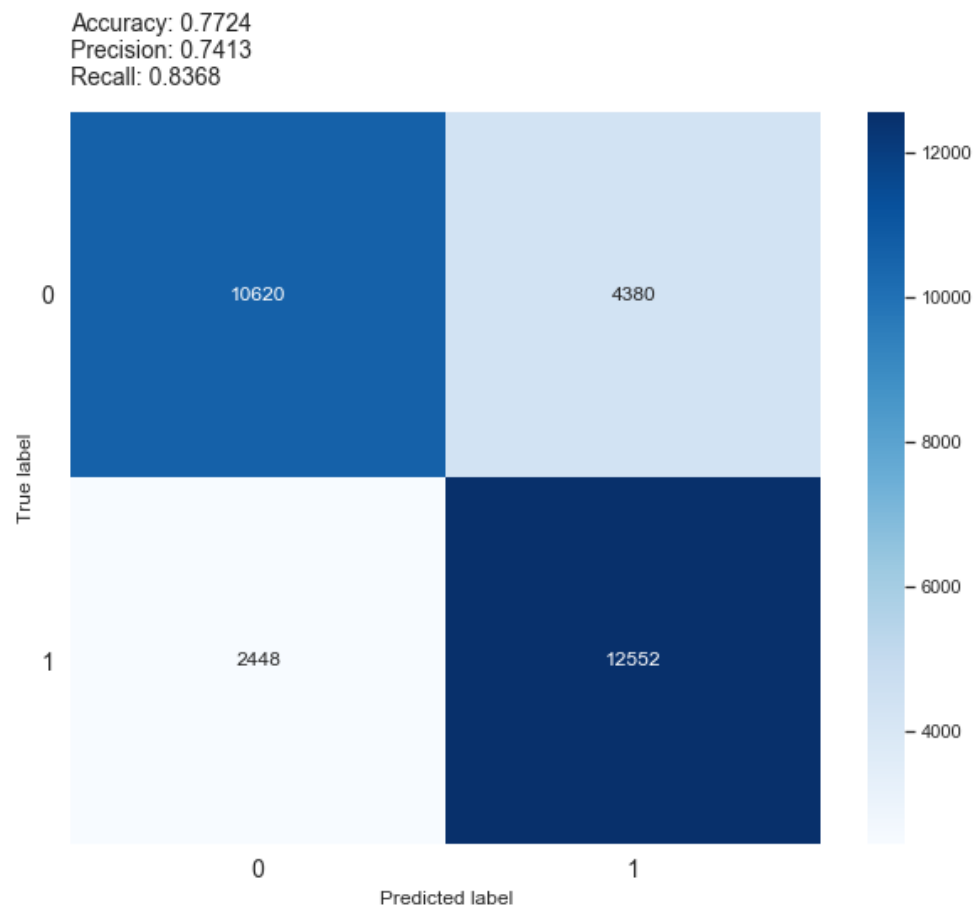


Figure 79: Validation confusion matrix for FastText + SNN + TN + Output MLP.

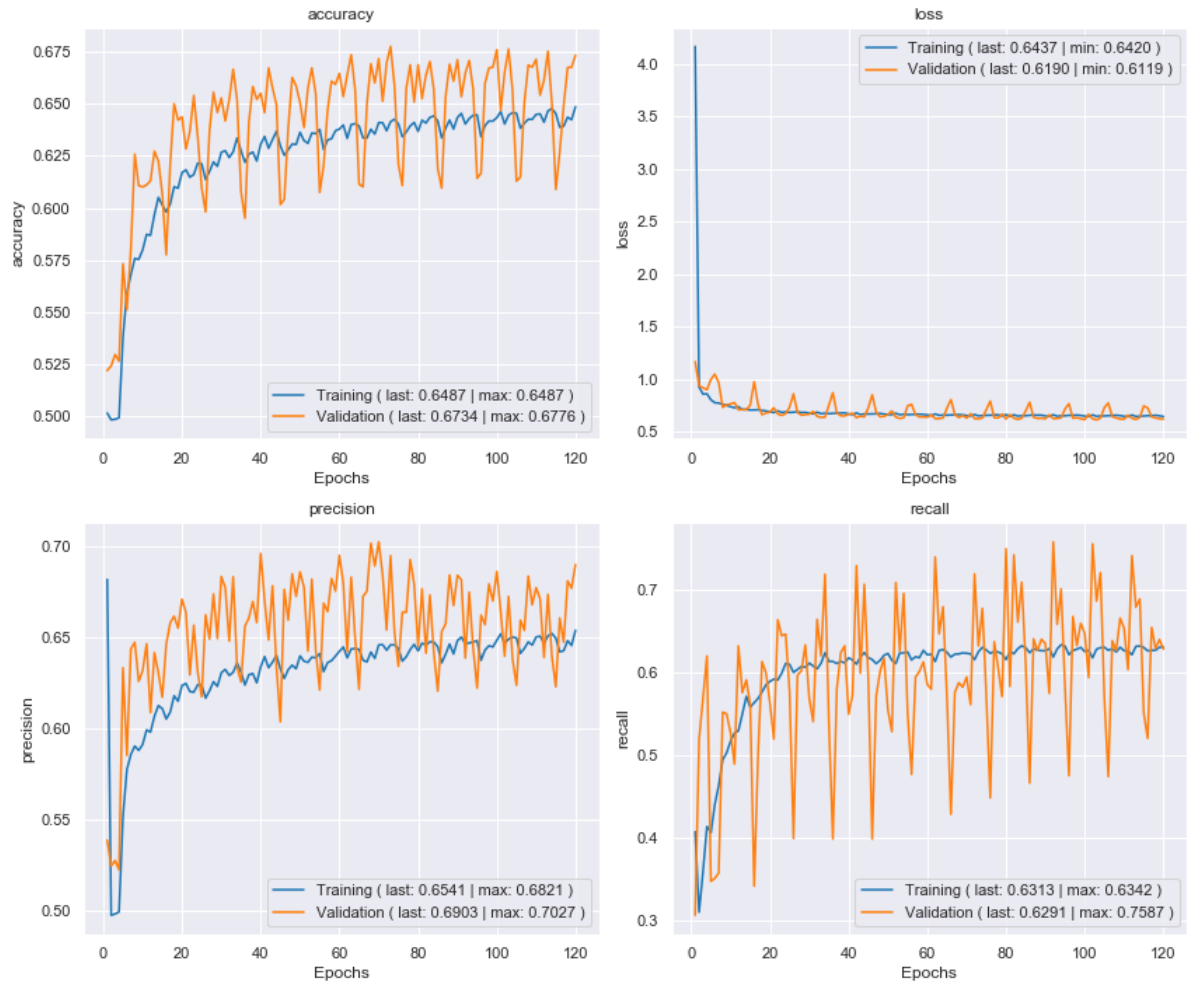


Figure 80: Accuracy, precision and recall for FastText + SNN + TN + Output Manhattan.

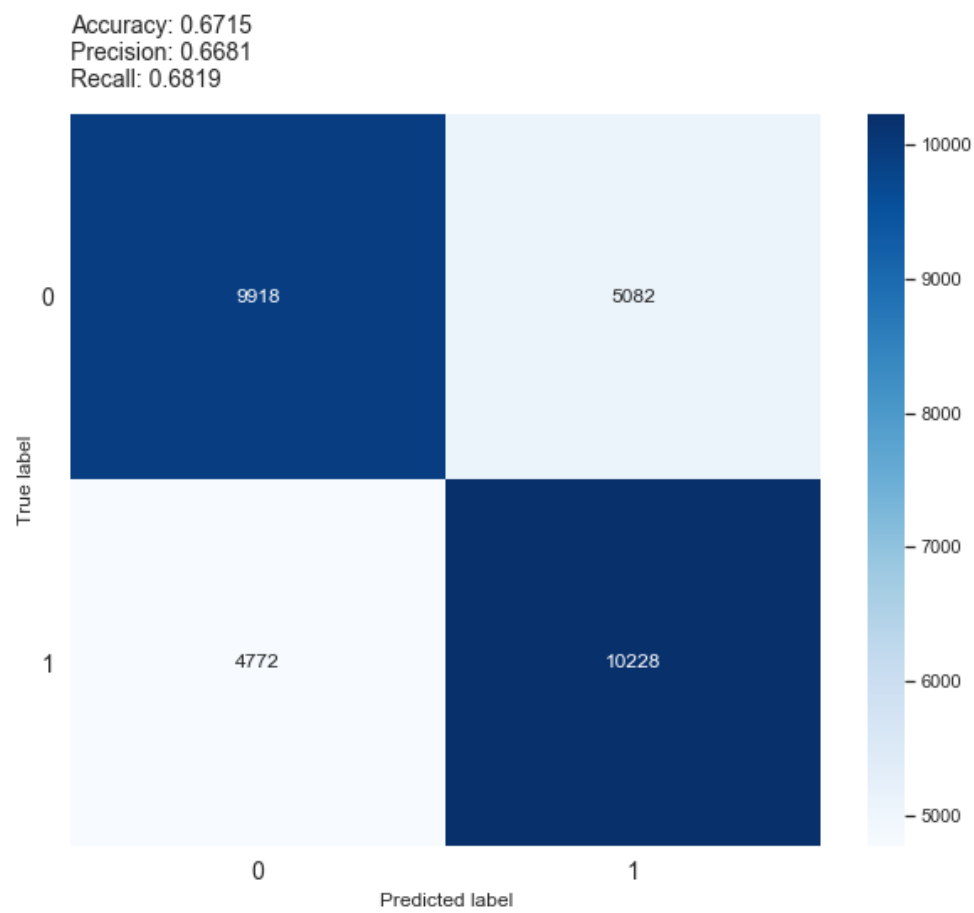


Figure 81: Validation confusion matrix for FastText + SNN + TN + Output Manhattan.