Language modeling for text classification

Clemens Westrup

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 16.1.2015

Thesis supervisor:

Michael Mathioudakis, Ph.D.

Thesis advisor:

Prof. Aristides Gionis

Aalto University
School of Science

Author: Clemens Westrup

Title: Language modeling for text classification

| Date: 16.1.2015 | Language: English | Number of pages: 6+27 |

Department of Information and Computer Science

Professorship: Machine Learning, Data Mining, and Probabilistic Modeling

Supervisor: Michael Mathioudakis, Ph.D.

Advisor: Prof. Aristides Gionis

Your abstract in English. Try to keep the abstract short; approximately 100 words should be enough. The abstract explains your research topic, the methods you have used, and the results you obtained. Your abstract in English. Try to keep the abstract short; approximately 100 words should be enough. The abstract explains your research topic, the methods you have used, and the results you obtained. Your abstract in English. Try to keep the abstract short; approximately 100 words should be enough. The abstract explains your research topic, the methods you have used, and the results you obtained. Your abstract in English. Try to keep the abstract short; approximately 100 words should be enough. The abstract explains your research topic, the methods you have used, and the results you obtained.

Keywords: NLP, bla bla, keyword

Preface

I want to thank bla bla bla

Otaniemi, 16.1.2015

Clemens Westrup

Contents

Symbols and abbreviations

Symbols

$\mathbf{B}$	magnetic flux density
c	speed of light in vacuum $\approx 3 \times 10^8$ [m/s]
ω_D	Debye frequency
ω_latt	average phonon frequency of lattice
$\uparrow$	electron spin direction up
$\downarrow$	electron spin direction down

Operators

$\nabla \times \mathbf{A}$	curl of vectorin $\mathbf{A}$
$\dfrac{\mathrm{d}}{\mathrm{d}t}$	derivative with respect to variable t
$\dfrac{\partial}{\partial t}$	partial derivative with respect to variable t
$\sum_i$	sum over index i
$\mathbf{A} \cdot \mathbf{B}$	dot product of vectors $\mathbf{A}$ and $\mathbf{B}$

Abbreviations

AC	alternating current
APLAC	an object-oriented analog circuit simulator and design tool (originally Analysis Program for Linear Active Circuits)
BCS	Bardeen-Cooper-Schrieffer
DC	direct current
TEM	transverse eletromagnetic

Todo list

0.1 TODO

- Describe the process of finding the problem

1 Introduction

1.1 Motivation

1.2 Structure

2 Background / Context

This thesis

"In the multiclass text classification task, we are given a training set of documents, each labeled as belonging to one of K disjoint classes, and a new unlabeled test document. Using the training set as a guide, we must predict the most likely class for the test document."[Do and Ng, 2006]

Meta-learning

other shit

2.1 Research objectives

main objective interim objectives

2.2 Related work

The problem of text classification has been studied from a variety of perspectives.

2.2.1 Feature extraction from Text

2.2.2 Text classification

3 Methods

3.1 TODO

- Linear machine

- explain metrics

- single class, multi-class and multi-label classification

- hard vs soft allocation (probabilistic vs discriminant)

3.2 Problem formalism

The problem at hand is to design a predictive function μ that best approximates the correct target response y_i for an input document d_i, given as

$$\{(d_1, y_1), (d_2, y_2), \ldots, (d_n, y_n)\}, \quad d_i \in \mathcal{D}, \; y_i \in \mathcal{Y} \tag{1}$$

where $\mathcal{D}$ is a set of n documents, each of which is sequence of varying length that is composed of UTF-8 encoded characters, and $\mathcal{Y}$ is a the label space with $y_i \in \{0, 1\}^c$ indicating the presence of each of the c class labels to be predicted.

4 Data

> **Describe data format**

In order to perform supervised learning labelled data was needed for training. Together with the process of reframing of the research problem this was approached in an iterative way. First a quick prototypical tool was built to collect labels in a crowd-sourced fashion. This allowed getting more knowledge about the problem itself, especially with regards to how humans perform the task of labelling topics of text sections, and to perform first experiments of algorithmically achieving meaningful results in agreement to human behavior on this task. Then these learnings were taken into consideration when re-scoping the research problem and according to that data was collected using the microtasking service crowdflower [cro, 2016], leading to a quality dataset of labelled sentences from job ads.

4.0.1 Explorative Data Collection

> **Picture of software setup?**

To collect first data a tool was build, consisting of a Node.js [nod, 2016] server using MongoDB[mon, 2016] as a database and communicating via a JSON with a simplistic website front-end using the mustache template engine [mus, 2016]. The

tool is online[1] and it's source code is publicly available on GitHub[2] with it's API documentation hosted online as well[3].

The data generated by using the free text description of each job ad and splitting it into paragraphs as can be seen in the software package as well[4].

The goal of this prototype tool for data collection was on the one hand to acquire data in order to carry our first experiments as fast as possible, and on the other hand to gain a deeper understanding about the research problem itself by giving an open, unbiased task to the participants. In particular the question at hand was how humans label the content of the different parts of a job ad.

The exact task given to the participants was "Describe what each section is about by adding one or more tags/keywords to it". They were shown a job ad that was split into paragraphs and besides each paragraph was a text field to enter 1 or more tags.

Figure 1: Interface of the tagging tool

In a first step the tool was only shown to 3 participants to get immediate feedback if the user interface had flaws and whether the task was understood. Based on this feedback the tool was improved by providing an example for the participants and then tested with a slightly larger group of 12 persons. After correcting a few minor details in the user interface a public link was then shared via social media and other channels with as many people as possible. A few days later the tool was then also shared internally within Sanoma where it was set up as a competition to tag the most possible job ads.

In total 91 job ads were tagged, resulting in 379 tagged text sections and 358 tags.

---

[1] http://thesis.cwestrup.de/jobad-tagger/
[2] https://github.com/cle-ment/thesis-tagger
[3] http://thesis.cwestrup.de/jobad-tagger/apidoc/
[4] https://github.com/cle-ment/thesis-tagger/blob/master/pre-processing.ipynb

Figure 2: t-SNE Embedding

4.0.2 Crowdsourced Data Collection with Refined Research Problem

Figure 3: Principal Components Projection

(a) Confidence

(b) Cumulative Confidence
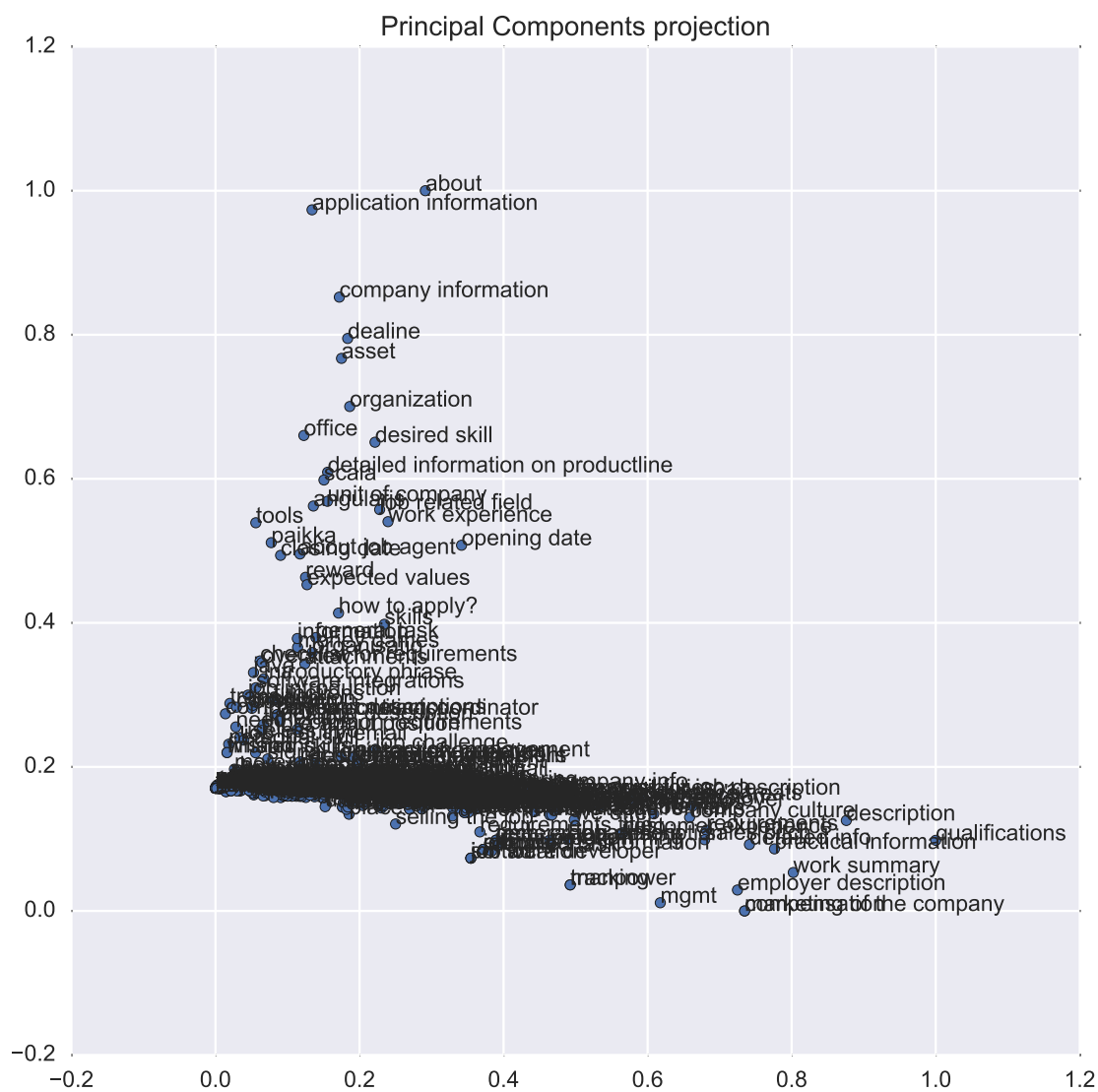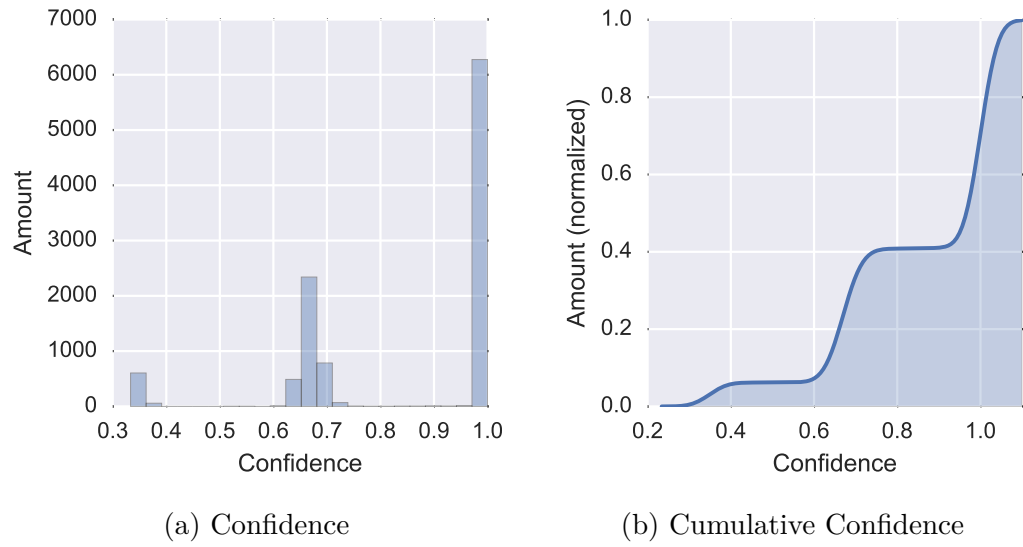
Figure 4: Amount of label judgements versus label confidence of the sentence label data collected via crowdflower
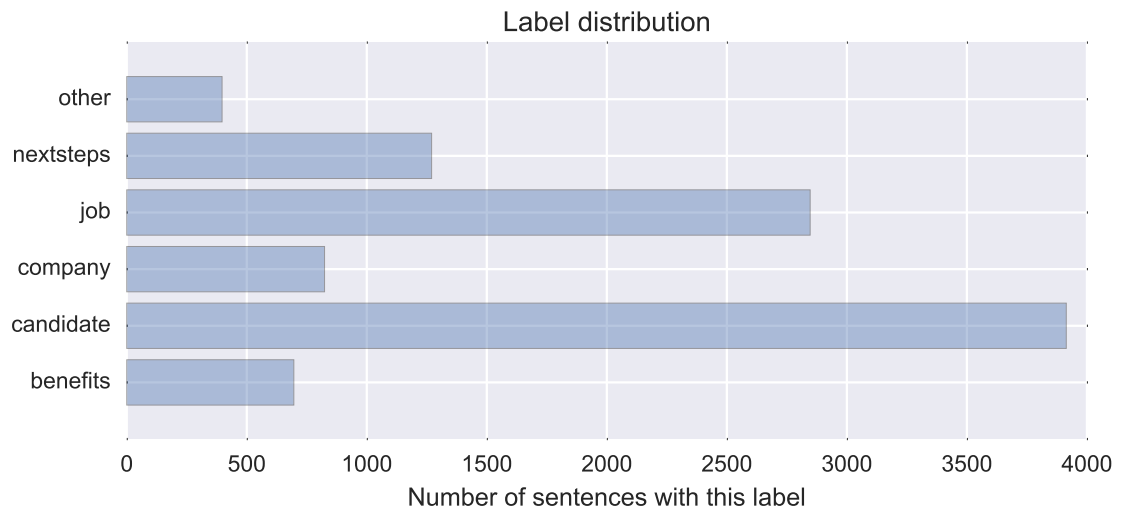
Figure 5: Distribution of labels in sentence data

5 Evaluation

5.1 Binary Classification

In binary or dichotomous case of classification we are given a single class k and a set of labelled data points $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ where targets $y_i \in \{0, 1\}$ encode whether a data point x_i belongs the class c or not. The task is then to achieve correct classification of new data points without knowing the true label via a model function or predictor $f(\cdot)$.

To evaluate such a predictor it is useful to present the results in form of a contingency table as shown in table Table 1, because it gives valuable insights about the performance of the prediction. The table shows the proportion of data points that belong to the class (RP) or not (RN) and were predicted correctly (TP) or incorrectly (FN), as well as the number of data samples that do not belong to the class (RN) and were falsely predicted to be in the class (FP) or correctly predicted to not be in the class (TN), and the same proportions for the positively (PP) and negatively (PN) predicted cases with respect to the true assignments to the data. N refers to the total amount of data points.

	Real Positives (RP)	Real Negatives (RN)
Predicted Positives (PP)	True Positives (TP)	False Positives (FP)
Predicted Negatives (PN)	False Negatives (FN)	True Negatives (TN)

Table 1: Contingency table for binary classification

An intuitive choice towards classification is to simply ask which data points were correctly classified to belong to the class or not. In terms of the contingency table above the ratio of $(TP + FP)/(N)$, commonly referred to as the "accuracy" of the classifier.

This choice can give a good intuition and it does capture the effectiveness on both true positives as well as true negatives, but it is strongly influenced by bias of the true and predicted class distribution (known as prevalence RP/N and label bias) as pointed out by [Powers, 2011]. For example given a population of 900 positive and 100 negative examples, a predictor that simply always chooses a positive assignment can achieve accuracy of 90% while it obviously is not a great predictor.

5.1.1 Precision, Recall and F1 Score

In the field of Information Retrieval it is common practice to measure the effectiveness of a predictive system in terms of its precision and recall. The precision of such system is "the proportion of retrieved material that is actually relevant" whereas the

explain suervised clas-sifi-ac-tion and bi-naty vs multi

recall measures "proportion of relevant material actually retrieved in answer to a search request" [Rijsbergen, 1979]. Formally these two measures are defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

This

As both, high precision and recall, are important for an robust information retrieval system they are typically combined into a single measure such as the F-measure, also referred to as F-score. The F-score is the weighted harmonic mean between precision and recall, derived from the measure of effectiveness proposed in [Rijsbergen, 1979]. The most common form is the F_1 score where precision and recall are assigned equal weight:

$$F_1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

The F_1 score has the advantage of its intuitive interpretability as both precision and recall are well understood measures and, analogous to recall, precision and accuracy, as it lives in the range $[0, 1]$, giving a single number that can express the effectiveness of the system in terms of percentage.

The F1 score is widely used in the field of Machine Learning and Data Mining and thus it is an important measure to consider to compare results to outcomes of prior publications by others. It is however important to point out that any version of the F-measure is a biased score as it "ignores TN which can vary freely without affecting the statistic" [Powers, 2011]. This can affect the evaluation of a classifier when the class distribution is skewed (prevalence) or the classifier develops a bias towards certain classes (label bias), motivating the use of unbiased measures in these cases, such as the ones described next.

5.1.2 Informedness, Markedness and Matthews Correlation Coefficient

[Powers, 2011] introduces unbiased analogue measures to Recall and Precision, called "Informedness" and "Markedness" respectively. As [Powers, 2011] lays out, "Informedness quantifies how informed a predictor is for the specified condition, and specifies the probability that a prediction is informed in relation to the condition (versus chance).":

$$\begin{aligned}
\text{Informedness} &= \text{Recall} + \text{Inverse Recall} - 1 \\
&= 1 - \text{Miss Rate} - \text{Fallout} \\
&= 1 - \frac{\text{FN}}{\text{RN}} - \frac{\text{FP}}{\text{RP}}
\end{aligned} \tag{5}$$

Further he defines: "Markedness quantifies how marked a condition is for the specified predictor, and specifies the probability that a condition is marked by the predictor (versus chance)."

$$\begin{aligned} \text{Informedness} &= \text{Recall} + \text{Inverse Recall} - 1 \\ &= 1 - \text{Miss Rate} - \text{Fallout} \\ &= 1 - \frac{\text{FN}}{\text{RN}} - \frac{\text{FP}}{\text{RP}} \end{aligned} \tag{6}$$

Based on Informedness and Markedness we can then see that *Matthews Correlation Coefficient* r_G, first proposed by [Matthews, 1975], is a score that balances these two measures:

$$\begin{aligned} r_G &= \pm\sqrt{\text{Informedness} \cdot \text{Markedness}} \\ &= \frac{(\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN})}{(\text{TP} + \text{FN})(\text{FP} + \text{TN})(\text{TP} + \text{FP})(\text{FN} + \text{TN})} \end{aligned} \tag{7}$$

Matthews Correlation Coefficient can thus be used as unbiased alternative to the F-measure and offers a similar ease of interpretability as it ranges from -1 to 1, the former indicating a negative correlation or adverse estimation and the latter indicating a perfect prediction, while a coefficient of 0 reflects chance.

5.2 Multi-class classification

Multi-class classification refers to a generalization of the binary case where we aim to predict for each datapoint x_i one of K labels for the classes at hand. The target space $\mathcal{Y}$ can be represented with each $y_i \in \{0,1\}^k$, known as *one-hot encoding*, where each target is c-dimensional vector. Alternatively we can encode the targets as categorical variables $y_i \in c_1, c_2, \ldots, c_k$. The contingency table from the binary case can be extended as in table 2, which is then commonly known as *Confusion Matrix* or *Error Matrix* .

citation for Confusion Matrix?

	Real Class 1	Real Class 2	...	Real Class k
Predicted Class 1	...	...		...
Predicted Class 2	...	...		...
...				
Predicted Class k	...	...		...

Table 2: Contingency table for k classes, also referred to as Confusion Matrix

5.3   Multi-label classification

5.4   Discriminant Functions for Multi-class Classification

A simple approach to multi-class classification is to pose the learning problem as a combination of binary classification problems as described in [Bishop, 2006, Chapter 4.1.2, p. 182]. This can be done by using K separate classifiers, each of which predicts one of the classes against all $K - 1$ other classes, which is known as the *one-versus-the-rest* classification scheme. An alternative approach is to train $K(K-1)/2$ binary classifiers for each possible pair of classes, referred to as *one-versus-one* classification.

These extensions though have major drawbacks as pointed out by [Duda et al., 1973, Chapter 5.2.2]. As illustrated by 6 both of the classification schemes lead to ambiguous regions in the hypothesis space as their classification is undefined.
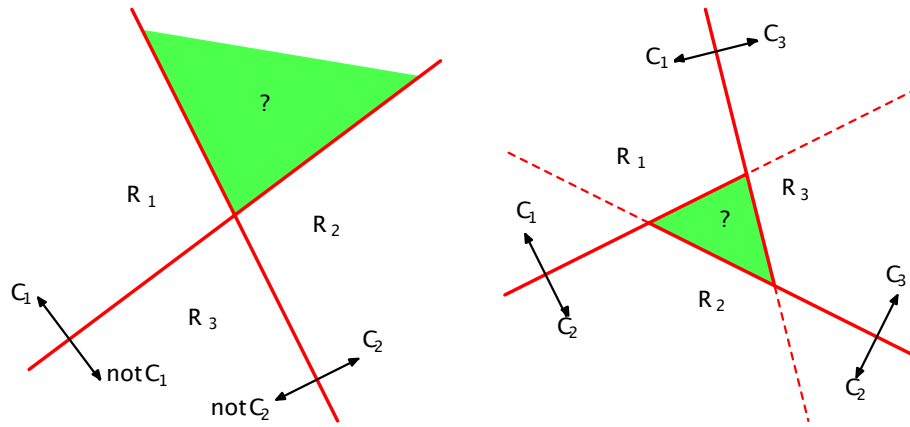
(a) One-Vs-Rest classification scheme   (b) One-Vs-One classification scheme

Figure 6: : Ambiguous regions in the hypothesis space ([Bishop, 2006] Chapter 4, Figure 4.1)

[Bishop, 2006]

6   Vector-Space Models

Thus each document document vector has the same dimensionality its dimensions can be used as features to be fed into most popular classification metho

6.1   N-gram language models

N-gram language models are based on co-occurrences of word or character sequences, so-called N-grams or k-shingles as they are referred to in the Data Mining literature [Leskovec et al., 2014, Chapter 3.2, p. 72]. Formally an N-gram is defined as a sequence of n items, each of which consist of n characters or words, effectively used to capture sub-sequences of text. Common choices are N-grams of size 1, 2 or 3 — called "unigrams", "bigrams" and "trigrams" respectively — and the definition can be extended to using a window size $[w_{min}, w_{max}]$, employing all combinations of N-grams in this interval.

N-grams are usually used to create a vector-space model by representing each document in a dataset as a *bag-of-words* or *bag-of-N-grams* vector so that each dimension of the vector represents statistics about the corresponding N-gram. For unigrams

parameters - words vs chars - stop words - ngram range - max features - TF.IDF vs simple wordcounts - sublinear TF - norm

- show T-SNE embeddings of doc2vec vectors

Today N-gram models are still in wide use and considered as state of the art "not because there are no better techniques, but because those better techniques are computationally much more complex, and provide just marginal improvements" [Mikolov, 2012, p. 17].

Notable shortcomings of this method are it's inability to capture word-order

- vector space models

- bag of words

- "The most important weakness is that the number of possible n-grams increases exponentially with the length of the context, preventing these models to effectively capture longer context patterns. This is especially painful if large amounts of training data are available, as much of the patterns from the training data cannot be effectively represented by n-grams and cannot be thus discovered during training. The idea of using neural network based LMs is based on this observation, and tries to overcome the exponential increase of parameters by sharing parameters among similar events, no longer requiring exact match of the history H." [Mikolov, 2012, p. 17]

http://localhost:8888/notebooks/thesis/sandbox/crowdflower-data-collection/extract-data-crowdflower.ipynb

7 Experiments

7.1 TODO

- Comparison one-vs-rest and one-vs-one against linear machine

- Visualizations and embeddings of data in 2D (and decision boundaries?)

- show T-SNE embeddings of doc2vec vectors

7.2 Effectiveness and Expressiveness of Statistical (Vector Space) Language Models

As section 6 explains, a popular way to approach text classification and other tasks in natural language processing is to build a language model by creating explicit representations of the objects or entities to be processed in a vector space. This way such vectors can be used as features and depending on the representation they can also encode notions of similarity of associativity between these objects.

In order to determine the most effective text representation, in a first set of experiments different approaches where studied and compared. Each method was studied with regards to the effect of its hyper-parameters on effectiveness when producing an input space to different classifiers, but also time and memory requirements at training and inference time are taken into account .

say why using the sentence dataset here

reference jupyter notebook here

actually discuss time and memory requirements

7.2.1 Baselines Classifiers: Uniform and Stratified Guessing

As a baseline for comparing the performance of classification two different guessing strategies were used, namely uniform and stratified guessing. Uniform guessing refers to a predictor that samples from the given classes assuming a uniform distribution whereas stratified guessing takes the label distribution in the data as the underlying probability distribution and samples from it. Both, uniform and stratified guessing achieve a Matthews Correlation Coefficient score of around 0 (averaged over 1000 runs) as expected for such guessing strategies (see section 5.1.2). On the other hand the accuracy for uniform guessing is around 0.16 which corresponds to 1/C for the C classes and around 0.26 for stratified guessing which reflects the skew of the label distribution. Figure 7 shows the confusion matrices for these baseline variants in absolute and normalized form which, clearly revealing the guessing strategies.

7.2.2 N-gram Language Models

The first class of language models that was investigated for the task of multi-class classification are N-gram models that were explained in section **??**

(a) Uniform

(b) Uniform (normalized)

(c) Stratified

(d) Stratified (normalized)

Figure 7: Confusion matrices of uniform and stratified guessing strategies. It

8 Results

8.1 TODO

- compare the two datasets in quality

Something

9   Discussion and Conclusions

Something

Further research

- how well do word2vec and comparable methods generalize: e.g. initialize a text corpus with word vectors from a bigger corpus (Google News), then train an RNN to predict the next word vector using the small corpus but use the bigger corpus to validate and see if words in bigger corpus can be inferred

-

References

[mus, 2016] (2016). mustache . https://mustache.github.io.

[cro, 2016] (2016). Crowdflower. https://www.crowdflower.com.

[mon, 2016] (2016). Mongodb. https://www.mongodb.com.

[nod, 2016] (2016). Node.js. https://nodejs.org/en/.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Do and Ng, 2006] Do, C. B. and Ng, A. Y. (2006). Transfer learning for text classification. In Weiss, Y., Schölkopf, B., and Platt, J. C., editors, *Advances in Neural Information Processing Systems 18*, pages 299–306. MIT Press.

[Duda et al., 1973] Duda, R. O., Hart, P. E., and others (1973). *Pattern classification and scene analysis*, volume 3. Wiley New York.

[Leskovec et al., 2014] Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of massive datasets*. Cambridge University Press.

[Matthews, 1975] Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451.

[Mikolov, 2012] Mikolov, T. (2012). *Statistical Language Models Based on Neural Networks*. PhD thesis, Ph. D. thesis, Brno University of Technology.

[Powers, 2011] Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.

[Rijsbergen, 1979] Rijsbergen, C. J. V. (1979). *Information Retrieval*. Butterworth-Heinemann, London ; Boston, 2nd edition edition.

A   Appendix

B   Appendix : Experiments

doc2vec

April 27, 2016

1 Classification using Distributed representations of sentences and documents

Testing paragraph vectors approach as proposed in [1].
Date: 22.04.2016

1.1 Context

Different language models lead to differente expressiveness in the feature space and thus alternatives were explored. This particular approach promised state-of-the-art results.

1.2 Experiment Rationale

The main goal of these experiments was to compare the performance of the approach in [1] with a simple bag-of-words model, especially given the rather small dataset.

1.3 Testing Procedure and Metrics

Using [4,5] a bag-of-words model was trained on the the labelled sentence dataset with a TF.IDF transformation applied to it and then classification was carried out by

1.4 Test Results

From manual judgement the word2vec mapping works exceptionally well to find relations between words, however simply adding up vectors to represent several words was an overly naive approach and does not work (Note: In retrospective this approach does not even make sense mathematically.)

1.5 Learnings

- Word2vec mapping has potential if the thesis scope will focus on NLP (natural language processing) since it is aware of the local context of words as opposed to a simple bag-of-words approach.
- Representing documents needs a more sophisticated approach.

1.6 References

1. Le QV, Mikolov T (2014) Distributed representations of sentences and documents. arXiv preprint arXiv:1405.4053
2. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed Representations of Words and Phrases and their Compositionality. In: Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ, ed., Advances in Neural Information Processing Systems 26. Curran Associates, Inc.. 3111–3119
3. Mikolov T (2012) Statistical Language Models Based on Neural Networks. Ph.D. dissertation. Ph. D. thesis, Brno University of Technology.

4. http://scikit-learn.org/stable/
5. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay É (October 2011) Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12:2825−2830
6. https://radimrehurek.com/gensim/
7. Řehůřek R, Sojka P (May 2010) Software Framework for Topic Modelling with Large Corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. Valletta, Malta: ELRA. 45–50

2 Code

```python
In [8]: def warn(*args, **kwargs):
            pass
        import warnings
        warnings.warn = warn
```

```python
In [9]: import pickle
        import datetime
        import logging
        import time
        import pandas as pd
        import numpy as np
        from random import shuffle
        import multiprocessing


        import sys
        sys.path.append('../joblearn-experiments/triton-experiments/workdir/')

        import joblearn.dataset
        import joblearn.gridsearch
        import joblearn.feat_extr
        import joblearn.feat_trans
        import joblearn.scoring
        import joblearn.target_trans
        import joblearn.estimation

        import sklearn.cross_validation
        import sklearn.feature_extraction
        import sklearn.linear_model
        import sklearn.neighbors

        import gensim
        from gensim import models
```

```python
In [10]: ### Basic setup

         FEAT_DIM = 200

         # test set size
         TEST_SIZE = 0.3

         # timestamp at runtime
```

```
          TIMESTAMP = str(int(time.time()))
          EXP_NAME = "paragraph2vec"
```

In [11]: *### Dataset Initialization*

```
          df = pd.read_csv(
              "../joblearn-experiments/local-experiments/workdir/data/sentences_aggregated_50-249.csv")

          # Use entries with label confidence over 0.6 and aren't test questions:
          df_conf = df[df['0_label:confidence'] > 0.6]
          df_conf = df_conf[df_conf['_golden'] == False]
          df_conf = df_conf[['0_label', '0_label:confidence', '0-sentence',
                  '0-context-after', '0-context-before']]

          label_array = np.array(df_conf['0_label'])
          le = sklearn.preprocessing.LabelEncoder()
          le.fit(label_array)
          data_Y = le.transform(label_array)
          data_Y_labels = le.classes_


          data_X = np.array(df_conf['0-sentence'])

          ### Train/Test Splits Setup

          label_groupings = {}
          data_splits = {}

          # no grouping
          label_groupings["none"] = joblearn.target_trans.LabelGrouping("No grouping",
                                                            data_Y,
                                                            data_Y_labels)
          (X_train, X_test, Y_train, Y_test) = sklearn.cross_validation.train_test_split(
                  data_X, data_Y, test_size=TEST_SIZE, random_state=0)
          data_splits["none"] = joblearn.target_trans.DataSplit(X_train, X_test,
                                                            Y_train, Y_test)
```

2.1 Doc2Vec

In [12]:
```
alldocs = []
          for line_no, document in enumerate(data_X):
              words = gensim.utils.to_unicode(document).split()
              alldocs.append(gensim.models.doc2vec.LabeledSentence(words, [line_no]))

          Xdocs = alldocs[:]
```

In [13]:
```
# model = Doc2Vec(documents, size=100, window=8, min_count=5, workers=4)
          # model = gensim.models.Doc2Vec.load_word2vec_format(
          #     '../05.1-tag-classification-tfidf-clustered/data/word2vec/text8.bin',
          #     binary=True)  # C binary format

          cores = multiprocessing.cpu_count()
          assert gensim.models.doc2vec.FAST_VERSION > -1, "this will be painfully slow otherwise"
          model = gensim.models.Doc2Vec(dm=1, dm_concat=1, size=100, window=10, negative=5, hs=0, min_co
```

In [14]: `model.build_vocab(Xdocs)`

```
In [15]: for i in range(0,20):
             shuffle(Xdocs)
             model.train(Xdocs)
```

WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s

```
In [16]: X_transformed_word2vec = np.matrix(model.docvecs)
```

2.2 N-grams BOW

```
In [17]: vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(
             analyzer='word', max_features=100)
         X_transformed_bow = vectorizer.fit_transform(data_X).todense()
```

2.3 compare

```
In [18]: from sklearn import cross_validation
         from sklearn import linear_model
         from sklearn import neighbors
```

```
In [19]: (X_transformed_bow_train, X_transformed_bow_test,
          Y_bow_train, Y_bow_test) = cross_validation.train_test_split(
             X_transformed_bow, data_Y, test_size=0.3, random_state=0)

         (X_transformed_word2vec_train, X_transformed_word2vec_test, Y_word2vec_train,
          Y_word2vec_test) = cross_validation.train_test_split(
             X_transformed_word2vec, data_Y, test_size=0.3, random_state=0)
```

```
In [20]: # classifier_bow = linear_model.LogisticRegression().fit(X_transformed_bow_train, Y_bow_train)
         # classifier_word2vec = linear_model.LogisticRegression().fit(X_transformed_bow_train, Y_bow_t
         classifier_bow = neighbors.KNeighborsClassifier().fit(X_transformed_bow_train, Y_bow_train)
         classifier_word2vec = neighbors.KNeighborsClassifier().fit(X_transformed_bow_train, Y_bow_trai
```

```
In [21]: scores_word2vec = cross_validation.cross_val_score(
             classifier_word2vec, X_transformed_word2vec_test, Y_word2vec_test, cv=5,
             scoring='f1_weighted')

         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_word2vec.mean(), scores_word2vec.std() * 2))
```

```
Accuracy: 0.34 (+/- 0.03)
```

```python
In [22]: scores_bow = cross_validation.cross_val_score(
             classifier_bow, X_transformed_bow_test, Y_bow_test, cv=5,
             scoring='f1_weighted')

         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_bow.mean(), scores_bow.std() * 2))
```

```
Accuracy: 0.63 (+/- 0.07)
```

3 Improved Doc2Vec

```python
In [23]: model = gensim.models.Doc2Vec(dm=1, dm_concat=1, size=300, window=5, negative=5, hs=0, min_cou
         model.build_vocab(Xdocs)
         X_transformed_word2vec = np.matrix(model.docvecs)
```

```python
In [24]: for i in range(0,20):
             shuffle(Xdocs)
             model.train(Xdocs)
```

```
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
WARNING:gensim.models.word2vec:under 10 jobs per worker: consider setting a smaller 'batch_words' for s
```

```python
In [30]: model.intersect_word2vec_format(
             '../05.1-tag-classification-tfidf-clustered/data/word2vec/GoogleNews-vectors-negative300.b
             binary=True)
```

```python
In [31]: (X_transformed_bow_train, X_transformed_bow_test,
          Y_bow_train, Y_bow_test) = cross_validation.train_test_split(
             X_transformed_bow, data_Y, test_size=0.3, random_state=0)

         (X_transformed_word2vec_train, X_transformed_word2vec_test, Y_word2vec_train,
          Y_word2vec_test) = cross_validation.train_test_split(
             X_transformed_word2vec, data_Y, test_size=0.3, random_state=0)
```

```python
In [32]: classifier_bow = linear_model.LogisticRegression().fit(X_transformed_bow_train, Y_bow_train)
         classifier_word2vec = linear_model.LogisticRegression().fit(X_transformed_bow_train, Y_bow_tra
         # classifier_bow = neighbors.KNeighborsClassifier().fit(X_transformed_bow_train, Y_bow_train)
         # classifier_word2vec = neighbors.KNeighborsClassifier().fit(X_transformed_bow_train, Y_bow_tr
```

```
In [33]: scores_word2vec = cross_validation.cross_val_score(
            classifier_word2vec, X_transformed_word2vec_test, Y_word2vec_test, cv=5,
            scoring='f1_weighted')

         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_word2vec.mean(), scores_word2vec.std() * 2))

Accuracy: 0.23 (+/- 0.00)

In [34]: scores_bow = cross_validation.cross_val_score(
            classifier_bow, X_transformed_bow_test, Y_bow_test, cv=5,
            scoring='f1_weighted')

         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_bow.mean(), scores_bow.std() * 2))

Accuracy: 0.68 (+/- 0.06)

In [ ]:
```