Affective Sentiment and Emotional Analysis of Pull Request Comments on GitHub

by

Deepak Rishi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2017

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Sentiment and emotional analysis on online collaborative software development forums can be very useful to gain important insights into the behaviors and personalities of the developers. Such information can later on be used to increase productivity of developers by making recommendations on how to behave best in order to get a task accomplished. However, due to the highly technical nature of the data present in online collaborative software development forums, mining sentiments and emotions becomes a very challenging task. In this work we present a new approach for mining sentiments and emotions from software development datasets using Interaction Process Analysis(IPA) labels and machine learning. We also apply distance metric learning as a preprocessing step before training a feed forward neural network and report the precision, recall, F1 and accuracy.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Jesse Hoey of the David R. Cheriton School of Computer Science for giving me the incredible opportunity to work with him. He gave me the freedom to take any number of courses and decide the direction of my thesis. I could not have asked for a better mentor and supervisor for my masters.

I would like to thank Prof. Mei Nagappan and Prof. Jimmy Lin of the David R. Cheriton School of Computer Science for reading my thesis. Special thanks to Prof. Mei Nagappan for his incredible guidance on this thesis.

I would also like to thank my dear friends Josh Jung, Nisarg Bhavsar, Chintak Sheth, Sukriti Arora, Chris Zhu, Julia Zhou, Alex Sacs, Chengyi Zhou, Hemant Surale, Cheryl Shang, Dan Wang, Sunjay Varma, Areej Alhothali and Aaron Li for their constant motivation and support.

Finally I would like to thank my parents, my sisters and my grandma for their endless love and support. I am forever grateful for having you in my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Sentiment and emotion analysis has become an integral part of social media. All interactions on the internet generate some kind of sentiments or emotions. Emotions and sentiments can influence our actions in a number of domains. For example, changes in activity of buying and selling in the stock market can be correlated with the moods of people evaluated with tweets [6] and consumer opinions on retailer sites can influence buyer decisions [47]. Companies all over the world use sentiment analysis for various purposes, such as:

- Understanding public opinion about products,
- Developing market strategy, and
- Improving customer service

Most sentiment and emotion analysis tasks are performed on datasets such as Twitter feed, news articles and customer opinions about products (movies reviews, product reviews on Amazon etc.). One relatively unexplored area in sentiment and emotional analysis are discussions related to software engineering. Such discussions are very common on technical blog posts, developer conversations on slack/IRC channels, mailing lists of open source programming projects and pull request discussions on GitHub. Such discussions can provide valuable insights into the behaviors and personalities of developers. Fredickson [15] states that positive emotions like happiness helps people to be more creative, which are essential for successful software design. On the other hand, Ambler [1] states that negative emotions, fear or absence of courage might refrain developers from changing/refactoring

their code. De Choudhury and Counts [12] also show that emotions affect task quality, productivity, creativity, group rapport and job satisfaction.

While data regarding discussions related to software engineering is openly available, it is hard to perform sentiment and emotional analysis on such datasets due to the highly technical nature of the discussions. The fact that these discussions might contain some segments of code, only makes it harder to perform sentiment or emotion analysis. Most of the work done on such datasets is not based on machine learning or natural language processing. Murgia et al. [44] perform a feasibility study of emotions mining using *Parrott's framework* on the issue reports of *Apache software foundation.* Guzman et al. [24] use lexical sentiment analysis to study emotions expressed in commit comments of different open source projects. Pletea et al. [49] use Natural Language Text Processing (NLTK) tool to explore sentiment analysis of security related discussions on GitHub. While these studies are good, they do not necessarily achieve the main purpose of performing sentiment analysis. The goal of sentiment/emotion analysis on software engineering discussions is to build a generic model/framework which is able to provide insights into behaviors of developers. None of Murgia et al. [44], Guzman et al. [24] and Pletea et al. [49] accomplish this goal. Either the techniques used are not robust or the dataset on which the analysis is performed is not big enough or it is simply not possible to infer the behaviors of the developers from that dataset.

As previously mentioned, performing such analysis on software engineering datasets is difficult. Getting such discussions labelled with emotions is very challenging and expensive. This thesis aims to solve the problem of performing sentiment/emotional analysis on software engineering datasets with limited labelled data. This thesis projects uses pull request comments from GitHub. GitHub is one of the largest collaborative code hosting site built on top of the git version control system. Pull request comments are comments written by developers who are trying to get a patch of code merged with an existing codebase on GitHub. Such comments are an excellent example of technical conversations between developers.

The main contributions of this thesis are as follows:

1. We propose a new approach using **Interaction Process Analysis (IPA)** to perform sentiment/emotional analysis on pull request comments on GitHub.

2. We created a new dataset of pull request comments from GitHub and got it annotated into twenty two different categories (12 IPA categories and 10 emotions explained in Chapter 3).

3. We used state of the art deep learning methods to use classify the pull request comments into some of the categories (explained in Chapter 3) mentioned in step 2.

4. We applied state of the art distance metric learning algorithms: *Information Theoretic Metric Learning* and *Large margin Nearest Neighbor* as a preprocessing step, before training feedforward neural nets to predict precision, F1, recall and accuracy while classifying the pull request comments into the different categories mentioned in step 3.

The rest of the thesis is organized as follows:

1. Chapter 2 discusses some of the related work in this domain.

2. Chapter 3 discusses the dataset used in this thesis.

3. Chapter 4 discusses some deep learning algorithms and the classification results from them.

4. Chapter 5 discusses two distance metric learning algorithms: *Information Theoretic Metric Learning* and *Large Margin Nearest Neighbor* and the results after applying them as preprocessing step.

5. Chapter 6 talks about about the conclusions, details some examples of how our approach works better than an lexical analysis tool called SentiStrength and talks about further work.

Chapter 2

Related Work

Over the past decade machine learning and natural language processing have played a major role in advances in sentiment analysis. This chapter reviews the related work in sentiment analysis in software engineering datasets, machine learning, natural language processing, and Interaction Process Analysis (IPA).

2.1 GitHub as a Collaborative Software Development Platform

These days, software development is a collaborative activity in which developers interact to create and maintain a software system. **GitHub** is a collaborative code hosting site built on top of the git version control system. In addition to code hosting, collaborative code review, and integrated issue tracking, GitHub has integrated social features which developers use to communicate, collaborate, and be aware of changes and others activities. Users are able to subscribe to information by watching projects and following users, resulting in a feed of information on those projects and users of interest [32]. Currently, GitHub is one of the world's largest code hosting site, with over 10.6 million repositories.

One of the major features of GitHub that enables collaborative development, is the **fork and pull model**. In the **fork and pull model**, developers can create their own copy of a repository and submit a pull request when they want the project maintainer to pull their changes into the main branch. A pull request can be in one of the three states:

1. Open: An open pull request means that the pull request can still accept changes in the code.

2. Merged: A merged pull request means that the patch of code sent with the pull request has been integrated with the repository. After merging the pull request is usually closed.

3. Closed: A closed pull request means that no further changes can take place for the code on the pull request. A pull request does not need to be merged in order to be closed.

Many of the projects hosted on GitHub are public, thus anyone can view the activity within those projects. The activities includes actions around issues, pull requests, and commits including comments and subscription information. The large amount of public data on GitHub makes it possible for researchers to easily mine the project data [32].

To mine the data, GitHub has a public REST *(Representation State Transfer)* [14] API that gives access to all of GitHub's public repositories, user info, and pull request comments, etc., which researchers can use to access information on GitHub. Although the REST API is well designed and documented, it suffers from a few shortcomings such as:

1. The REST API has a rate limit of five thousand requests per hour, which makes the complete download of the data impossible.

2. The overall schema of GitHub's data is not documented.

3. The API does not provide facilities to obtain collections of the data's key entities. The existing API calls mainly allow navigation from one entity to another.

4. Events are only provided as streams, which disappear into a sliding window of 300 entries.

To overcome the aforementioned limitations of mining the data from GitHub, Gousios and Spinellis [22] came up with **GHTorrent**: a service that gathers event streams and data from the GitHub's hosting site and provides that data back to the community in the form of incremental MySQL/MongoDB data dumps distributed through the peer-to-peer BitTorrent [9] protocol.

The main contributions of Gousios and Spinellis [22] are as follows:

1. Creating a documentation for GitHub's schema by using GitHub's REST API. Figure 2.1 shows a documentation of the schema created by Gousios and Spinellis [22].

2. Design and implementation of an extensible infrastructure for the collection of all events exposed through GitHub's API.

3. Development of a scalable mechanism for providing researchers with GitHub's data, based on distributing incremental data dumps using a peer-to-peer protocol.

4. Ability to download separate csv (comma separated values) files for tables such as pull request comments, commit comments [1] etc.

5. The provision of data that can track developers through both a projects process (issue tracking, wiki) and its corresponding source code without resorting to heuristics.

[1] Commit comments/messages: are comments written by a developer when a patch of code is comitted to the local git repository. These are usually short sentences describing the function of the patch of code.

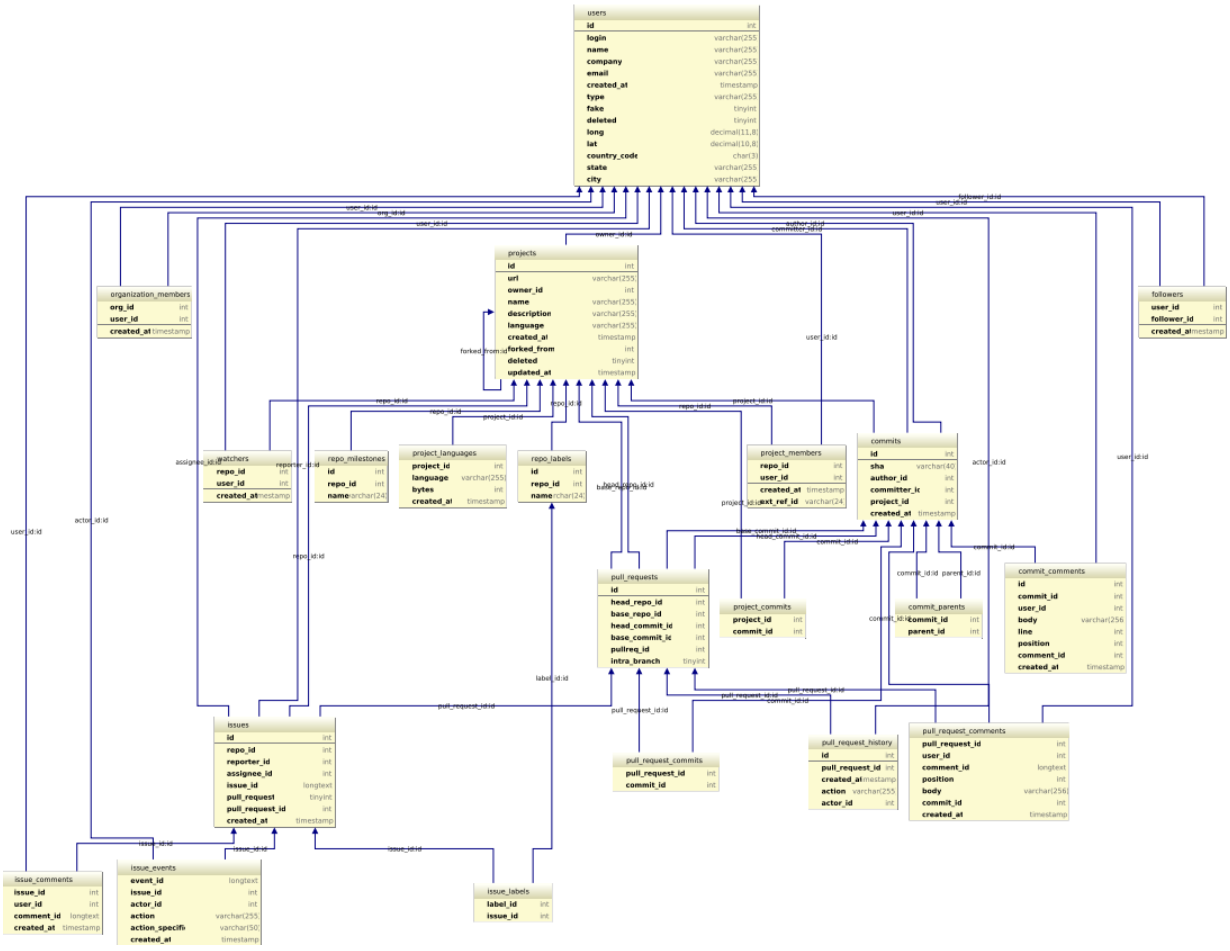


Figure 2.1: An overview of GitHub's data schema[22]

2.1.1 Emotional Analysis on GitHub Data

Section 2.1 describes **GitHub** as a collaborative software development website. Human collaboration inevitably evokes emotions like joy or sadness, which can affect the collaboration either positively or negatively [44]. Determining emotions in open source projects is particularly challenging as traditional ways of collecting information through experience reports, interviews with managers or surveys are more difficult due to the non-hierarchical structure of open source projects. The geographical distribution of developers among different regions and the volunteer basis in which developers contribute makes it harder to infer emotional information present in online software development projects.

Guillory et al. [23] and Hancock et al. [26] show that it is possible to detect emotions through computer-mediated communications systems, such as mailing lists and discussion boards. Mining emotions from online discussion boards is relevant when face-to-face meetings are not feasible. Most projects in a distributed environment have no personal interaction. In software engineering, emotion mining applied to text artifacts can be further used to provide hints on factors responsible for joy and satisfaction amongst developers (e.g., new release), or fear and anger (e.g., deadline or a recurring bug) and also provide a different perspective to interpret productivity and job satisfaction.

Murgia et al. [44] perform a feasibility study of emotions mining using *Parrott's framework* on the issue reports of *Apache software foundation.*

Parrott's Framework[45]

Parrott's framework classifies human emotions into a tree structure with three levels . Each level refines the granularity of the previous level. Level one consists of 6 primary emotions: *love, sadness, anger, joy, surprise* and *fear.* The other two levels consist of more fine grained versions of emotions present in the previous level. For example, the second level of **love** consists of *affection, lust* and *longing.* Level 3 of **love** consists of emotions such as: *passion, arousal, desire, etc.* Murgia et al. [44] only consider the 6 primary emotions for their work citing concise and intuitive nature of the primary emotions.

Emotion mining on issue commit comments

Murgia et al. [44] chose 392 random issue comments from the Apache software foundation and then proceeded to label them in the six primary emotions described by Parrott's framework. They had a total of sixteen people label a small number of comments in groups of two individuals. They found that on average $46.11 \pm 5\%$ of all comments had the same rating for all six emotions from both raters. The major results of [44] can be summarized as follows:

1. Emotions such as *love, sadness and joy* obtained the least agreement among raters.
2. Raters agree more on the absence of emotions. This means that raters had more confidence in identifying when an emotion is not present.
3. Knowing a context in which a comment is made does not play a significant role in the rating of emotions, but when it does, it seems to cast more doubt than confidence, unless more raters are used.

Sentiment analysis of commit comments on GitHub

Guzman et al. [24] use lexical sentiment analysis to study emotions expressed in commit comments of different open source projects and analyze their relationship with different factors such as used programming language, time and day of the week in which the commit was made, team distribution and project approval.

Guzman et al. [24] analyzed a total of 60425 commit comments from the *ghtorrent suite* [22]. The commit comments included 90 of the top-10 starred projects on GitHub. **SentiStrength** [54], a lexical sentiment analysis extraction tool specialized in dealing with short low quality texts was used for sentiment analysis of the commit comments. Thelwall et al. [57] show that, SentiStrength has good accuracy for short texts on Twitter tweets. Since GitHub commit messages are usually short and written in informal language, SentiStrength is a good candidate for analyzing emotions in commit messages.

SentiStrength assigns fixed scores to tokens in a dictionary where common emoticons are also included. Words with a negative emotion are given a value between [-5, -1] and words with a positive emotion are given a value in the [1, 5] range. The 1 and -1 values are used to give neutral scores to words, whereas 5 and -5 are used for words with a very positive and very negative emotion respectively. All the scores assigned by SentiStrength are integers. For example, love is assigned a score of 3, 1 and hate a 1, 4 score. Modifier words and symbols also alter the score.

The procedure used by Guzman et al. [24] to find the sentiments in the commit messages using SentiStrength is as follows:

1. The text in the commit message was divided into snippets of one or more sentences and assigned a positive and negative score to each of the sentences by taking the maximum and minimum scores among all the words in the sentence.

2. The positive and negative score of a snippet is calculated by taking the maximum and minimum scores of the sentences forming it.

3. The emotion of the entire commit was the positive and negative emotion score snippet average.

4. The whole commit message was assigned a score of zero when the positive and negative snippet average emotion scores are in the range $[-1, 1]$.

5. The commit emotion score is equal to the negative snippet average score when the snippets average negative emotion score times 1.5 is less than the average positive

score. When the opposite occurs, the commit is assigned the positive snippet average score.

6. The snippet average negative scores are multiplied by 1.5, because negativity is considered to be less frequent in human written texts.

Table 2.1 shows an example of SentiStrength scores in some commit comments on GitHub.

Sentence	Positive score	Negative score
Sigh? Its fixed, man rejoice!!	5	-2
Wow amazing thread! even If Im not a Rails developer!	5	-1
MY PRECIOUSSS!!!	5	-1
If PHP code is producing errors with register globals on you are terrible terrible programmer. If you are using magic quotes you are simply stupid.	1	-5
But this commit message makes me sad :cry	1	-5
This is really terrible - changing :private to :public without any deprecation warning? Not cool.	1	-5
This is really terrible - changing :private to :public without any deprecation warning? Not cool.	1	-5

Table 2.1: SentiStrength scores of some sentences

Sentiment Analysis of Security Discussions on GitHub

Pletea et al. [49] explore sentiment analysis of security related discussions on GitHub. Pletea et al. [49] state that it is important to perform sentiment analysis on such discussions so as to properly train developers to address security concerns in their applications, as well as the need to test applications thoroughly for security vulnerabilities in order to reduce frustration and improve overall project atmosphere. This is an another benefit of sentiment analysis in software engineering in addition to the ones mentioned in Chapter

1 and Subsection 2.1.1. Pletea et al. [49] reported that approximately 10% of all the discussions on GitHub were security related and they tended to have negative sentiments. Pletea et al. [49] analyzed 60,658 commits and 54,892 pull request comments.

Pletea et al. [49] used the Natural Language Text Processing (NLTK) tool [5] for sentiment analysis. Given an input text, NLTK outputs the probabilities that the text is neutral, negative or positive as well as an aggregate label (one of neutral, negative or positive) summarising the three scores. The probabilities for negative and positive will add up to 1, while neutral is standalone. If neutral is greater than 0.5 then the label will be neutral. Otherwise, the label will be negative or positive, whichever has the greater probability. The tool was trained on movie reviews and uses two classifiers, a Naive Bayes Classifier and a Hierarchical Classifier.

Using NLTK Pletea et al. [49] inferred that security-related discussions tend to be more emotional than non-security-related discussions.

2.2 Machine Learning Techniques for Sentiment Analysis

2.2.1 Input Feature Vectors

Bag of words

In a bag of words representation [53] each feature corresponds to a single word found in the training corpus, usually with case and punctuation removed. The Bag of words model is also known as *1-of-N(or 'one-hot')* encoding. Figure 2.2 shows an example of the bag of words model.

A common preprocessing step is to filter out *frequent* and *infrequent* words along with *stop-words* (stop words are functional or connective words that are assumed to have no information content).

In addition to the removal of frequent, infrequent, and stop words, a *stemming algorithm* is often used to make the features more statistically independent [63]. Stemming has the effect of mapping several morphological forms of words to a common feature. For example the words learner, learning, and learned would all map to the common stem learn, and this latter string would be placed in the feature set rather than the former three.

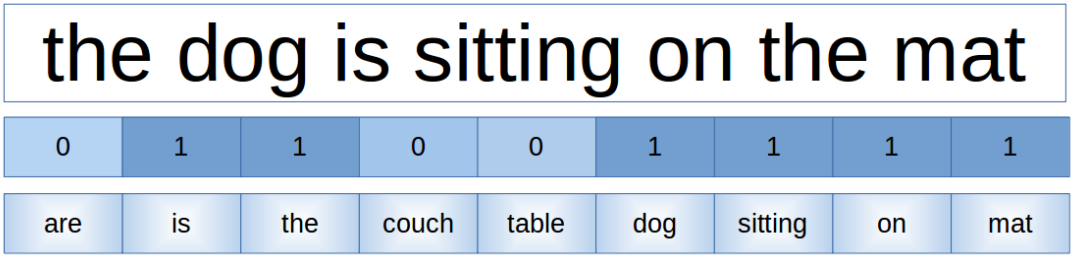


Figure 2.2: Bag of words feature vector for a sentence [18]

2.2.2 Term frequency−inverse document frequency (tf-idf)

Term frequency inverse document frequency (tf-idf) [31] is a statistical measure which is used to evaluate how important a word is in a collection of documents or a corpus. The importance of a particular word increases proportionally to the number of times a word appears in the document and is offset by the number of times the word appears in the corpus.

Tf-idf is one of the most popular features vector for text based applications like: *search engines*, *text summarization* and *recommender systems* [62].

Tf-idf is calculated by the product of *Term Frequency (TF)* and *Inverse Document Frequency(IDF)* which are defined as follows:

Term Frequency (TF): measures how frequently a term occurs in a document. It is usually normalized by the number of terms in the document (document length).

Inverse Document Frequency(IDF): measures how important a term is. It weighs down the frequent terms while scaling up the rare ones. It is calculated as eq. 2.1.

$$IDF(t) = \log \frac{N}{N(t)} \tag{2.1}$$

where N is the total number of documents and $N(t)$ is the total number of documents with the term t in it.

Haddi et al. [25] shows that simply using tf-idf as a preprocessing step for sentiment analysis on movie reviews sentiments datasets, leads to considerable improvements in *accuracy, precision, recall* and *F1*. For text query ranking applications, the sum of all the *tf-idf* terms in a text query is also used as a ranking metric for that query.

2.2.3 Word Vectors

Subsections 2.2.1 and 2.2.2 describe a feature vector for textual data where the length of the feature vector is the size of the entire vocabulary. Using the aforementioned approaches, there isn't much meaningful comparison that can be made between different feature vectors other than equality testing. Also, it is impossible to compare the similarity/dissimilarity of any two words/sentences using just the *bag of words* or *tf-idf*.

Word vectors (also known as distributed representation of words) on the other hand, embed words in a continuous vector space where semantically similar words are mapped to nearby points.

Distributed representation of words can be learnt with two approaches:

1. **Continuous Bag-of-Words model (CBOW)** [42]: In this approach, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction.

2. **Skip-Gram model** [43]: In this approach, the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weighs nearby context words more heavily than more distant context words.

According to Mikolov et al. [43] *Skip-gram works well with small amount of the training data, represents well even rare words or phrases*, whereas *CBOW can be several times faster to train than the skip-gram, and have slightly better accuracy for the frequent words.*

The concept of distributed representation of words can be traced back to Bengio et al. [4], where the goal was to learn the joint probability function of sequences of words in a language which is extremely difficult because of the curse of dimensionality. Bengio et al. [4] proposed to solve the curse of dimensionality by learning a distributed representation for words, where word vectors for a word were learnt by conditioning a word with the previous words. Bengio et al. [4] use a single hidden layer neural network with *tanh* nonlinearity followed by a softmax layer. Their approach can be summarized as follows:

1. Each word is mapped to a random fixed sized vector. These are the feature vectors.

2. The joint probability function of word sequences is modelled using feature vectors described in step 1.

3. The word feature vectors and the parameters of the probability function are learnt simultaneously.

The unnormalized log-probabilities for each output word y_i is given by eq. 2.2.

$$y = b + Wx + Utanh(d + Hx) \tag{2.2}$$

Using the trained word vectors, Bengio et al. [4] were able to get better **10-20%** improvements in perplexity, when compared to the state of the art method of smoothed trigrams [34] [7].

The word vectors we use are skip gram based word vectors described by Mikolov et al. [43]. Figure 2.3 shows the skip gram architecture used by Mikolov et al. [43].

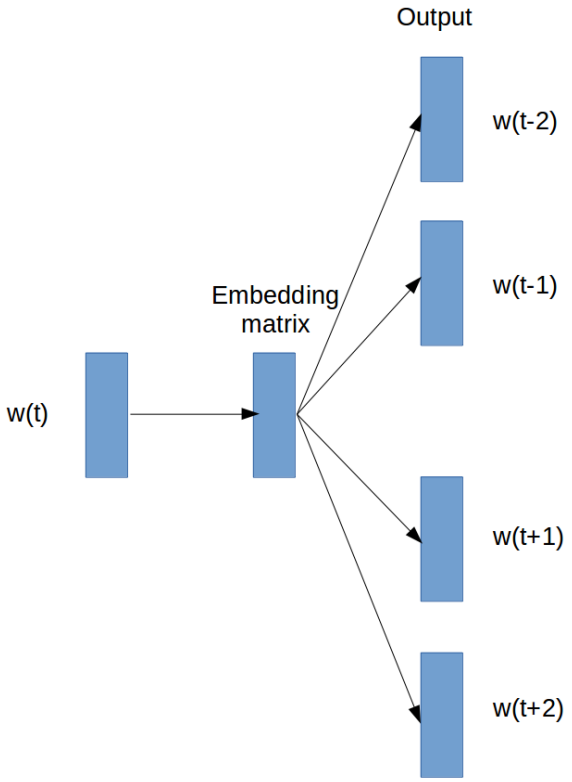


Figure 2.3: The Skip-gram model architecture. The training objective is to learn word vector representations that are good at predicting the nearby words [43]

Skip-gram models are trained using a binary classification objective (logistic regression) to discriminate the actual target word from the context words. This can be written as a softmax classifier as shown by eq. 2.3.

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{W} \exp(u_w^T v_c)} \tag{2.3}$$

where W is the vocabulary size, o is the output/target word, c is the context word, u_o^T is the word vector for the output word, u_w^T is the word vector for the output word w and v_c is the word vector for the context word. The issue with softmax comes from the fact the normalization term in the denominator of eq. 2.3 is computationally very expensive when we have a large vocabulary (usually of the order of tens of millions). To get around this problem, Mikolov et al. [43] use binary classification objective (logistic regression) to discriminate the real target words w_t from k imaginary (noise) words $\tilde{w}$, in the same context. This is also known as **negative sampling**. Figure 2.4 illustrates this for **CBOW** [20]. For skip-gram the direction is simply inverted.

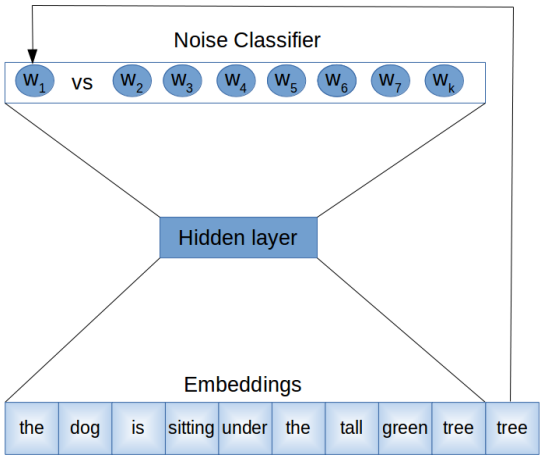


Figure 2.4: CBOW training using k negative examples.For skip-gram the direction is simply inverted. [20]

The objective function used by Mikolov et al. [43] is given by 2.4.

$$J_t(\theta) = log\sigma(u_o^T v_c) + \sum_{i=1}^{k} E_{j\sim P(w)}[log\sigma(-u_j^T v_c)] \tag{2.4}$$

where σ is the sigmoid function, o is the output/target word, c is the context word, u_o^T is the word vector for the output word, v_c is the word vector for the context word and $P(w)$

is the probability distribution from which the negative samples are taken from. Usually, $P(w)$ is the unigram distribution raised to the power 3/4 as shown by eq. 2.5. This power enables less frequent words to be sampled more often.

$$P(w) = U(w)^{3/4} \tag{2.5}$$

Equation 2.4 is then summed up for all the time steps as shown by eq. 2.6.

$$J_t(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta) \tag{2.6}$$

We trained the skip gram vectors using 64 negative samples on 400,000 random pull request comments from GitHub. These 400,000 pull request comments were not a part of the dataset we used for this project. These were random comments chosen from the openly available GHTorrent database (discussed in Section **??**). Table 2.2 shows the nearest words to some random words after 44100000 iterations.

As can be seen in Table 2.2, semantically similar words are closer to each other. Word vectors are also able to learn the relationship between the words.

Apart from **CBOW** [42] word vectors and skip gram word vectors [43], Pennington et al. [46] presented another popular approach to get word vectors called **Global Vectors(GloVe)**. For GloVe the objective function to minimize is given by eq. 2.7.

$$J_t(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - log P_{ij}) \tag{2.7}$$

where P_{ij} is the probability that word j appears in the context of word i, u_i and v_j are the word vectors of the two co-occurring words and $f()$ is a weighting function which should satisfy three properties:

1. $f(0) = 0$
2. $f(x)$ should be non decreasing
3. $f(x)$ should be relatively small for large values of x.

An example of $f(x)$ given by Pennington et al. [46] is shown by eq. 2.8.

$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases} \tag{2.8}$$

Table 2.2: Nearest words to some random words trained by Skipgram on 400k github pull request comments

Word	Nearest Words
check	checks, checking, checked, qpointers, fleko, detect, rp-, settingscache
data	stream, disk, payload, chunks, dataset, content, expressionvalueinfo, raw
since	likely, longer, especially, currently, however, even, became, albeit
always	false, truthy, guaranteed, true, never, beforehand, ever, null
reason	reasons, use-case, question, storemixin, motivation, point, ehashman, rubbed
ok	okay, yes, yeah, alright, ah, fine, yep, thanks
rather	instead, opposed, recommend, suggest, consider, without, simply, avoid
name	names, named, id, nam, rename, renamed, identifier, underscorized
change	changes, revert, pingall, changed, bump, nononono, punt, reverted
could	would, easily, please, maybe, bracers, configureflow, brandwe, possibly
even	likely, still, though, since, doubt, anyway, possibly, obviously
probably	think, maybe, perhaps, guess, suppose, wonder, might, definitely
test	tests, testing, suite, unit, cover, covers, pushapi, locatorio
want	need, wan, able, going, wanted, worthwhile, na, wants
right	yeah, yes, okay, ok, yep, yah, perfect, idiot
better	nicer, best, cleaner, clearer, appropriate, suitable, simpler, safer
class	classes, abstract, method, trait, interface, methods, fixedpathtypeddelimited, subclasses
like	similar, weird, promising, strange, odd, amiss, sujest, passregistry
move	put, moved, extract, moving, extracted, placed, hoist, encapsulate
call	calls, calling, called, invoke, invoking, invoked, clientinvocation, onrejected
set	setting, sets, unset, assign, overwrite, initialize, default, initialized
two	three, couple, several, multiple, many, various, four, consecutive
fix	fixing, fixes, fixed, investigate, rebase, correcting, tackle, merging
time	minutes, hour, hours, minute, day, seconds, cycles, interval
though	hmm, fine, certainly, still, guess, tbh, although, moment
version	versions, vers, newer, releases, bumped, release, -snapshot, bump
comment	comments, todo, fixme, note, documentation, javadoc, retracted, stableprovenance
object	objects, array, obj, easystagedataset, nulltask, collection, model, map
maybe	perhaps, probably, might, saying, think, wonder, possibly, could
pull	commit, pul, pr, issues, bbbcbad, diff-ea, maybe-upgrade, bfabc
variable	variables, varia, vars, var, constant, varaible, shadowing, resetboard
user	users, owner, superuser, person, customer, student, administrators, someone

2.2.4 TF-IDF weighted word vectors

To get a distributed representation of a sentence, the average of word vectors can be taken to get the vector representation. Another popular method is to take the weighted average the vectors of the words according to the tf-idf values. The weighted vector representation is then used to represent the vector representations of the sentence as a whole.

Wang [60] uses this approach to predict personalities of individuals from twitter.

2.3 Distance Metric Learning

Distance metric learning learns a distance function over a set of objects. The distance function describes the similarity and dissimilarity among the different objects. Distance metric learning is needed because each problem (such as recommendation systems, information retrieval, image compression) has its own semantic notion of similarity, which is often badly captured by standard metrics such as Euclidean distance. Distance metric learning learns a metric that assigns small (resp. large) distance to pairs of examples that are semantically similar (resp. dissimilar).

Learning a good distance metric in feature space is crucial in real-world applications, such as information retrieval for learning to rank, in face verification/identification, and in recommendation systems. For content based image retrieval systems, it is essential to use a well defined similarity criteria to define similarity between images. Many machine learning algorithms, such as K Nearest Neighbor (KNN), heavily rely on the distance metric for the input data patterns since KNN relies on labels of nearby objects to decide on the label of a new object. Metric learning can significantly improve the performance in classification, clustering and retrieval tasks.

A distance metric $D(x, y)$ should satisfy the following 4 properties

1. Non negativity: $D(x, y) \geq 0$.
2. Identity of indiscernibles: $D(x, y) = 0$, iff x=y, else it is called a pseudo distance metric.
3. Symmetry: $D(x, y) = D(y, x)$.
4. Subadditivity: $D(x, y) + D(y, z) \geq D(x, z)$.

In practice, metric learning algorithms ignore the condition of identity of indiscernibles and learn a pseudo-metric.

The field of distance metric learning can be divided into two main categories

1. Unsupervised distance metric learning.
2. Supervised distance metric learning: In this case the training examples are divided into pairwise constraints: the equivalence constraints where pairs of data points that belong to the same classes, and inequivalence constraints where pairs of data points belong to different classes. This is further divided into the following two categories:
   (a) Global distance metric learning: learns a distance metric in the global sense. The learned metric satisfies all the pairwise constraints simultaneously.
   (b) Local distance metric learning: learns a metric in the local setting. The learned metric only satisfies local pairwise constraints. This is very useful for information retrieval and K nearest neighbours classifiers since both of their performance is influenced by data instances close to test/query examples.

The pairwise constraints can be represented as follows:

If n is the number of data points, $C = x_1, x_2, x_3.....x_n$ are the collection of data points where $x_i \epsilon R^m$ is a data vector of m features, the set of equivalence constraints can be denoted by eq. 2.9.

$$S = (x_i, x_j) | x_i \text{ and } x_j \text{ belong to the same class} \tag{2.9}$$

and the set of inequivalence constraints are denoted by eq. 2.10.

$$D = (x_i, x_j) | x_i \text{ and to } x_j \text{ belong to different classes} \tag{2.10}$$

The distance metric matrix is denoted by $M \epsilon R^{m \times m}$ and the distance between two points x and y is given by eq. 2.11.

$$d_A^2(x, y) = ||x - y|_A^2 = (x - y)^T M (x - y) \tag{2.11}$$

M in eq. 2.11 is also called as **Mahalanobis matrix**.

2.3.1 Supervised Global Distance Metric Learning

Supervised Global Distance Metric Learning learns a metric which attempts to keep all data points within the same class close, while keeping all the data points in different classes far. Xing et al. [65] formulate the metric learning problem as a convex programming problem which learns a global distance metric that minimizes the distance between data points in the equivalence constraints, subject to the constraint that the data pairs in the inequivalence constraints are well separated.

Xing et al. [65] formulate the metric learning problem as follows:

$$min_{A \epsilon R^{mxm}} \sum_{(x_i,x_j)\epsilon S} ||x_i - x_j||_A^2 \text{ s.t. } A \geq 0, \sum_{(x_i,x_j)\epsilon D} ||x_i - x_j||_A^2 \geq 1 \tag{2.12}$$

Although eq. 2.12 is a convex programming objective, it is difficult to solve it efficiently for two reasons:

1. It does not fall into any special classes of convex programming such as quadratic programming or semidefinite programming.
2. The optimization objective is not scalable as it is quadratic in the number of features.
3. With objective function eq. 2.12 it is not possible to estimate the probability of how likely two points belong to the same class.

Kwok and Tsang [37] extend the optimization objective eq. 2.12 for non linear case with the use of kernels.

2.3.2 Supervised Local Distance Metric Learning

This subsections details some of the important supervised local distance metric learning algorithms.

Local Adaptive Distance Metric Learning

Supervised Local Distance Metric Learning attempts to learn feature weights that are adapted to the individual test examples. For a given test example x_0, the class posterior probability can be estimated as eq. 2.13.

$$\hat{P}(j|x_0) = \frac{\sum_{i=1}^{n} \theta(x_i \epsilon N(x_0)) \theta(y_i = j)}{\sum_{i=1}^{n} \theta(x_i \epsilon N(x_0))} \tag{2.13}$$

where n is the number of examples, $x_i \epsilon R^m$, and $y_i \epsilon 1, 2, 3....J$ classes.

Local Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) [2] is technique to find a linear combination of features that separates two or more classes. LDA computes the directions (linear discriminants) that will represent the axes that that maximize the separation between multiple classes and minimize the separation within classes. To find such a set of weights, LDA calculates the eigenvectors of the matrix T as shown by eq. 2.14.

$$T = S_w^{-1} S_b \tag{2.14}$$

where S_w is the within class covariance matrix (the weighted sum of covariance matrices of each class) and S_b is the between class covariance matrix. S_w^{-1} captures the compactness of each class, and S_b represents the separation of the class means. The transformed matrix can be represented by eq. 2.15.

$$y = S_T x \tag{2.15}$$

where S_T is formed by stacking the principal eigenvectors of T together.

Figure 2.5 shows an example of LDA when applied to the MNIST dataset. The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits from 0 to 9. Figure 2.5 shows that LDA is able to discriminate between different digits to some extent. The **x** and **y** axis of Figure 2.5 have no labels because this is simply a dimension reduction example.

Hastie and Tibshirani [27] state that, LDA can be localized with the following procedure:

1. Initialize distance metric $\sum$ as an identical matrix.
2. Calculate S_b and S_w using the points which are in the neighborhood of the testing point x_0 measured by distance metric $\sum$.

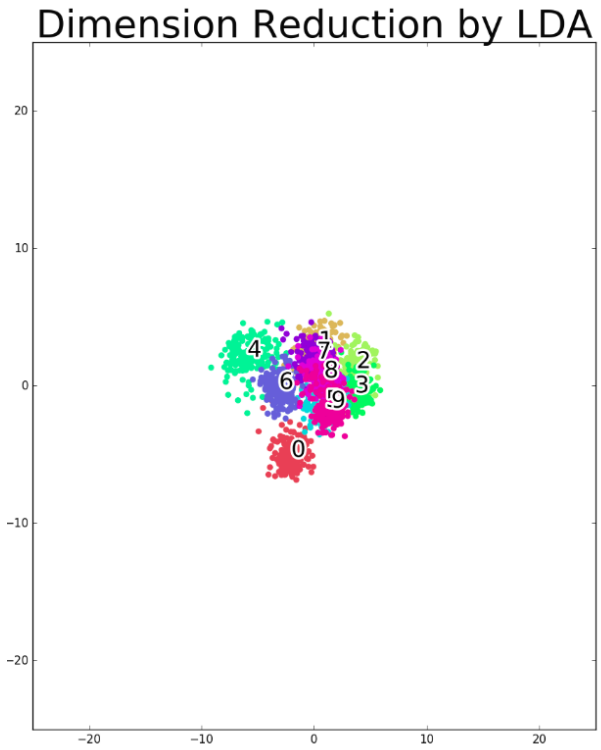


Figure 2.5: Example of LDA on the MNIST dataset[51]

3. Update the distance metric $\sum$ as equation 2.16.

$$\sum = S_w^{\frac{-1}{2}}[S_b^* + \epsilon I]S_w^{\frac{-1}{2}} \tag{2.16}$$

where S_b^* is $S_w^{\frac{-1}{2}} S_b S_w^{\frac{-1}{2}}$

Neighborhood Components Analysis

Neighborhood Components Analysis learns a Mahalanobis distance metric by maximizing the leave one out cross validation.

Given labelled data $L = (x_1, c_1), ...(x_n, c_n)$, NCA learns a distance matrix $Q = A^T A$, where A can be any matrix. This form of Q guarantees the distance metric to be positive semi-definite. The distance in eq. 2.11 can now be written as eq. 2.17.

$$d_A^2(x,y) = ||x-y||_A^2 = (x-y)^T A(x-y) = (Ax-Ay)^T(Ax-Ay) \tag{2.17}$$

The probability of a point x_i sharing the same label as x_j can be written as eq. 2.18.

$$p_{ij} = \frac{\exp(-||Ax_i - Ax_j||^2)}{\sum_{k \neq i} \exp(-||Ax_i - Ax_k||^2)} \tag{2.18}$$

The objective function of NCA is the maximization of the expected number of correctly classified points $f(A)$ given by eq. 2.19.

$$f(A) = \sum_{i=1}^{n} log(\sum_{j \epsilon C_i} p_{i,j}) \tag{2.19}$$

where C_i is the class i.

Yang and Jin [66] explain a few drawbacks of NCA such as:

1. NCA is not scalable, since its objective function is quadratic in the number of features.
2. Since NCA uses gradient descent, it is not guaranteed to converge to a global optima.
3. NCA can overfit, if the training data is less.

Relevant Component Analysis

Relevant Component Analysis (RCA) learns a full rank Mahalanobis distance metric based on the weighted sum of in-class covariance matrices. It does so by applying a global linear transformation to assign large weights to relevant dimensions and low weights to irrelevant dimensions. These relevant dimensions are estimated using chunklets. In RCA, a chunklet is defined as a subset of points that are known to belong to the same although unknown class.

The steps to perform RCA are as follows:

1. Center each chunklet by subtracting its mean.

	Linear	non linear
Global	PCA, MDS	ISOMAP, TSNE
Local	LLP, LLE, Laplacian Eigenmap	

Table 2.3: Unsupervised algorithms for dimension reduction

2. Compute the covariance matrix of the chunklets. For p points in k chunklets, each chunklet containing x_{ji} with mean as $\hat{m}_j$, the covariance matrix is computed as eq. 2.20.

$$\hat{C} = \frac{1}{p}\sum_{j=1}^{k}\sum_{i=1}^{n_j}(x_{ji} - \hat{m}_j)(x_{ji} - \hat{m}_j)^T \tag{2.20}$$

3. Whiten the covariance matrix in equation 2.20 by multiplying it with $W = \hat{C}^{\frac{-1}{2}}$. The Mahalanobis distance is the inverse of $\hat{C}$.

4. The new transformed space can be found by $x_{new} = Wx$.

Tsang et al. [58] show that RCA can also be kernelized.

2.4 Unsupervised Distance Metric learning

Unsupervised metric learning (also known as manifold learning) learns a low dimensional manifold where the geometric relationships between most of the data are preserved. Unsupervised Distance Metric learning is closely related to dimension reduction. Table 2.3 details the different unsupervised learning algorithms.

2.4.1 Linear Methods

The two main linear dimension reduction algorithms are

1. Principal Component Analysis (PCA)

2. Multidimensional Scaling (MDS)

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a linear dimensional reduction technique which finds a subspace that best preserves the variance of the data. Principal components are orthogonal directions that capture the most variance in the data.

If $v_1, v_2 ... v_d$ are the d principal components, $X = [x_1, x_2 .... x_n]$ is the data (column are the data points) PCA's objective function is to maximize the sample variance of the projected data (equation 2.21).

$$\frac{1}{n}\sum_{i=1}^{n}(v^T x_i) = v^T X X^T v \text{ s.t. } v^T v = 1 \tag{2.21}$$

Taking the Lagrangian of the equation 2.21 we get 2.22.

$$\max_v v^T X X^T v - \lambda v^T v \tag{2.22}$$

Taking the derivative of 2.22 we get eq 2.23.

$$(X X^T)v = \lambda v \tag{2.23}$$

Equation 2.23 shows that the principal components are the eigenvectors of the covariance matrix. λ shows the amount of variability captured along that principal component.

An alternative viewpoint on PCA is that PCA finds the vectors v such that projection onto the vectors yields the minimum mean squared reconstruction error as shown in eq. 2.24.

$$\frac{1}{n}\sum_{i=1}^{n} ||x_i - (v^T x_i)v|| \tag{2.24}$$

Figure 2.6 shows an example of PCA on a sample dataset. The arrows represent the principal axes of the data, and the length of the vector is an indication of how important that axis is in describing the distribution of the data. The **x** and **y** axis of Figure 2.6 have no labels because this is simply a dimension reduction example.

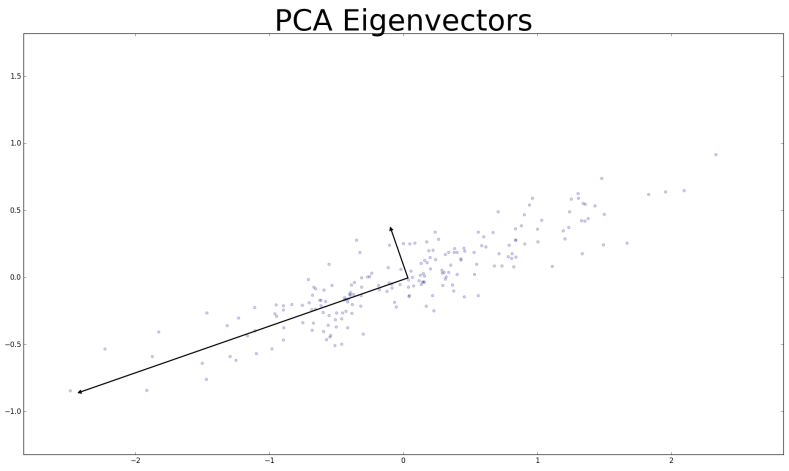


Figure 2.6: An example of eigenvectors found by PCA. [59]

Multidimensional Scaling (MDS) [36]

Multidimensional Scaling (MDS) finds the rank m projection that best preserves the interpoint distance (dissimilarity) given by a pairwise distance matrix D. It is a means of visualizing the level of similarity between examples of a dataset. The steps to perform MDS are as follows:

1. Compute pairwise distance matrix D.
2. Double center D by computing 2.25.

$$B = \frac{-1}{2} HDH \text{ where } H = I - \frac{1}{N} 11^T \tag{2.25}$$

   where $1 \epsilon R^m$ is an all one vector.

3. Compute the eigenvectors and eigenvalues of B.
4. The transformed matrix X is given by equation 2.26.

$$X_{new} = V^{mds} (\lambda^{mds})^{\frac{1}{2}} \tag{2.26}$$

   where V^{mds} are the top eigenvectors of B and λ^{mds} is the diagonal matrix of top eigenvalues of B.

The relationship between PCA and MDS is shown by equations 2.27 and 2.28.

$$V^{pca} = XV^{mds}, \lambda^{pca} = \lambda^{mds} \tag{2.27}$$

$$Y^{pca} = (\lambda^{pca})^{\frac{1}{2}} Y^{mds} \tag{2.28}$$

In the case of Euclidean metric the MDS only differs from PCA by starting with D and calculating X_{new}. When the distance metric is not Euclidean, it's better to use MDS.

Locally-Linear Embedding (LLE)

PCA (2.4.1) and MDS (2.4.1) are both linear dimension reduction/metric learning techniques. Being linear algorithms, they cannot find nonlinear structure in the data.

Locally-Linear Embedding (LLE) [52] preserves the local order relation in the low dimensional embedding space and the original space. Each data point in the observation space is a weighted average of its neighbors. LLE assumes that each point and its neighbors lie on a locally linear patch of a manifold.

The steps for LLE can be summarized as follows:

1. Represent each point x_i as a weighted sum of its nearest n neighbors.
2. Solve for the weight matrix by minimizing the reconstruction error given by eq. 2.29.

$$\begin{aligned} &\underset{W}{\text{minimize}} \quad ||x_i - \sum_{j=1}^{n} W_{ij} x_{ij}||^2 \\ &\text{subject to} \quad W_{ij} = 0 \text{ if } x_j \text{ is not a neighbor of } x_i \text{ and } \sum_{j=1}^{n} W_{ij} = 1 \end{aligned} \tag{2.29}$$

3. Model the low dimension data points as the weighted sum of the low dimensional neighbors using the same weight matrix calculated in eq. 2.29. The reconstruction error in this case can be written as eq. 2.30.

$$\begin{aligned} &\underset{Y}{\text{minimize}} \quad ||y_i - \sum_{j=1}^{n} W_{ij} y_{ij}||^2 \\ &\text{subject to} \quad \sum_{i=1} y_i = 0 \text{ and } \sum_{i} y_i y_i^T / r = I \end{aligned} \tag{2.30}$$

4. Equation 2.30 can be written as eq. 2.31.

$$\underset{Y}{\text{argmin}} \quad ||Y^T(I-W)^T(I-W)Y||^2 \tag{2.31}$$

   The optimum low dimensional embeddings are the lowest $m+1$ eigenvectors of $(I-W)^T(I-W)$ (since the first eigenvector would be zero, as $I-W$ is the graph laplacian and the number of zero eigenvalues of a graph laplacian gives the number of connected components).

There are several different versions of LLE

1. Modified LLE [68]
2. LLE with Hessian Eigenmaps [13]
3. Local tangent space alignment [69]

One disadvantage of LLE is the need to compute the SVD of $(I-W)^T(I-W)$ for every test point. To overcome this, Zhang et al. [67] state that, for a new test point x' , its internal coordinates on the manifold can be computed by $y' = \sum_{i=1}^{n} \alpha K(x_i, x')$ where $K(x_i, x') = exp(-\frac{||x_i - x'||^2}{2\sigma^2})$ and α can be computed by the complete data $(X; Y)$.

Figure 2.7 shows the results when LLE is applied to the MNIST dataset. The **x** and **y** axis of Figure 2.7 have no labels because this is simply a dimension reduction example. Figure 2.7 shows that LLE is more effective than LDA (Figure 2.5) in segregating the digits.

ISOMAP [56]

One of the challenges in distance metric learning is that, Euclidean distance would not be a good distance metric for checking the similarity of the two arbitrary points in a nonlinear manifold. Instead of Euclidean distance, *geodesic* distance (distance along the manifold) should be used to calculate the similarity of the points. In the case of dimension reduction, points far apart on the manifold should be far apart in the low dimensional representation. Thus only the geodesic distance is capable of revealing the true low-dimensional geometry of a nonlinear manifold.

To solve the problem of finding the true low-dimensional geometry, ISOMAP uses eigen analysis for nonlinear embedding.

The detailed steps for ISOMAP are as follows:

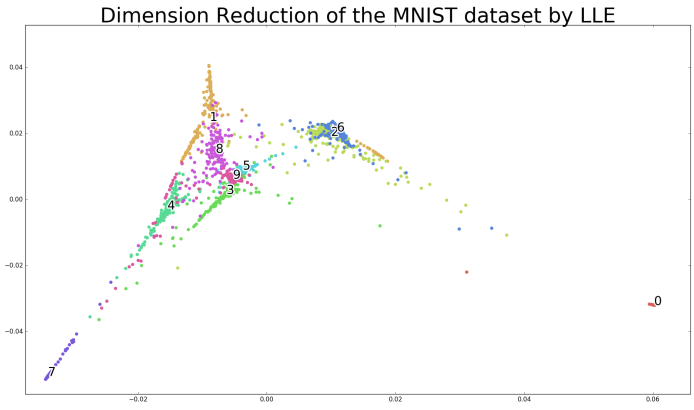


Figure 2.7: LLE applied to the MNIST dataset

1. Construct a neighborhood graph for every point on the manifold by using Euclidean distance as the edge distance. Assign each edge in the graph a weight of $d_x(i, j)$.

2. Estimate the geodesic distance $d_M(i, j)$ on the manifold M. The geodesic distance is estimated as the shortest path in the graph constructed in step 1. For neighboring points, Euclidean distance is a good approximation to geodesic distance; for far away points, geodesic distance can be estimated by adding up a sequence of short hops between neighboring points which is actually the shortest paths in a graph with edges connecting neighboring data points.

3. Apply MDS to the matrix of geodesic distance to construct the low dimensional embedding.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE [41] visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map which can then be visualized in a scatter plot. It is a variation of Stochastic Neighbor Embedding (SNE) [29]. t-SNE and SNE belong to a class of algorithms which use mutual information as a measurement of the differences of probability distribution between the observed space and the embedded space.

$t - SNE$ can be described in two steps :

1) Construct a probability distribution on the high dimensional feature space such that nearby objects have a high probability of being picked, and dissimilar points have an small

probability of being picked. This can be done with the help of a Gaussian Kernel. The probability p_{ij} denoting the similarity of points x_i and x_j is modelled as eq. 2.32. Figure 2.8b shows a comparison of some of the dimension reduction methods discussed here.

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2||)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2||)} \tag{2.32}$$

The bandwidth of the Gaussian kernels σ_i, is set in such a way that the perplexity of the conditional distribution equals a predefined perplexity using the bisection method.

2) Model the lower (2 or 3) dimensional representation as a Student-t distribution. If we had modelled the lower dimensional distribution as a Gaussian Distribution, then we might have had an imbalance in the distribution of the distances of a points neighbors.

The similarities q_{ij} between two points y_i and y_j, in the learnt lower dimensions is modelled as eq. 2.33

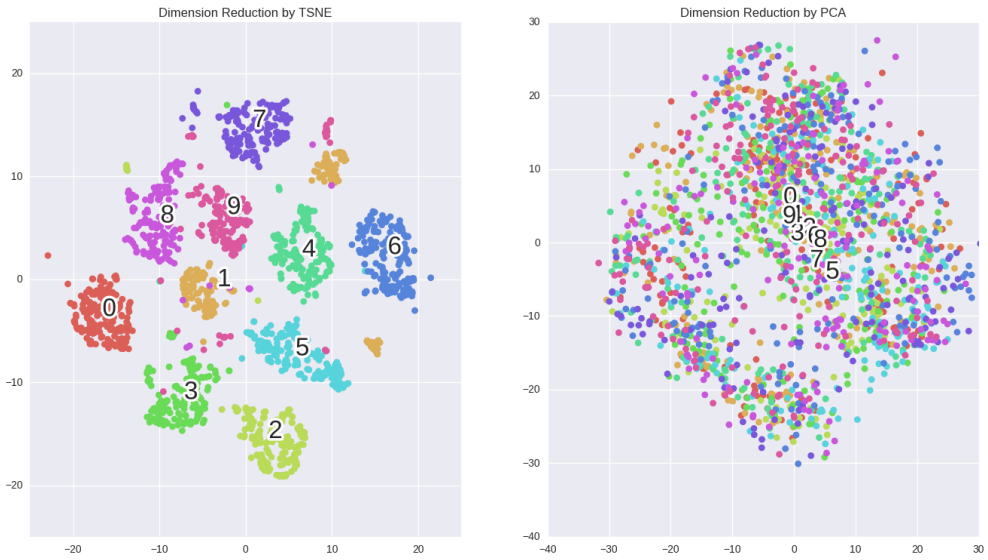
$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq m}(1 + ||y_k - y_m||^2)^{-1}} \tag{2.33}$$

Then minimize the Kullback-Leibler (KL) divergence between the two distributions using Gradient Descent. The KL divergence of distribution Q from the distribution P is written as eq. 2.34.

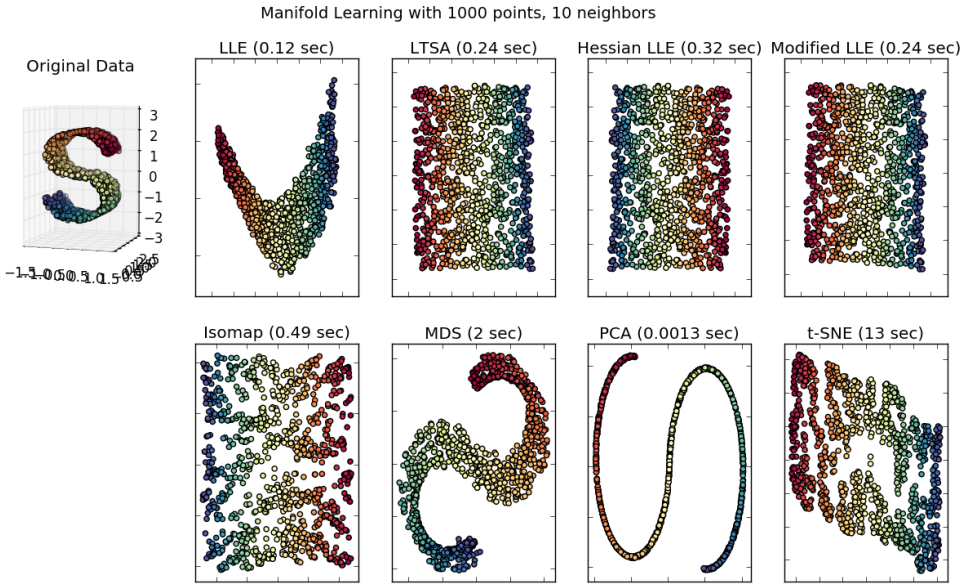
$$KL(P||Q) = \sum_{i \neq j} p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{2.34}$$

Figure 2.8a shows a comparison when t-SNE and PCA are applied to MNIST for reduction to 2 dimensions. Figure 2.8a shows that, t-SNE is way better when in segregating the digits into different clusters than PCA.

To end the section of distance metric learning, Figure 2.8b shows a comparison of most of the metric learning algorithms discussed here on a sample dataset. Figure 2.8b shows the difference in the way different algorithms discussed in this section reduce the dimensions of a dataset.



(a) Difference between TSNE and PCA on dimension reduction on MNIST dataset



(b) Comparison of Manifold Learning algorithms

Figure 2.8: A) TSNE on the MNIST dataset B)Comparison of manifold learning algorithms

2.5 Interaction Process Analysis

Interaction Process Analysis (IPA) is a sentiment analysis technique developed by Robert Bales [3]. IPA measures a set of labels/sentiments which model the interactions between small groups. IPA consists of twelve categories.

1. Shows solidarity
2. Shows tension release
3. Agrees
4. Gives suggestion
5. Gives opinion
6. Gives orientation
7. Asks for orientation
8. Asks for opinion
9. Asks for suggestion
10. Disagrees
11. Shows tension
12. Shows antagonism

Given the highly technical nature of the pull request comments on GitHub, the IPA categories are best suited for modelling the pull request comments.

IPA categories have an advantage that they can be mapped to a certain set of behaviours and emotions using **Affect Control Theory**. Affect Control Theory models social interactions on an individual level and assigns an affective meaning to all actions/behaviours and physical entities. Affective meaning varies on three dimensions: **Evaluation, Potency**, and **Activity**.

Evaluation contrasts pleasant and good with unpleasant and bad. Potency contrasts powerful and strong with powerless and weak. Activity contrasts lively and active with

quiet and inactive. EPA values for over a thousand of words have been calculated by sociologists by conducting large scale surveys.

Table 2.4 shows the sample behaviors and EPA values of the IPA categories. These values are taken from the paper *Modeling interactions in small groups* by Heise [28].

IPA	Sample behaviours	E	P	A
Shows solidarity	help, compliment, gratify	1.78	1.29	.21
Shows tension release	josh, laugh with, cheer	1.48	.91	1.12
Agrees	agree with, understand, accomodate	1.6	.91	1.12
Gives suggestion	encourage, cue, coach	1.28	1.18	.25
Gives opinion	evaluate, analyze, entreat	.16	.59	-.02
Gives orientation	inform, educate, explain	1.68	1.62	-.14
Asks for orientation	quiz, question, ask about	.50	.62	.45
Asks for opinion	consult, prompt, query	.48	.74	.16
Asks for suggestion	entreat, ask, beseech	.30	.24	.09
Disagrees	disagree with, ignore, hinder	-1.00	.35	.45
Shows tension	fear, cajole, evade	-.89	-.16	.35
Shows antagonism	argue with, deride, defy	-.82	.71	1.32

Table 2.4: Categories of Interaction Process Analysis (IPA), With Sample Behaviors, and Average Evaluation, Potency, and Activity Scores for Sample Behaviors

Using the EPA values for some emotions shown in Table 2.5, we can find the closest emotions to each IPA label using Euclidean distance. Table 2.6 shows the closest two emotions (from the ones mentioned in Table 2.5) to each IPA category using the EPA mapping.

Emotions	E	P	A
Calm	2.18	1.23	2.18
Thanks	2.76	1.54	2.76
Nervous	-1.37	-1.05	-1.37
Careless	-1.76	-1.10	-1.76
Angry	-1.61	-0.5	-1.61
Defensive	-0.28	-0.05	-0.28
Cautious	1.39	0.19	1.39
Happy	3.25	2.62	3.25
Aggressive	0.27	1.57	0.27
Sorry	-0.49	-0.37	-0.49

Table 2.5: EPA values of some emotions

IPA	Closest 2 emotions based on EPA mapping
Shows Solidarity	Thanks, Cautious
Gives Suggestion	Thanks, Cautious
Disagrees	Defensive, Angry
Shows Tension	Defensive, Nervous
Shows Antagonism	Defensive, Angry
Gives Orientation	Thanks, Calm
Asks for Opinion	Defensive, Cautious
Gives opinion	Defensive, Cautious
Asks for Orientation	Defensive, Aggressive
Agrees	Thanks, Cautious
Asks for Suggestion	Defensive, Cautious
Shows Tension Release	Aggressive, Thanks

Table 2.6: Closest two emotions to each IPA category based on EPA mapping

Chapter 3

Dataset description

3.1 Dataset

We now explain the dataset used in this project. Initially we were using GitHub's public API to mine the pull request comments of different repositories. Due to the API's hit rate limit, we decided to use the data from GHTorrent [22]. Chapter 2 Section 2.1 describes the GHTorrent dataset in more detail. We used the GHTorrent's GitHub dump up to February 2017. The entire dump consisted of the following tables: *commit_comments, commit_parents, commits, followers, issue_comments, followers, issue_comments, issue_events, issue_labels, issues, organization_members, project_commits, project_languages, projects, pull_request_comments, pull_request_commits, pull_request_history, pull_requests, repo_labels, repo_milestones, users, project_members* and *watchers*.

For this project we only used the following tables: *pull_request_comments, pull_request_commits, pull_request_history,* and *pull_requests*. We loaded the aforementioned csv files into a MySQL database for ease of querying.

We randomly selected 834 pull requests from GitHub and a total of 3000 pull request comments. Out of the 834 pull requests, 41 were open , 343 were closed without being merged and 450 were merged. We set out to annotate these comments into the twelve Interaction Process Analysis labels and ten emotions. The ten emotions we used were *Thanks, Sorry, Calm, Nervous, Careless, Cautious, Aggressive, Defensive, Happy* and *Angry*. These particular set of emotions were chosen because all of them a a mapping in the EPA space. This mapping is later used to get a mapping from the IPA categories to the emotions.

Initially, I was the only one who annotated the 3000 comments. In order to get an unbiased dataset, we put up the dataset on Amazon Mechanical Turk to be labelled by three people. We obtained University of Waterloo's Ethics committee's approval to get the dataset labelled on Mechanical Turk. We had a screening process to choose the best people to get the dataset labelled. Fifty pull requests comments (from the 3000 pull request comments) were opened to the general public. This means that anyone registered with Amazon Mechanical Turk could attempt the task of annotating the comments. These fifty comments were very intuitive and easy to label in the IPA categories and the ten emotions. They did not require deep understanding of the task at hand. Most of these fifty comments had emotional and sentimental keywords such as: *great, sorry, happy, good, bad, wrong etc.*, which made them easy to label.

Calvin Zhou, an undergraduate research assistant also separately annotated these fifty pull request comments. This was required to avoid any bias in marking the fifty comments.

The best performing people were given an option to continue with the task. The reason for choosing participants this way was that we wanted to filter out people who were not qualified to complete the task or the people who were simply marking randomly. A total of fifteen people took part in the task of annotating these fifty pull request comments.

We chose the top three people by the following criteria:

1. People who had the annotations most similar to mine and Calvin's.
2. People who had some experience in programming any language and people who had heard of GitHub.

We hosted the website for Mechanical Turk (written in Python/Django with a MySQL database) on pythonanywhere.com.

We provided detailed instructions on how to annotate a particular pull request comment. Each pull request comment could be annotated into a maximum of three IPA categories and a maximum of three emotions. Participants were given an option to choose from a checkbox. Appendix A has more details on the web interface used for annotation.

Apart from annotating a pull request sentence into the IPA categories and the emotions, the participants were also asked to filter out any unnecessary sections of code in the comment. Although we had performed a preprocessing (using regular expressions to filter out code segments) to remove sections of code from the comments, there were some instances where the code was not formatted properly enough to be filtered by the regular expressions.

A few examples of the sentences and their corresponding IPA and emotions are shown in Table 3.1.

3.2 Annotated Dataset description

After all three people had completed the annotations on Mechanical Turk, we assigned each IPA/emotion a value of 1 if three out of four people had assigned it a value of 1 (I was the fourth person), else the IPA/emotion was assigned a value of 0. Table 3.2 and Table 3.3 show the number of comments with each emotion/IPA after annotations. Table 3.3 shows the number of comments with each IPA categories in the full dataset. Table 3.2 shows the number of comments with each emotion in the full emotions dataset.

Figures 3.1a and 3.1b show the number of normalized instances present in the merged/ open/and closed(without being merged) pull requests.

Based on the nature of conversations taking place at a particular pull request, we believe that the following six categories (three classification tasks) appropriately describe the dataset.

1. **Agrees/Shows solidarity/Shows Tension release vs Disagrees/Shows Antagonism/Shows Tension**: To get the dataset for these categories, we followed the following steps:

    - From the full IPA dataset we collected all comments which were annotated at least one of Agrees/Shows solidarity/Shows Tension release. There were 1680 comments in this category.
    - From the full IPA dataset we collected all comments which were annotated at least one of Disagrees/Shows Antagonism/Shows Tension. There were 1301 comments in this category.

    We did not find any overlap between these two categories.

2. **Gives opinion/Gives Suggestion/Gives orientation vs Asks for opinion/Asks for orientation/Asks for suggestion**: To get the dataset for these categories, we followed the following steps:

    - From the full IPA dataset we collected all comments which were annotated at least one of Gives opinion/Gives Suggestion/Gives orientation. There were 1952 comments in this category.

COMMENT	IPA	EMOTIONS
If the pull request,for #195 gets given the go ahead, I'd be happy to convert this to be a new descriptor-style GSObject., It looks very good, and great to have the new improved factors for HLR and FWHM.	Shows_Solidarity,Gives_orientation	Calm,Happy
As has happened before, I have to confess to not feeling very qualified to comment on this Tree stuff, having not coded up anything similar in the past myself...	Gives_orientation,Shows_Tension	Nervous,Cautious
Great that you harmonized these,params to the python-layer style.	Shows_Solidarity,Agrees,	Thanks,Happy
As the status of,is not changing, would it be faster to move this,switch outside the ?	Asks_for_orientation,Asks_for_opinon	Nervous,Cautious
Is this replicating code in,? If so, can we do without the former now?	Asks_for_orientation,Asks_for_opinon,	Nervous,Cautious,
Am I being totally dense, or is this a square region?, (If so, I was a little confused by the mention of R at line 390)	Asks_for_orientation,Asks_for_opinon	Nervous,Cautious
Sorry.,I started this file by copying the Airy version and forgot to edit this description.	Gives_orientation	Sorry,Careless

Table 3.1: Sample sentences and their IPA/emotions

Emotions	Aggressive	Angry	Calm	Careless	Cautious	Defensive	Happy	Nervous	Sorry	Thanks
Number of instances	375	16	1606	49	1567	335	69	380	76	138

Table 3.2: Number of emotions categories instances

IPA	Shows solidarity	Shows tension release	Agrees	Gives suggestion	Gives opinion	Gives orientation	Asks for orientation	Asks for opinion	Asks for suggestion	Disagrees	Shows tension	Shows antagonism
Number of instances	1329	27	514	696	1098	1351	608	376	199	1207	17	162

Table 3.3: Number of IPA categories instances

- From the full IPA dataset we collected all comments which were annotated at least one of Asks for opinion/Asks for orientation/Asks for suggestion. There were 1025 comments in this category.

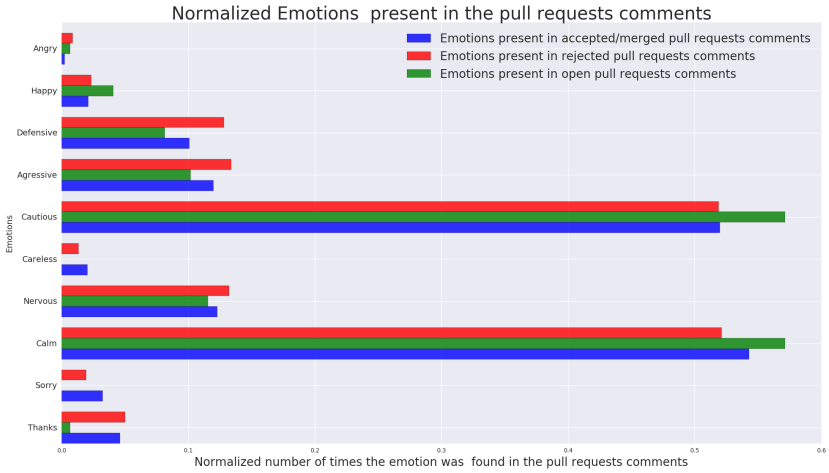
There were 254 sentences which had an overlap in both categories which were disregarded. This is because wanted explicitly classify one category against the other.

3. **Positive vs Negative Emotions**: To get the dataset for these categories, we followed the following steps:

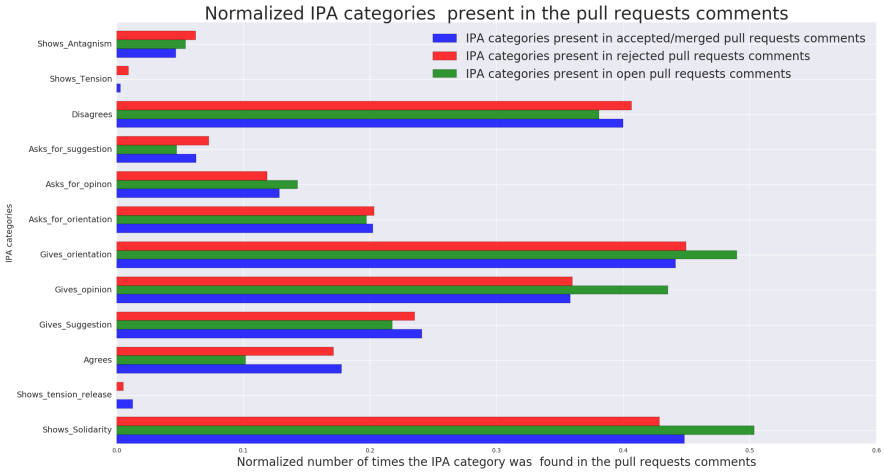
   - From the full emotions dataset we collected all comments which were annotated at least one of *Thanks, Calm, Happy,* and *Cautious*. There were 2555 comments in this category.
   - From the full emotions dataset we collected all comments which were annotated at least one of *Sorry, Nervous, Careless, Aggressive , Defensive,* and *Angry*. There were 437 comments in this category.

   There were 534 comments which had an overlap in both the categories which were assigned the negative label. This is because most of these 534 comments either had more negative labels or only had *Cautious* from the positive category and some label from the negative category. Finally we had 971 negative comments and 2021 positive comments.

Figure 3.2 shows the number of normalized instances (normalized means divided by total number of comments in that state of pull request:open/close/merged) of each of the aggregated categories by merged/open/closed (without merged) pull request comments. Even



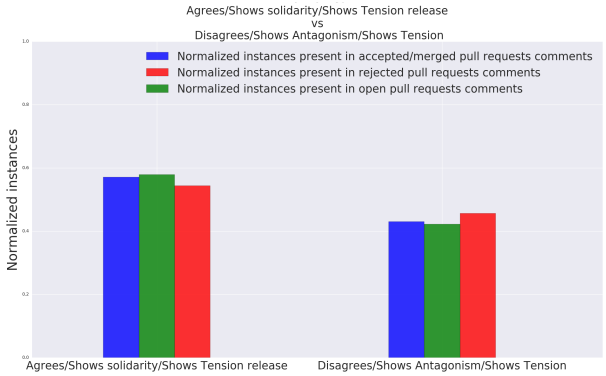
(a) Normalized emotions instances of open/closed/merged pull requests



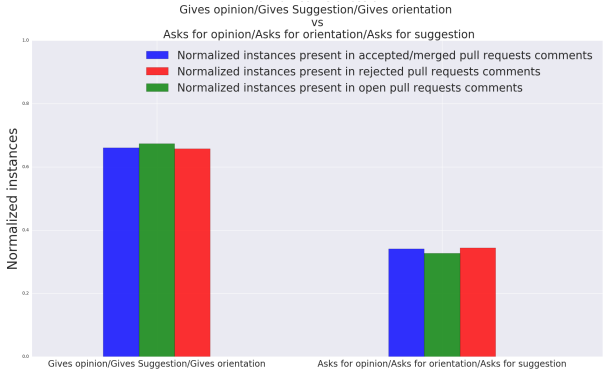
(b) Normalized IPAs instances of open/closed/merged pull requests

Figure 3.1: Normalized emotions and IPAs instances among pull requests

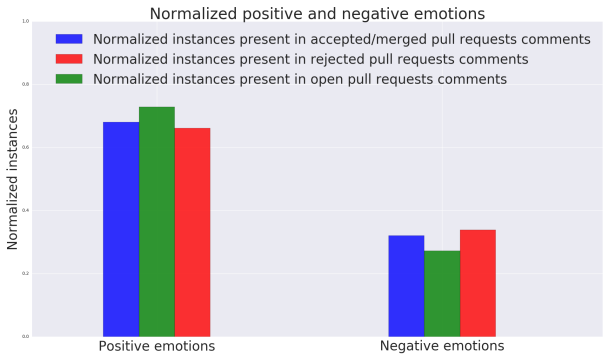
though we only had 3000 comments, figures 3.1 and 3.2 show that merged pull requests have higher proportion of positive emotions and positive IPA categories.



(a) Normalized Agrees.../Disagrees... instances of open/closed/merged pull requests



(b) Normalized Gives opinion.../Asks for opinion... instances of open/closed/merged pull requests



(c) Normalized positive and negative emotion instances of open/closed/merged pull requests

Figure 3.2: Number of normalized instances merged/open/closed(without merged) pull request comments in aggregated datasets

Aggregated IPA	Closest two emotions
Agrees/ Shows solidarity/ Shows Tension release	Thanks, Cautious
Disagrees/ Shows Antagonism/ Shows Tension	Defensive, Angry
Gives opinion/ Gives Suggestion/ Gives orientation	Cautious,Thanks
Asks for opinion/ Asks for orientation/ Asks for suggestion	Defensive, Cautious

Table 3.4: Closest two emotions to the grouped IPAs

3.2.1 Emotions inference from IPA mapping

One of the benefits of using IPA is that, ACT provides a mapping from IPA to the EPA space as shown in Table 2.4, which gives us the ability to map the aggregated IPA categories to a set of emotions. Table 2.5 shows the EPA values of the emotions used in this project. Although we do not use this mapping in further experiments, the purpose of descrbing the mapping of IPA to emotions is to show that emotions can be inferred from IPAs.

We can infer the emotions closest to a group of IPA labels as follows

1. Average out the EPAs of each IPA category.
2. Use Euclidean distance to find the closest set of EPA's corresponding to the emotions.

Following the aforementioned steps, we found the closest two emotions to each of the aggregated IPA categories. Table 3.4 shows the closest emotions to the grouped IPAs. Note that here we only consider the four grouped IPA categories. The fifth and sixth category defined earlier already consists of emotions.

Chapter 4

Machine Learning experiments

This chapter applies the state of the art machine learning models to the pull request comments annotated on mechanical turk. We first discuss two state of the art deep learning methods for sentence classification: **Recurrent Neural Nets** along with its variants and **Convolutional Neural Nets**.

4.1 Recurrent Neural Nets

Recurrent Neural Nets (RNNs) are a class of neural nets which make use of sequential information. They have a feedback loop which allows them to capture memory of past events and make use of that memory in the future. They also tie weights at each time step.

RNNs have been immensely successful in natural language processing. In case of sentences, RNNs are conditioned on all the previous words in a sentence.

Given a list of word vectors $x_1, x_2, x_3.....x_n$ the RNN equation for each time step can be written as eq. 4.1.

$$\begin{aligned} h_t &= \sigma(W^{hh}h_{t-1} + W^{hx}x_t) \\ y_t &= softmax(w^S h_t) \end{aligned} \tag{4.1}$$

where σ is the sigmoid function, W^{hh} is the weight matrix for the hidden layer, W^{hx} is the weight matrix between the hidden layer and input layer, w^s is the softmax weight matrix, h_t is the hidden layer output and y_t is the output.

A popular variant of RNN used in this thesis is called the **Bidirectional RNN**. In bidirectional RNNs, there are two RNNs one of which processes the input sentence from left to right and the other from right to left. Both of their outputs are combined at the end for further classification.

The loss function that RNNs optimize can either be cross entropy or perplexity. RNNs are trained using backpropagation through time, which is essentially the same as back-propagation after unwinding the RNN through all the time steps. Due to the fact that the same weight matrix gets multiplied at each time step, RNNs suffer from the problem of vanishing gradient. In case of NLP, it can cause the RNN to forget an important piece of information at earlier time steps. To overcome this, there are two variants of RNNs namely: **Long Short Term Memory (LSTM)** and **Gated Recurrent Units GRUs**. Both LSTMs and GRUs perform the following two functions:

1. Store information from past timesteps, and
2. Backpropagate the gradient/error at different strengths depending on the inputs.

4.1.1 Long Short-Term Memory (LSTM)[30]

Long short-term memory (LSTM) is a variant of RNN that helps avoid the vanishing gradient problem, by preserving the error that can be backpropagated through time which allows the RNN to learn across many time steps.

LSTMs contain gated cells which allow memory to be retained by storing information from past time steps. Since LSTMs are gated, information can be written to the cells when required. A LSTM can be described by the following sets of equations 4.2.

$$\begin{aligned}
\text{Input gate } i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1}) \\
\text{Forget gate } f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) \\
\text{output gate } o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) \\
\text{New memory cell } \tilde{c}_t &= tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) \\
\text{Final Memory cell } c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
\text{Final hidden state } h_t &= o_t \circ tanh(c_t)
\end{aligned} \tag{4.2}$$

where σ is the sigmoid function, $\circ$ is the element wise operator, W and U are the weight matrices for the respective gates, h_t is the hidden layer output and i_t is the input.

4.1.2 Gated recurrent units GRUs [8]

Gated recurrent units GRUs are another gated variant of RNNs, which have fewer parameters than LSTM though have similar performance to LSTMs.

GRUs can be summarized by equations 4.3.

$$\begin{aligned} \text{Update gate } z_t &= \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}) \\ \text{Reset gate } r_t &= \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}) \\ \text{New memory content } \tilde{h}_t &= tanh(Wx_t + r_t \circ Uh_{t-1}) \\ \text{Final memory } h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t \end{aligned} \tag{4.3}$$

where σ is the sigmoid function, $\circ$ is the element wise operator, W and U are the weight matrices for the respective gates, h_t is the hidden layer output and i_t is the input.

As per equations 4.3, GRUs will ignore the previous hidden state if the reset gate r_t is 0. The update gate z_t controls how much of the past state would matter now. If the update gate is close to 1, then the GRU can copy information from that unit through many time steps, thus avoiding the vanishing gradient problem.

It has been observed that for GRUs, units with short term dependencies have very active reset gates.

4.1.3 Convolutional Neural Nets (CNNs)[40]

Convolutional Neural Nets (CNNs) have been very successful for image and video classification tasks due to the fact that they preserve the spatial structure of an image while training. A CNN typically comprises of a few convolutional layers optionally followed by a fully connected neural network. Convolutional neural nets might also have a *pooling layer*, which combines the outputs of the previous layer into one value which later goes as an input to the next layer. With pooling, it becomes possible to use inputs of various sizes for training the same neural network. Some of the useful pooling methods are: *max pooling, average pooling* and *L2-norm pooling.*

Kim [33] presented an approach to use CNNs for sentence classification. For a given sentence Kim [33] used word vectors for every word in a sentence to construct a 2D matrix for each sentence, resembling that of an image. Given a set of word vectors $x_1, x_2, x_3....x_n$ for each word in a sentence, the 2D matrix is constructed by concatenating each word vector vertically $x_1 \bigoplus x_2 \bigoplus x_3..... \bigoplus x_n$. The resultant matrix is then passed through three filters

(a hundred of each) of sizes: $3 \times k, 4 \times k, 5 \times k$ (k is the dimension of the word vector) following which is a max-pooling layer followed by a dropout and softmax layer.

Kim [33] used four different variants of CNNs:

1. CNN-Rand: Word embeddings initialized with random word vectors.
2. CNN-Static: Word embeddings initialized with google word vectors as word embeddings.
3. CNN-Non-static: Word embeddings initialized with google word vectors, which are fine tuned i.e. the error is back propagated into the word embeddings.
4. CNN-Multi-channel: Word embeddings with two word embedding matrices initialized with google word vectors, though only one channel is fine tuned.

We use all of the above four variants for our experiments.

4.2 Experiment and Results

We first tried to predict all the IPA categories and the emotions. This was done in order to get a baseline for the results. We used Linear SVM to train and predict all the IPA and emotion categories. Since, this was was done just to get a baseline, we used tf-idf as feature vectors. Tables 4.1 and 4.2 show the results when we tried to predict all IPA categories and all emotions. We used $K = 5$ fold cross validation (with random seed and K-1 folds were used for training and the last fold for testing) and grid search to search for the optimal parameters for all the algorithms. The results were not good which led us to the belief that our dataset is either very noisy or we do not have enough labelled examples to learn from.

We ran a total of twelve algorithms on the aggregated datasets mentioned previously in Chapter 3. For RNNs, we implemented both bidirectional RNNs: stacked LSTMs and stacked GRUs. For both RNNs, we implemented two word embeddings versions: random word embedding and google word embeddings. The output from both directions of the RNN was concatenated for the softmax layer. We used $K = 5$ fold cross validation (with random seed and K-1 folds were used for training and the last fold for testing) and grid search to search for the optimal parameters for all the algorithms. For each algorithm we report the best average precision, recall, F1 and accuracy over five fold validation. The metrics

IPA Category	Precision	Recall	F1	Accuracy
Shows Solidarity	63.2	55.7	56.8	65.3
Shows tension release	20.0	6.7	10.0	99.1
Agrees	**95.2**	56.1	**64.0**	88.0
Gives Suggestion	54.4	34.5	33.4	76.5
Gives opinion	62.6	50.0	51.4	68.9
Gives orientation	62.0	**58.3**	58.6	64.5
Asks for orientation	88.8	37.7	36.2	81.7
Asks for opinion	30.0	24.1	22.9	87.5
Asks for suggestion	60.0	9.1	10.6	93.4
Disagrees	67.9	55.4	56.6	68.1
Shows Tension	40.0	26.7	30.0	**99.5**
Shows Antagonism	50.0	10.4	13.2	94.6

Table 4.1: One vs All classification results for IPA categories

Emotion	Precision	Recall	F1	Accuracy
Thanks	**100.0**	44.1	54.7	96.8
Sorry	93.5	46.1	58.7	**98.4**
Calm	65.2	75.8	69.3	64.8
Nervous	42.4	24.2	23.6	87.4
Careless	41.3	10.3	15.7	98.4
Cautious	64.8	**75.9**	**69.8**	65.8
Aggressive	58.1	23.9	25.2	87.7
Defensive	30.0	15.1	16.7	88.8
Happy	6.7	1.5	2.5	97.7
Angry	0	0	0	0

Table 4.2: One vs All classification results for Emotions

reported in this section for each algorithm are the best average metrics. For the non-deep learning algorithms (Linear SVM, Random forests, Logistic Regression and Gaussian Naive Bayes) the best average precision/recall/F1 may be from different hyperparameters. The meaning of this is that we do a separate cross validation to find the best performing metric. One cross validation is done to performed to find the best average precision. Another cross validation is done to find the best performing F1. Similarly, a seperate cross validation is done to find the best performing recall and accuracy. We consider F1 to be the best metric across algorithms, the other metrics are shown in order to highlight how each algorithm could perform.

Table 4.3 describes the input feature vector for each of the algorithms used in classifying the aggregated datasets explained in Chapter 3.

Algorithm	Feature Vector
Convolutional Neural Nets: (all variants)	A 3D feature vector with the word vector for every word. The word vector could be a google word vector or a random word vector. Each sentence was padded to a fixed sentence length of 56 (this was the max sentence length in the dataset) by 0 to use CNNs efficiently.
Recurrent Neural Nets: LSTM and GRUs	A 3D feature vector with the word vector for every word. The word vector could be a google word vector or a random word vector. There was no padding since we used dynamic RNNs.
Linear SVM, Random Forests, Logistic Regression, Gaussian Naive Bayes	Average of tf-idf weighted google word vectors.

Table 4.3: Feature vector description for each algorithm

The tuning parameters for the different algorithms are described as follows:

1. Convolutional Neural Nets: We performed a grid search over the number of filters, the sizes of filters, dropout rate and the learning rate. The number of filters were varied from $50, 100$ to 150. The learning rate was varied from 0.1 to 0.0001 in multiples of 0.1. The dropout rate was varied from 0.1 to 0.8 in steps of 0.1.

2. Recurrent Neural Nets (LSTMs and GRUs): We performed a grid search over the number of hidden layers, the number of LSTM/GRUs in each layer and the learning rate. The learning rate was varied from 0.1 to 0.0001 in multiples of 0.1. The number of hidden layers were varied from 1 to 3. The number of LSTM/GRU units in each layer were varied between $1024, 512, 256$ and 128.

3. Linear SVM: We used soft-margin SVM and performed a grid search for the regularization parameter C over the values: $0.1, 0.2, 0.5, 0.7, 0.5, 1, 5, 10, 15, 20, 30, 50, 100, 150$ and 200.

4. Logistic Regression: We used logistic regression with $L2$ regularization and performed a grid search for the regularization parameter C over the values: $0.001, 0.01, 0.1, 0.2, 0.5, 0.7, 0.5, 1, 5, 10, 15, 20, 30$, $50, 100, 150$ and 200.

5. Gaussian Naive bayes: We did not use any class priors for gaussian naive bayes.

6. Random Forests: We performed a grid search for the number of trees in the forest over the values: $10, 50, 100, 150, 200, 150, 300$ and 500.

We refer the three classification experiments as three tasks:

- Task 1: Agrees/Shows solidarity/Shows Tension release vs Disagrees/Shows Antagonism/Shows Tension.
- Task 2: Gives opinion/Gives Suggestion/Gives orientation vs Asks for opinion/Asks for orientation/Asks for suggestion.
- Task 3: Positive vs negative emotions.

The input to each algorithm is described as follows:

- For all RNNs we used the word vector (either random or google word vector) of each word and implemented dynamic RNN in tensorflow.

- For all CNNs we we used the word vector (either random or google word vector) for each word, and constructed a 2D matrix for a sentence. We padded the sentences to make all the sentences of same length.
- For linear SVM, logistic regression, random forests and gaussian naive bayes, we used the tf-idf weighted average of google word vectors.

4.2.1 Results

Table 4.4 shows the results for task 1. Table 4.5 shows the results for task 2. Table 4.6 shows the results of classification of task 3.

Algorithm\Metric	Precision	F1	Recall	Accuracy
CNN-rand	70.9	73.6	98.8	64.8
CNN-static	78.2	72.3	84.9	**66.5**
CNN-non-static	77.9	**74.6**	96.93	66.4
CNN-multichannel	**81.4**	74.2	96.3	66.4
Static Bidir LSTM RNN	75.9	73.4	97.7	64.2
Rand Bidir LSTM RNN	69.9	73.2	89.8	64.0
Static Bidir GRU RNN	74.3	73.1	95.1	63.9
Rand Bidir GRU RNN	69.8	73.3	89.8	64.0
Logistic Regression	65.2	74.1	**100**	64.75
Linear SVM	65.02	73.2	86.6	64.4
Random Forests	64.9	70.5	77.1	63.6
Gaussian Naive Bayes	61.2	69.6	80.6	60.5

Table 4.4: Measures of performance for classification for Agrees/Shows solidarity/Shows Tension release vs Disagrees/Shows Antagonism/Shows Tension

Algorithm\Metric	Precision	F1	Recall	Accuracy
CNN-rand	75.6	80.5	95.7	70.2
CNN-static	82.5	80.3	90.9	71.3
CNN-non-static	80.3	80.8	91.2	72.1
CNN-multichannel	83.2	**81.0**	91.2	**72.3**
Static Bidir LSTM RNN	81.2	80.8	97.6	69.9
Rand Bidir LSTM RNN	75.3	80.2	93.5	69.7
Static Bidir GRU RNN	**83.8**	80.3	98.6	68.8
Rand Bidir GRU RNN	75.7	80.2	93.4	69.6
Logistic Regression	73.9	**81.0**	**100**	71.8
Linear SVM	73.8	**81.0**	93.4	71.9
Random Forests	74.7	79.3	84.8	70.8
Gaussian Naive Bayes	72.9	77.7	83.2	68.5

Table 4.5: Measures of performance for Gives opinion/Gives Suggestion/Gives orientation vs Asks for opinion/Asks for orientation/Asks for suggestion

Algorithm\Metric	Precision	F1	Recall	Accuracy
CNN-rand	75.4	79.9	97.6	67.8
CNN-static	77.1	79.9	97.6	67.5
CNN-non-static	70.7	80.1	98.5	67.8
CNN-multichannel	71.2	**80.5**	99.0	67.9
Static Bidir LSTM RNN	75.4	80.3	**100.0**	67.1
Rand Bidir LSTM RNN	71.7	79.4	97.2	66.4
Static Bidir GRU RNN	**78.0**	80.3	100	67.2
Rand Bidir GRU RNN	72.0	76.2	87.1	63.9
Logistic Regression	69.9	**80.5**	**100.0**	**68.0**
Linear SVM	70.5	**80.5**	98.5	67.8
Random Forests	68.4	**80.5**	99.0	67.7
Gaussian Naive Bayes	70.9	29.1	41.3	44.2

Table 4.6: Measures of performance for positive vs negative emotions

4.3 Observations and Discussions

Since we started out with a small dataset, we consider F1 score to be a better estimator of performance over K-Fold cross validation.

The observations from the results can be summarized as

- We observed that for CNNs, the minimum learning rate of 0.001 was required to avoid having high bias. Any learning rate below that would lead to high bias resulting in underfitting.
- For RNNs a learning rate of 0.01 was good enough to avoid high bias.
- For CNNs, the architecture proposed by Kim [33] worked the best. A hundred filters each of sizes $3 \times 300, 4 \times 300$ and 5×300, dropout of 0.5 performed the best.
- Using pre trained word vectors always resulted in an increase in precision, recall, F1 and accuracy. This was observed for CNNs, LSTM RNNs and GRUs RNNs.
- In case of tasks 2 and 3 CNN-Multichannel had the best F1 score. In case of task 1 CNN-non-static had the best F1 score.
- Static bidirectional GRU RNN achieved the best precision in task 2 and 3. CNN-multichannel achieved the best precision in task 1.
- In tasks 1,2 and 3 logistic regression achieved the best recall, though static bidirectional LSTM RNN achieves the same recall in task 3.
- For task 1, CNN static achieves the best accuracy. For task 2 CNN multichannel achieves the best accuracy. For task 3 logistic regression achieves the best accuracy.
- In tasks 2 and 3, linear SVM also achieved the best F1. For task 3 random forests also achieved the best F1 score.

Chapter 5

Metric Learning experiments

Chapter 2 Section 2.3 discusses some of the important distance metric learning algorithms. This chapter focuses on two algorithms that are used in this thesis namely: **Large Margin Nearest Neighbor (LMNN)** and **Information Theoretic Metric Learning (ITML)**.

5.1 Large Margin Nearest Neighbor

Large Margin Nearest Neighbor (LMNN) [61] learns a Mahalanobis distance metric for kNN classification which is optimized with the goal that k-nearest neighbors always belong to the same class while examples from different classes are separated by a large margin. This approach is different from the ones used in Section 2.3 where most of the approaches minimize the pairwise distances between all similarly labeled examples. Most classification algorithms do not require similarly labelled examples to be clustered together. This is because, to achieve good generalization in classification setting, a good distance metric should not only achieve high consistency in the neighborhood, but also maintain large margin at the boundaries between different classes. LMNN also does not make any assumptions about the distribution of the data.

The optimization goal of LMNN is quite similar to that of SVMs [10]. They both have a convex optimization objective and a goal of margin maximization.

LMNN aims to learn a linear transformation L, which can be used to compute squared distances (eq 5.1).

$$D(x_i, x_j) = ||L(x_i - x_j||^2 \tag{5.1}$$

where x_i and x_j are any two data points. Some of the terminology specific to the LMNN paper by Weinberger et al. [61] is as follows:

1. Target neighbors: Each data point x_i has a set of k neighboring data points with the same label as x_i. Weinberger et al. [61] uses $\eta_{ij}\ \epsilon\ [0, 1]$ to indicate whether input x_j is a target neighbor of input x_i. The matrix η_{ij} is fixed during training.

2. Cost function: The cost function of LMNN is written as eq. 5.2.

$$\epsilon(L) = \sum_{ij} \eta_{ij} ||L(x_i - x_j)||^2 + c \sum_{ijl} \eta_{ij}(1 - y_{il})[1 + ||L(x_i - x_j)||^2 - ||L(x_i - x_l)||^2]_+ \tag{5.2}$$

   The first term penalizes large distances between each input and its target neighbors (not between all similarly labeled examples.), while the second term penalizes small distances between each input and all other inputs that do not share the same label. The second term denotes the standard hinge loss ($[z]+ = max(z, 0)$) and $c > 0$ is some positive constant (set by cross validation).

3. Large Margin: The second term in equation 5.2 reflects the large margin. For each input x_i, the hinge loss is incurred by differently labeled inputs whose distances do not exceed, by one absolute unit of distance, the distance from input xi to any of its target neighbors. The cost function thereby favors distance metrics in which differently labeled inputs maintain a large margin of distance and do not threaten to invade each others neighborhoods.

4. Convex optimization: Equation 5.2 can be reformulated as an instance of semidefinite programming [70]. A semidefinite program (SDP) is a linear program with the additional constraint that a matrix whose elements are linear in the unknown variables is required to be positive semidefinite. SDPs are convex; thus, with this reformulation, the global minimum of equation 5.2 can be efficiently computed. The SDP for equation 5.2 can be written as eq. 5.3.

$$\begin{aligned} & min \sum_{ij} \eta_{ij}(x_i - x_j)^T M(x_i - x_j) + c \sum_{ij} \eta_{ij}(1 - y_{il})\xi_{ijl} \\ & \text{subject to } \xi_{ijl} \geq 0 \\ & M \geq 0 \\ & (x_i - x_l)^T M(x_i - x_l) - (x_i - x_j)^T M(x_i - x_j) \geq 1 - \xi_{ijl} \end{aligned} \tag{5.3}$$

   where $M = L^T L$, and ξ_{ijl} are the slack variables.

5.2 Information-Theoretic Metric Learning

Information-Theoretic Metric Learning (ITML) [11] is a novel method of learning the Mahalanobis matrix by minimizing the logDet divergence between an initial starting metric and the learnt metric. The LogDet divergence is a loss function that describes the distance between positive definite matrices. ITML is quite different from all the algorithms described in section 2.3. ITML offers the following advantages:

1. Though the algorithms described in section 2.3 have good classification results, their constraints do not generalize beyond the class instances. ITML allows arbitrary linear constraints on the Mahalanobis matrix such as similarity or dissimilarity constraints, and relations between pairs of distances. ITML can also incorporate prior information regarding the distance function itself.

2. Most of the algorithms in section 2.3 require eigenvalue decompositions, an operation that is cubic in the dimensionality of data or in some cases quadratic in the number of examples. ITML on the other hand is fast and scalable.

ITML aims to learn the a positive semi-definite matrix A which parametrizes the Mahalanobis distance as shown in equation 5.4.

$$d_A(x_i, x_j) = (x_i - x_j)^T A(x_i - x_j); \tag{5.4}$$

The goal of ITML is to learn a similarity metric d_A which is close to some starting metric d_{A_0}. Instead of comparing the distance metrics it compares the relative entropy of the gaussians formulated by them and minimizes the *KL divergence* between them.

$$\begin{aligned} d_A(x, y) &\rightarrow N(x|\mu, A) \\ d_{A_0}(x, y) &\rightarrow N(x|\mu, A_0) \end{aligned} \tag{5.5}$$

The optimization problem can be formulated by eq 5.6.

$$\begin{aligned} &\min_A \int N(x|\mu, A_0) \log \frac{N(x|\mu, A_0)}{N(x|\mu, A)} dx \\ &\text{subject to } d_A(x_i, x_j) \leq u \text{ (i,j)}\epsilon \text{ S} \\ &d_A(x_i, x_j) \geq l \text{ (i,j)}\epsilon \text{ D} \\ &A \geq 0 \end{aligned} \tag{5.6}$$

where S is the set of similar examples, D is the set of dissimilar examples, l and u are predetermined thresholds (determined by 5th and 95th percentile of distribution) for similarity and dissimilarity measures.

The ITML optimization problem (eq. 5.6) can be formulated into a Bregman optimization problem by minimizing the LogDet divergence subject to linear constraints. The LogDet divergence is defined by eq. 5.7.

$$D_{ld}(X, Y) = trace(XY)^{-1} - logdetXY^{-1} - d \tag{5.7}$$

5.3 Experiment and Results

We use the aggregated dataset from Chapter 3. In order to apply ITML and LMNN to our problem, we constructed tf-idf weighted average of google word vectors feature vector for each pull request comment. We then proceeded to apply LMNN and ITML to the dataset. We used $K = 5$ fold cross validation (with random seed and K-1 folds were used for training and the last fold for testing) and grid search to search for the optimal parameters for all the algorithms.

The hyperparameter to tune in LMNN is k, the number of nearest neighbors. We used a grid search over the values $3, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 90$ and 100 to find the best value of k. The hyperparameter to tune in ITML is the number of constraints. We did a grid search over the values $200, 300, 400, 500, 800, 1000, 1200, 1500$ and 1600 to find the optimal value of the number of constraints. We used $PyMP$, an $OpenMP$ styled multiprocessing library in python to parallelize our grid search.

As earlier, we refer the three classification experiments as three tasks:

- Task 1: Agrees/Shows solidarity/Shows Tension release vs Disagrees/Shows Antagonism/Shows Tension.
- Task 2: Gives opinion/Gives Suggestion/Gives orientation vs Asks for opinion/Asks for orientation/Asks for suggestion.
- Task 3: Positive vs negative emotions.

The tuning parameters for the different algorithms are described as follows:

1. Linear SVM: We used soft-margin SVM and performed a grid search for the regularization parameter C over the values: $0.1, 0.2, 0.5, 0.7, 0.5, 1, 5, 10, 15, 20, 30, 50, 100, 150$ and 200.

2. Logistic Regression: We used logistic regression with $L2$ regularization and performed a grid search for the regularization parameter C over the values: $0.001, 0.01, 0.1$, $0.2, 0.5, 0.7, 0.5, 1, 5, 10, 15, 20, 30$, $50, 100, 150$ and 200.

3. Gaussian Naive bayes: We did not use any class priors for gaussian naive bayes.

4. Random Forests: We performed a grid search for the number of trees in the forest over the values: $10, 50, 100, 150, 200, 150, 300$ and 500.

5. Fully connected neural network: We performed a grid search over the number of hidden layers, number of sigmoid activated neurons in each layer, batch size and the learning rate. The learning rate was varied from 0.00001 to 0.1 in multiples of 10. The number of hidden layers were varied from one to three. The number of sigmoid activated neurons were varied between $1024, 512, 256$ and 128.

We form the feature vector after transforming the input tfidf weighted google word vectors by ITML and LMNN before classifying them using linear SVM, gaussian naive bayes, random forests, logistic regression and a fully connected deep neural net. Cross entropy loss was used to train the fully neural network. The feedforward neural network was run for 35 epochs. No increase in any metric was observed after 35 epochs.

5.3.1 Results

For each algorithm we report the best average precision, recall, F1 and accuracy over five fold validation. The metrics reported in this section for each algorithm are the best average metrics. The best average precision, F1 and recall may be from different hyperparameters. The meaning of this is that we do a separate cross validation to find the best performing metric. One cross validation is done to performed to find the best average precision. Another cross validation is done to find the best performing F1. Similarly, a seperate cross validation is done to find the best performing recall and accuracy. We consider F1 to be the best metric across algorithms, the other metrics are shown in order to highlight how each algorithm could perform.

The results are summarized in the following tables.

- Table 5.1 shows the results for task 1.
- Table 5.2 shows the results for task 2.
- Table 5.3 shows the results for task 3.

Algorithm\Metric	Precision	F1	Recall	Accuracy
FFNN with ITML	**77.1**	**74.7**	99.7	65.4
FFNN with LMNN	73.4	73.8	96.8	64.6
SVM with ITML	67.8	73.4	84.7	65.3
SVM with LMNN	68.8	72.3	83.2	65.8
Logistic Regression with ITML	66.3	74.0	**99.9**	65.2
Logistic Regression with LMNN	67.8	73.1	88.0	66.0
Gaussian Naive Bayes with ITML	61.5	70.9	83.9	60.6
Gaussian Naive Bayes with LMNN	64.0	70.1	79.9	64.1
Random Forests with ITML	65.8	71.2	78.2	64.0
Random Forests with LMNN	70.6	70.6	75.9	**66.1**

Table 5.1: Measures of performance with metric learning for Task 1

Algorithm\Metric	Precision	F1	Recall	Accuracy
FFNN with ITML	**78.7**	**81.4**	98.9	71.9
FFNN with LMNN	76.3	79.8	96.7	70.1
SVM with ITML	73.8	81.2	92.7	**72.3**
SVM with LMNN	76.1	79.7	90.9	70.9
Logistic Regression with ITML	74.2	**81.4**	**100**	**72.3**
Logistic Regression with LMNN	73.9	80.3	98.7	71.2
Gaussian Naive Bayes with ITML	73.2	79.3	86.5	70.2
Gaussian Naive Bayes with LMNN	73.0	77.2	82.3	68.4
Random Forests with ITML	72.5	80.2	85.8	72.1
Random Forests with LMNN	74.6	79.6	85.9	71.4

Table 5.2: Measures of performance with metric learning for Task 2

Algorithm\Metric	Precision	F1	Recall	Accuracy
FFNN with ITML	77.3	*80.5*	**100**	67.6
FFNN with LMNN	**83.4**	**80.6**	**100**	**68.7**
SVM with ITML	74.3	80.3	98.3	68.1
SVM with LMNN	73.0	78.8	92.4	66.9
Logistic Regression with ITML	70.1	*80.5*	**100**	68
Logistic Regression with LMNN	70.3	80.2	99.6	67.4
Gaussian Naive Bayes with ITML	70.4	37.0	25.3	42.5
Gaussian Naive Bayes with LMNN	71.5	30.3	30.0	48.3
Random Forests with ITML	69.4	80.4	98.4	67.9
Random Forests with LMNN	70.5	78.2	95.5	**68.7**

Table 5.3: Measures of performance with metric learning for task 3

5.4 Observations and Discussion

After cross validation, we found out the best parameters for the feed forward neural network were

- Two hidden layers with 512 and 256 sigmod activated neurons.
- Learning rate of 0.001.

As mentioned in Chapter 4, since we started out with a small dataset, we consider F1 score to be a better estimator of performance over K-Fold cross validation. The important observations can be summarized as follows:

- For task 1, feed forward neural networks with ITML as a preprocessing step leads to same max F1 score as all the algorithms in Table 4.4.
- Feed forward neural network(FFNN) with ITML as a preprocessing step gives the highest F1 score for task 2. Logistic regression with ITML also achieves the same F1 score.
- For task 3 feed forward neural networks with LMNN as a preprocessing step gives the highest F1 score.
- For task 2 we achieved the same best recall with ITML as we did without ITML.
- For task 3 we achieved the same best recall with ITML as we did with without ITML.

The results in this chapter also show that sometimes such as in task 2, a simple logistic regression model with L2 regularization was able to achieve the same best F1 score. Simpler models are better because they prevent overfitting.

Appendix B shows the variation of the performance metrics over the hyperparameters of ITML and LMNN.

The following important observations can be made regarding the plots in appendix B.

- Figure B.1 shows the variation in the measures of performance versus the number of constraints in ITML for task 1. For precision, F1, recall and accuracy , we observed that the metrics first increased reaching a maximum between 200-800 constraints, after which they either remained constant or decreased.

- Figure B.2 shows the variation in the measures of performance versus the number of constraints in ITML for task 2. For this task, we observed that the metrics generally increased when the number of constraints were increased before dropping off near the end (around 1500-1600 number of constraints).

- Figure B.3 shows the variation in the measures of performance versus the number of constraints in ITML for task 3. For F1, recall and accuracy, we observed that increasing the number of constraints had almost no effect on the metrics. For precision, we observed that it reached a maximum between 400-800 constraints before becoming constant. For random forests, precision reached a maximum at 1600 constraints.

- Figure B.4 shows the variation in the measures of performance versus the number of k nearest neighbors in LMNN for task 1. We observed that the precision and F1 increased as we increased the number of nearest neighbors. For recall, we observed a drop when we increased the number of nearest neighbors. For accuracy, we observed that it reached a maximum between 20-50 nearest neighbors before decreasing. This was mostly due to the fact the the model seemed to optimize the number of false positives which led to an increase in the number of false negatives.

- Figure B.5 shows the variation in the measures of performance versus the number of k nearest neighbors in LMNN for task 2. We observed that recall, F1 and accuracy increased before dropping off at 100 nearest neighbors. Precision did not seem to follow any pattern as the number of neighbors were increased.

- Figure B.6 shows the variation in the measures of performance versus the number of k nearest neighbors in LMNN for task 3. We observed that recall, F1 and accuracy almost stay constant as k is increased. Precision did not seem to follow any pattern as the number of neighbors were increased.

We now compare how SentiStrength performs on some of the comments from our negative emotions dataset. Table 5.4 shows an example of **SentiStrength** scores on some of the negative emotion comments. These negative comments are from the labelled dataset Table 5.4 shows that the **SentiStrength** tool is unable to recognize negative emotion appropriately.

Sentence	Positive score	Negative score
I don't think this is correct. If I'm looking at it right, it would default enabled to FALSE if does not exist	1	1
This one should have been left out of the repository	1	-1
I think you might need to prefix the string with ˆ like above, because of some other processing we do later on the chain with error messages. Also, if we don't allow the comma at all in a collection title, the reserved string ',,' doesn't make sense, sin	2	-1
This line doesn't do anything?!	2	-1
It's best we simply close this pr	2	-1
I believe you're incorrect here:1 for true 0 for false, since you're doing numeric tests and not using it as return values. We've also got to think about backwar	2	-2
Why the split between impl and int-tests ? Do we foresee a api/spi module coming here? If we only have one 'exported' artifact, we should name it (exclude the -impl)	1	-1
Share function header! Move macros inside!	2	-1
I still don't see any reference to, so how could it possibly be being used??I would have expected a line like	1	-1

Table 5.4: Sentistrength scores of some negative emotions labelled in our dataset

Chapter 6

Conclusions

Our goal was to solve the problem of sentiment/emotion analysis in software engineering datasets. Existing work on this was mostly based on lexical analysis. We presented a machine learning and affect control theory based framework for a better understanding of the sentiments and the emotions on the pull request comments from GitHub. We created a new dataset of pull request comments and got it annotated by three different people on Amazon Mechanical turk and myself. The dataset was labelled in to twelve **Interaction Process Analysis** labels and ten **emotions**. Since, we used concepts from *Affect Control Theory* we can also gather valuable insights into the behaviors of the people writing comments on GitHub. Our machine learning approach is better than the lexical analysis in terms of generalization and for sentences of longer length.

Based on four aggregated IPA categories we can use Table 3.4 to get an idea of the different emotions present in a pull request comment. We also presented a metric learning and deep learning pipeline, involving feedforward neural nets with *Information-Theoretic Metric Learning* and *Large Margin Nearest Neighbor* which resulted in a higher $F1$ on the aggregated dataset than when we tried to predict each IPA and emotion separately. ITML being a scalable and online algorithm, can be used as online preprocessing step with feed forward neural networks in the event that we have limited training data.

We presented a machine learning based approach to do sentiment and emotional analysis on pull request comments from GitHub. Based on our aggregated categories, we were able to get some insights into the pull request comments. Given this dataset, our approach achieved the best possible results.

6.1 Further work

Although we were able to achieve good F1 score, we wanted to perform more fine grained classification of the individual IPA categories and the individual emotions (rather than aggregate like we did for this project). We believe the reason for not being able to perform further classification on the IPA categories and the emotions is that the annotated dataset was noisy. Next steps would be try and get a much cleaner dataset for IPA and emotions and then repeat the experiment to get deeper insights into the behaviors of developers. A cleaner dataset can be obtained by repeating the experiment on Mechanical Turk, but this time users get an option to mark each label on a separate screen. We had the users mark all the labels on one screen. This might have led to some bias to some of the labels. Another method to obtain a cleaner dataset would be to have some known students who are more familiar with GitHub and programming. This is because there was no way for us to verify if the people on Mechanical Turk knew about GitHub or how well were they versed with programming.

Once we are able to predict all the IPA categories or the emotions with a high F1 score, we can use the sample behaviors from Table 2.4 and the emotion mapping from IPA described in Table 2.6 to get an idea of the different behaviors/emotions taking place at a pull request comment. The information about the behaviours and the emotions can later be used to understand the social dynamics of interactions between developers on a pull request. As mentioned in Chapter 1, *positive emotions help in increasing task productivity and job satisfaction*. Analyzing emotions at pull requests can help us assess how a developer is feeling and appropriate steps can be taken to make sure that negative emotions are kept to a minimum. Such steps can include recommending the best way to act in order to achieve a positive emotion. For example, if a person is being very *aggressive*, the best way for the other person to act would be to stay *calm* and *cautious*. Other more complex analyses can then be attempted using the social interaction analysis approaches as detailed in Bales (1950) [3].

References

[1] Scott Ambler. *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons, 2002.

[2] Suresh Balakrishnama and Aravind Ganapathiraju. Linear discriminant analysis-a brief tutorial. *Institute for Signal and information Processing*, 18, 1998.

[3] Robert F Bales. Interaction process analysis; a method for the study of small groups. 1950.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[5] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[6] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of computational science*, 2(1):1–8, 2011.

[7] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

[8] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, pages 2067–2075, 2015.

[9] Bram Cohen. The bittorrent protocol specification, version 11031, 2008.

[10] Corinna Cortes and Vladimir Vapnik. Support vector machine. *Machine learning*, 20 (3):273–297, 1995.

[11] Jason V Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216. ACM, 2007.

[12] Munmun De Choudhury and Scott Counts. Understanding affect in the workplace via social media. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pages 303–316, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1331-5. doi: 10.1145/2441776.2441812. URL http://doi.acm.org/10.1145/2441776.2441812.

[13] David L Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[14] Roy T Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[15] BL Fredickson. The role of positive emotions in positive psychology. *American psychologist*, 56(3):218–226, 2001.

[16] github.com. Github public api, 2017. URL https://developer.github.com/v3/. [Online; accessed November 2017, 2017].

[17] Jacob Goldberger, Geoffrey E Hinton, Sam T Roweis, and Ruslan R Salakhutdinov. Neighbourhood components analysis. In *Advances in neural information processing systems*, pages 513–520, 2005.

[18] Google. Bag of words model, 2017. URL https://google.ca/images. [Online; accessed April 27, 2017].

[19] Google. Applications of autoencoders in natural language processing, 2017. URL https://www.doc.ic.ac.uk/~js4416/163/website/nlp/. [Online; accessed April 27, 2017].

[20] Google. Vector representations of words, 2017. URL https://www.tensorflow.org/tutorials/word2vec. [Online; accessed April 27, 2017].

[21] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[22] Georgios Gousios and Diomidis Spinellis. Ghtorrent: Github's data from a firehose. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 12–21, Piscataway, NJ, USA, 2012. IEEE Press. ISBN 978-1-4673-1761-0. URL http://dl.acm.org/citation.cfm?id=2664446.2664449.

[23] Jamie Guillory, Jason Spiegel, Molly Drislane, Benjamin Weiss, Walter Donner, and Jeffrey Hancock. Upset now?: emotion contagion in distributed groups. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 745–748. ACM, 2011.

[24] Emitza Guzman, David Azócar, and Yang Li. Sentiment analysis of commit comments in github: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 352–355, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2863-0. doi: 10.1145/2597073.2597118. URL http://doi.acm.org/10.1145/2597073.2597118.

[25] Emma Haddi, Xiaohui Liu, and Yong Shi. The role of text pre-processing in sentiment analysis. *Procedia Computer Science*, 17:26–32, 2013.

[26] Jeffrey T Hancock, Kailyn Gee, Kevin Ciaccio, and Jennifer Mae-Hwah Lin. I'm sad you're sad: emotional contagion in cmc. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, pages 295–298. ACM, 2008.

[27] Trevor Hastie and Robert Tibshirani. Discriminant adaptive nearest neighbor classification. *IEEE transactions on pattern analysis and machine intelligence*, 18(6): 607–616, 1996.

[28] David R Heise. Modeling interactions in small groups. *Social Psychology Quarterly*, 76(1):52–72, 2013.

[29] Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 857–864, 2003.

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[31] Li-Ping Jing, Hou-Kuan Huang, and Hong-Bo Shi. Improved feature selection approach tfidf in text mining. In *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, volume 2, pages 944–946. IEEE, 2002.

[32] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 92–101, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2863-0. doi: 10.1145/2597073.2597074. URL http://doi.acm.org/10.1145/2597073.2597074.

[33] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[34] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE, 1995.

[35] Donald Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

[36] Joseph B Kruskal and Myron Wish. *Multidimensional scaling*, volume 11. Sage, 1978.

[37] James T Kwok and Ivor W Tsang. Learning with idealized kernels. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 400–407, 2003.

[38] Leslie Lamport. *LATEX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[39] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[40] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

[41] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[42] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

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

[44] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 262–271, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2863-0. doi: 10.1145/2597073.2597086. URL http://doi.acm.org/10.1145/2597073.2597086.

[45] W Gerrod Parrott. *Emotions in social psychology: Essential readings*. Psychology Press, 2001.

[46] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[47] Charles Piller. Everyone is a critic in cyberspace. *Los Angeles Times*, 3(12):A1, 1999.

[48] Pintrest. Parrott's Framework of emotions, year = 2017, note = [Online; accessed April 27, 2017], url = https://www.pinterest.ca/patsyjpayne/emotions-n/?lp=true.

[49] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: sentiment analysis of security discussions on github. In *Proceedings of the 11th working conference on mining software repositories*, pages 348–351. ACM, 2014.

[50] Robert Plutchik and Herman Van Praag. The measurement of suicidality, aggressivity and impulsivity. *Progress in Neuro-Psychopharmacology and Biological Psychiatry*, 13: S23–S34, 1989.

[51] Sebastian Raschka. LDA vs PCA, year = 2017, note = [Online; accessed April 28, 2017], url = http://sebastianraschka.com/Articles/.

[52] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[53] Sam Scott and Stan Matwin. Feature engineering for text classification. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 379–388, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL http://dl.acm.org/citation.cfm?id=645528.657484.

[54] sentistrength. Sentistrength, a lexical sentiment analysis tool, year = 2017, note = [Online; accessed April 27, 2017], url = http://sentistrength.wlv.ac.uk/.

[55] Noam Shental, Tomer Hertz, Daphna Weinshall, and Misha Pavel. Adjustment learning and relevant component analysis. In *European Conference on Computer Vision*, pages 776–790. Springer, 2002.

[56] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[57] Mike Thelwall, Kevan Buckley, and Georgios Paltoglou. Sentiment strength detection for the social web. *Journal of the Association for Information Science and Technology*, 63(1):163–173, 2012.

[58] Ivor W Tsang, Pak-Ming Cheung, and James T Kwok. Kernel relevant component analysis for distance metric learning. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pages 954–959. IEEE, 2005.

[59] Analytics Vidhya. PCA, year = 2017, note = [Online; accessed October 29, 2017], url = https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/.

[60] Yilun Wang. Understanding personality through social media.

[61] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480, 2006.

[62] Wikipedia, the free encyclopedia. Tf-idf, 2013. URL https://en.wikipedia.org/wiki/. [Online; accessed April 27, 2013].

[63] Peter Willett. The porter stemming algorithm: then and now. *Program*, 40(3):219–223, 2006.

[64] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[65] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems*, pages 521–528, 2003.

[66] Liu Yang and Rong Jin. Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2(2), 2006.

[67] Junping Zhang, Stan Z Li, and Jue Wang. Manifold learning and applications in recognition. In *Intelligent multimedia processing with soft computing*, pages 281–300. Springer, 2005.

[68] Zhenyue Zhang and Jing Wang. Mlle: Modified locally linear embedding using multiple weights. In *Advances in neural information processing systems*, pages 1593–1600, 2007.

[69] Zhenyue Zhang and Hongyuan Zha. Principal manifolds and nonlinear dimension reduction via local tangent space alignment. *SIAM Journal of Scientific Computing*, 26:313–338, 2002.

[70] Zhihua Zhang, James T Kwok, and Dit-Yan Yeung. Parametric distance metric learning with label information. In *IJCAI*, page 1450, 2003.

Appendix A

Data Collection and Amazon Mechanical Turk Study

Here we detail the methodology for data collection from Amazon Mechanical turk. We chose 834 random pull request ids from the database and collected a total of 3000 pull request comments from those pull request ids.

We filtered the comments using the following procedure

1. We removed all the portions of code from the comment
2. We only used sentences in English. All non english words were filtered out.

After the aforementioned preprocessing steps, the 3000 comments were put on Amazon Mechanical turk for annotation. Each comment was annotated by four different people. Three of them were selected from amazon mechanical turk, and I was the fourth person.

We received ethics clearance from University of Waterloo Research Ethics Committee ORE #22117. We had a selection process for people from mechanical turk. Initially, we only put up fifty pull request comments for annotation and opened the task to the general public. The top three best performing people were given an option to continue marking the rest of the 2500 comments. The top three people were selected on the basis on how close their annotations were to a set of fifty annotations marked separately by me and Chengyi Zhou (student ID 20521138, who was an URA in the Computational Health Informatics lab during Winter 2017.) All the top three performing people completed the task.

We now present the snapshots of the Amazon Mechanical study experiment.

Multi Label Classification of Github comments

Please note that further registration to this HIT has been closed. Currently we have the required number of people working on the HIT.

Thanks for agreeing to take part in the project. The project's aim is to understand how conversations take place on the website Github.com.

Pre Requisite knowledge required before doing the task

You should have at some basic programming background in any language. Ideally you should be aware what Github.com(https://github.com/) is. You should be aware what code segments are and should be able to recognize that a given line is a piece of code rather than a normal sentence.

Github is like a social network for programmers. For example on Facebook or twitter , people post their pictures and other people comment on them. On Github people post their segments of code , and other people comment on

- how good/bad the code is,
- If the code does not work , people suggest ways on how to make it work

Here the data set we are dealing is from such interactions. People commenting on other people's code. Consider this , if you send a FRIEND REQUEST to someone on Facebook , the other person would have to ACCEPT it. Github operates in a similar manner. Instead of friend requests, people send other people something called a PULL REQUEST (which is basically a couple of lines of code) . On Facebook the friend request is accepted, on Github the Pull Request gets MERGED (which means that other people have accepted the code you have written). In all the sentences in this project, one person sends a piece of code (a.k.a. pull request) in hopes of getting it merged (a.k.a accepted) . Other people comment on the piece of code explaining what needs to be done to improve the code in order to get it merged. We would want you to label the sentences in the following 2 categories and remove any unnecessary lines of code in the sentence :

Task 1:

First category consists of the following 12 subcategories . The words in the brackets in the bracket signifies type of behaviour portrayed in the sentence. You have to select at least 1 and a maximum of 3 sub categories for each sentence.

- Shows Solidarity (help, compliment, gratify)
- Shows tension release (josh, laugh with, cheer)
- Agrees (agree with, understand,accommodate)
- Gives Suggestion (encourage, cue, coach)
- Gives opinion (evaluate, analyze, entreat)
- Gives orientation (inform, educate, explain)
- Asks for orientation (quiz, question, ask about)
- Asks for opinion (consult, prompt, query)
- Asks for suggestion (entreat, ask, beseech)
- Disagrees (disagree with, ignore, hinder)
- Shows Tension (fear, cajole, evade)
- Shows Antagonism (argue with, deride, defy)

Task 2:

The second category is the kind of emotion displayed by the person writing the comment. Again there are 10 different categories. You have to select at least 1 and a maximum of 3 emotions for each sentence.

- Thanks
- Sorry
- Calm
- Nervous
- Careless
- Cautious
- Aggressive
- Defensive
- Happy

- Angry

Task 3:

Removing unnecessary lines of code. In the website, the section where the sentence is displayed, you are allowed to modify the sentence. If you come across any unnecessary lines of code (which can be identified as anything not written in english), you can simple use the backspace key to erase those. **A segment of code can be identified as a sequence of words that do not seem to be a valid english sentence. Such sequence of words should be erased by simply pressing the backspace key or by selecting that portion of text with a mouse and then pressing backspace key or delete key. Any line of the sentence that seem rather irrelevant to the classificaton of the sentence into the specified categories should be removed.** For example for the sentence

Added test to ensure single class is passed. Multiple classes gave false positive. e.g. < div id="id" class="class class2"> $("#id").hasClass("class class2") => false

The latter part of the sentence is all code, you should erase that so that the sentence becomes

Added test to ensure single class is passed. Multiple classes gave false positive.

Instructions on how to label a sentence.

You do not have to understand what exactly is going on in a sentence. Since , each comment can involve a lot of programming dependent terminology. The way to label the sentences is as follows : Looks for a sequence of words would describe what the person did : For example :

Example 1

fallback for wildcard since there is no support to remove all classes and another loop didn't seem to make sense... there is an issue with the wildcard (existing) where removeClass("") gives invalid argument error.*

The above sentence is a bit hard to label in the 2 categories. The way to proceed is to look for a sequence of words that describe what the person did or is trying to convey:

fallback for wildcard since there is no support to remove all classes and another loop didn't seem to make sense... ***there is an issue with the*** *wildcard (existing) where removeClass("*")* ***gives invalid argument error.***

The bold part of the sentence, signifies that the person is trying to convey that there is some issue which gives a error. Now based on this bolded part, you have to label in the 2 categories. In the first category you could choose : *Disagrees* , *Shows antagonism* For the emotion you could choose : *Aggressive*

Example 2

I think you might need to *prefix the string with ^ like above, because of some other processing we do later on the chain with error messages.*

The first category could be : *Gives opinion* The second category could be *Cautious* , *Defensive*

Example 3

I'd like to keep the *YourKit stuff* ***out of the*** *oae module. You can deliver a custom setenv.sh.erb template via the setenv_template param to oae::app::server If you add the YK java arguments to localconfig/templates/oae-setenv.sh.erb then the oae module*

The first category could be : *Gives orientation, Shows Antagnism,DisAgrees.* The second category could be *Agressive* , *Defensive*

Example 4

Yep, you are right, *the problem is that we don't know if more headers have been added. Another option is to create a 'refresh' method and call it from outside every time a new header is created/removed.* ***Makes sense for me.***

The first category could be : *Shows solidarity, Agrees, Gives suggestion* The second category could be *Thanks, Calm*

Example 5

good catch, thought i got everything. *On Thu, Aug 9, 2012 at 12:21 PM, Greg Sadowski wrote: > In GroupCommerce.SDK.V3.PublisherConsumerApp/Form1.Designer.cs: > > > @@ -812,6 +860,12 @@ private void InitializeComponent()*

Since the above comment has some unncessary code (and one not required line), you should simply remove it. So the sentence now becomes

good catch, thought i got everything.

The first category could be : *Shows solidarity, Agrees* The second category could be *Thanks*

Example 6

I agree with Brad; *either the *complete* experience, or an unadorned page-refreshing experience.* ***I think a partially-working*** *automatic filter would be confusing. On 9 Aug 2012, at 17:02, Bradley Wright <notifications@github.com> wrote: > In app/asse*

Since the above comment has some unncessary code (and one not required line), you should simply remove it. So the sentence now becomes

I agree with Brad; *either the *complete* experience, or an unadorned page-refreshing experience.* ***I think a partially-working*** *automatic filter would be confusing.*

The first category could be : *Agrees ,Gives Opinion ,Gives Orientation* The second category could be *Thanks , Calm*

Please note that this HIT does not collect any personal information. The Register link down below only requires a username and password just to keep track of the web sessions

Information and Consent letter

You are invited to participate in a research study conducted by Deepak Rishi, under the supervision of Prof. Jesse Hoey of the University of Waterloo, Canada. The objectives of the research study are is to classify comments from Github into certain categories. You should have some programming experience (in any language) to take part in this study.

People without programming experience will not produce the necessary results. If you do not have programming experience, please do not continue with this HIT.

If you decide to volunteer, you will be asked to complete a online survey that is completed anonymously. Participation in this study is voluntary. There are no known or anticipated risks from participating in this study. The online suvey is completed in 2 parts. The first 20 sentences are a screening test. Once you finish marking the first 20 sentences and the results are in accordance to what we expect, you will be given the HIT code and an option to continue on to the 3000 sentences. Please note that you would be given the HIT code, regardless of fact that you get the option to continue on to the next 3000 sentences. You will be paid 7 cents per sentence annotated, or $1.40 for completing all sentences in the screening test.

Once we have the required number of people to do the task, further registration to this HIT would be closed. The register tab on the website will no longer be visible. This will also be indicated at the top of the website. Please do not attempt to continue with this HIT at that point.

At any point you may logout and continue the work at a later time. If you have already passed the screening , you will not be required to mark the first 20 again. You would continue from where you left off.

It is important for you to know that any information that you provide will be confidential. All of the data will be summarized and no individual could be identified from these summarized results. Furthermore, the web site is programmed to collect responses alone and will not collect any information that could potentially identify you (such as machine identifiers). You will be paid 7 cents per sentence annotated. We expect that each sentence will take about 25-30 seconds to complete. A person with some programming experience can complete about 120 sentences in one hour. After you have annotated 50-60 sentences, your speed for annotation would increase and you would be able to to annotate a sentence in about 15-20 seconds. In addition, reading the instructions and becoming familiar with the tasks

for the first few sentences will take approximately 10 minutes. We hope that each participant will complete 3000 sentences. From the moment you start the task , you would have 120 hours to complete the task. You can take breaks in between. As mentioned earlier , at any point you can logout and when you login back, you will be continued form the point you left off . We hope that each participant who passes the screening test will complete 3000 sentences.

If you choose to withdraw from the study, you can click on the get HIT code button and submit the code on Mechanical turk. You would be given the payment as a bonus payment at the rate of 7 cents a sentence. If you wish to participate, please visit the website mentioned below. The data, with no personal identifiers, collected from this study will be maintained on a password-protected computer database in a restricted access area of the university. As well, the data will be electronically archived after completion of the study and maintained for two years and then erased. Further instructions on how to start the survey and examples sentences from Github are mentioned in the homepage of the link given below.

This study has been reviewed and received ethics clearance through a University of Waterloo Research Ethics Committee (ORE # 22117). If you have questions for the Committee contact the Chief Ethics Officer, Office of Research Ethics, at 1-519-888-4567 ext. 35217 or ore-ceo@uwaterloo.ca.

For all other questions about the study, please contact (deerishi@gmail.com) Further, if you would like to receive a copy of the results of this study, please contact either investigator. Thank you for considering participation in this study.

Consent to Participate By clicking on the register/login on the homepage of the survey link given below you agree of your own free will, to participate in this study. By providing consent, you are not waiving your legal rights or releasing the investigator(s) or involved institution(s) from their legal and professional responsibilities.

Check Labelled Sentences So Far

Logout
Withdraw from task

Comment 57

User : calrishi Numarked : 57

This is fantastic news!

Select at least 1 and upto max 3 labels that best describe what the user is trying to convey

- Shows Solidarity (help, compliment, gratify)
- Shows tension release (josh, laugh with, cheer)
- Agrees (agree with, understand,accommodate)
- Gives Suggestion (encourage, cue, coach)
- Gives opinion (evaluate, analyze, entreat)
- Gives orientation (inform, educate, explain)
- Asks for orientation (quiz, question, ask about)
- Asks for opinon (consult, prompt, query)
- Asks for suggestion (entreat, ask, beseech)
- Disagrees (disagree with, ignore, hinder)
- Shows Tension (fear, cajole, evade)
- Shows Antagonism (argue with, deride, defy)

Select max 3 emotions that the person might have had while writing the comment

- Thanks
- Sorry
- Calm
- Nervous
- Careless
- Cautious
- Aggressive
- Defensive
- Happy
- Angry

Previous comment | Submit and Next comment

Next comment without Submitting

Appendix B

Metric Learning Result Figures

This chapter illustrates the variation in the metrics of performance in the three classification tasks (over six categories) discussed in Chapter 3 over the different parameters of ITML and LMNN.

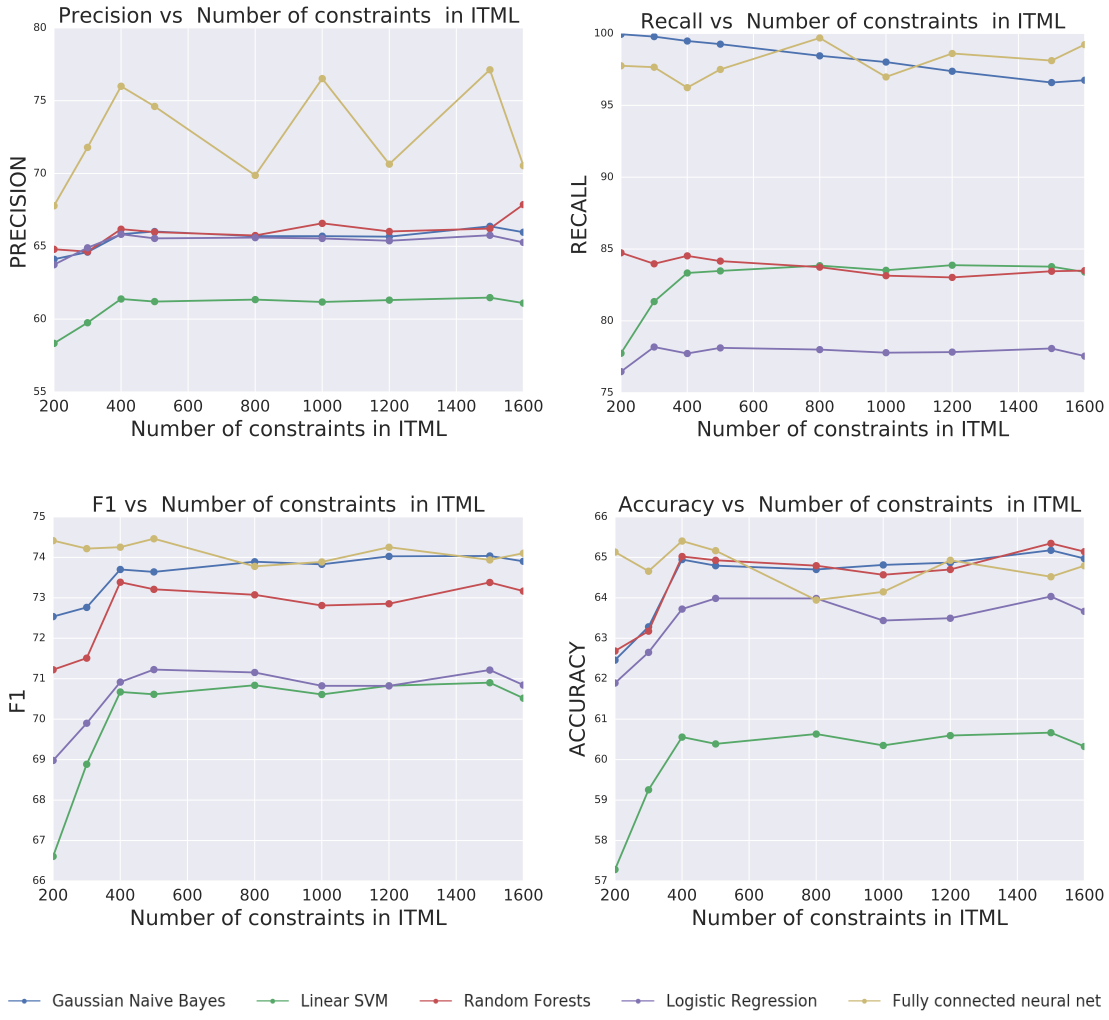


Figure B.1: Measures of performance vs number of ITML constraints for Agrees/Shows solidarity/Shows Tension release vs Disagrees/Shows Antagnism/Shows Tension

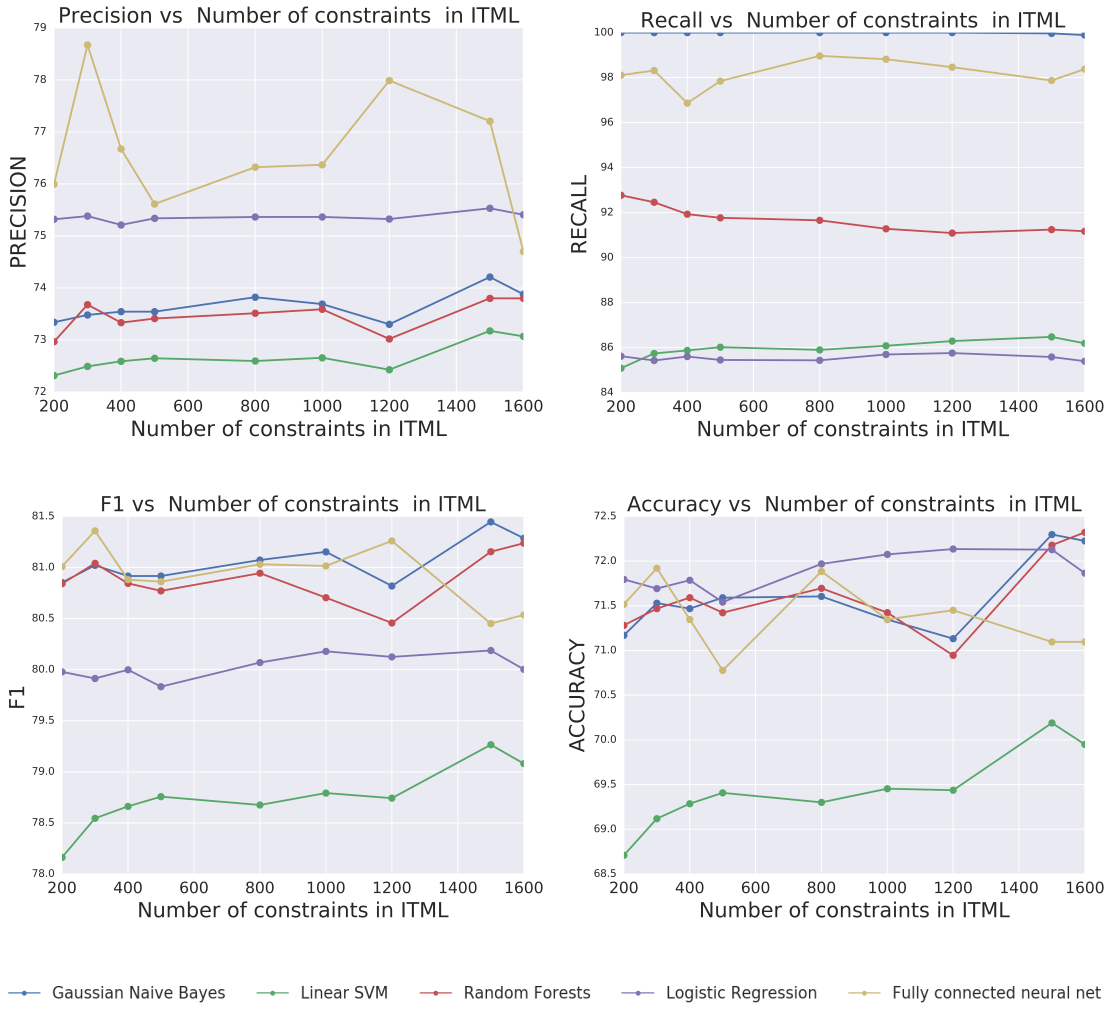


Figure B.2: Measures of performance vs number of ITML constraints for Gives opinion/Gives Suggestion/Gives orientation vs Asks for opinion/Asks for orientation/Asks for suggestion

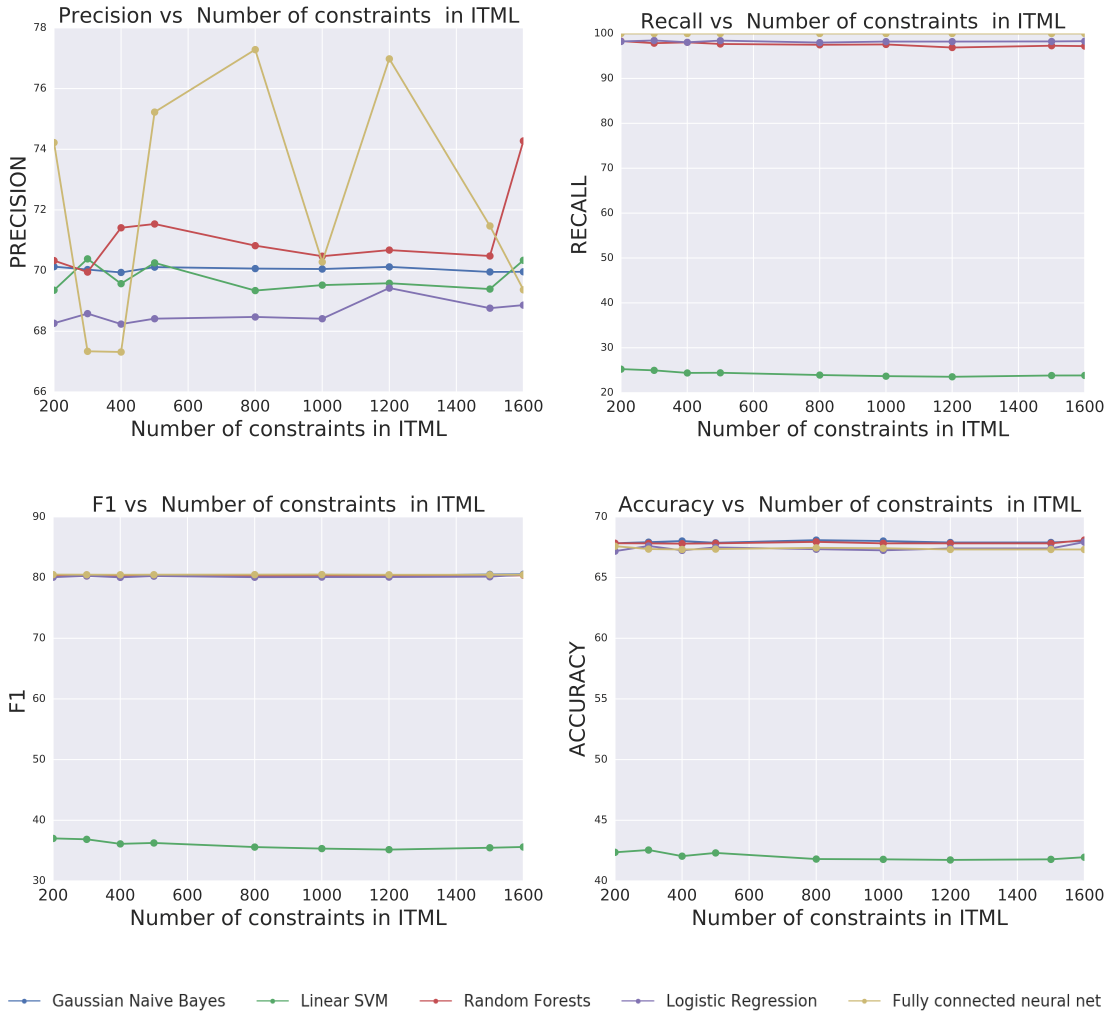


Figure B.3: Measures of performance vs number of ITML constraints for positive vs negative emotions

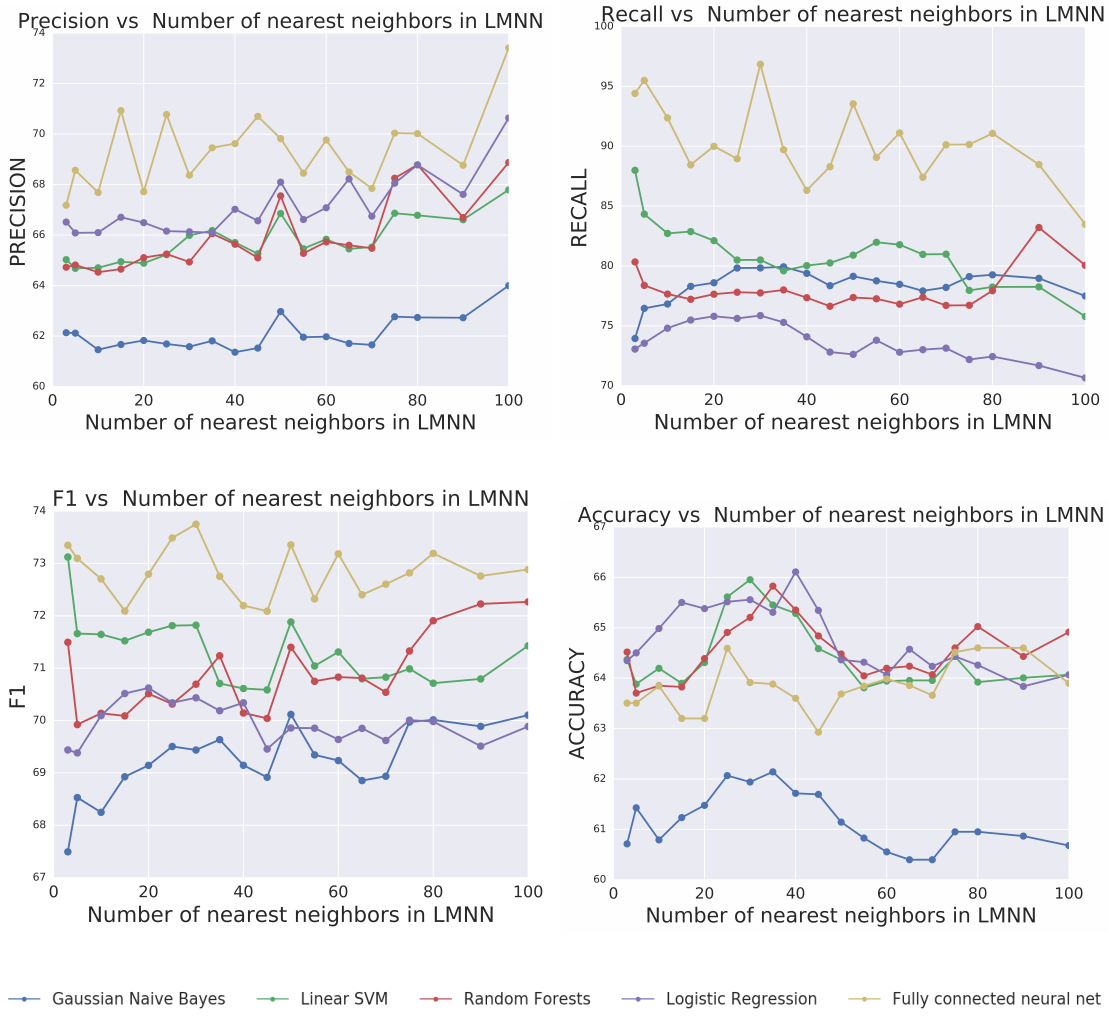


Figure B.4: Measures of performance vs number of LMNN constraints for Agrees/Shows solidarity/Shows Tension release vs Disagrees/Shows Antagnism/Shows Tension

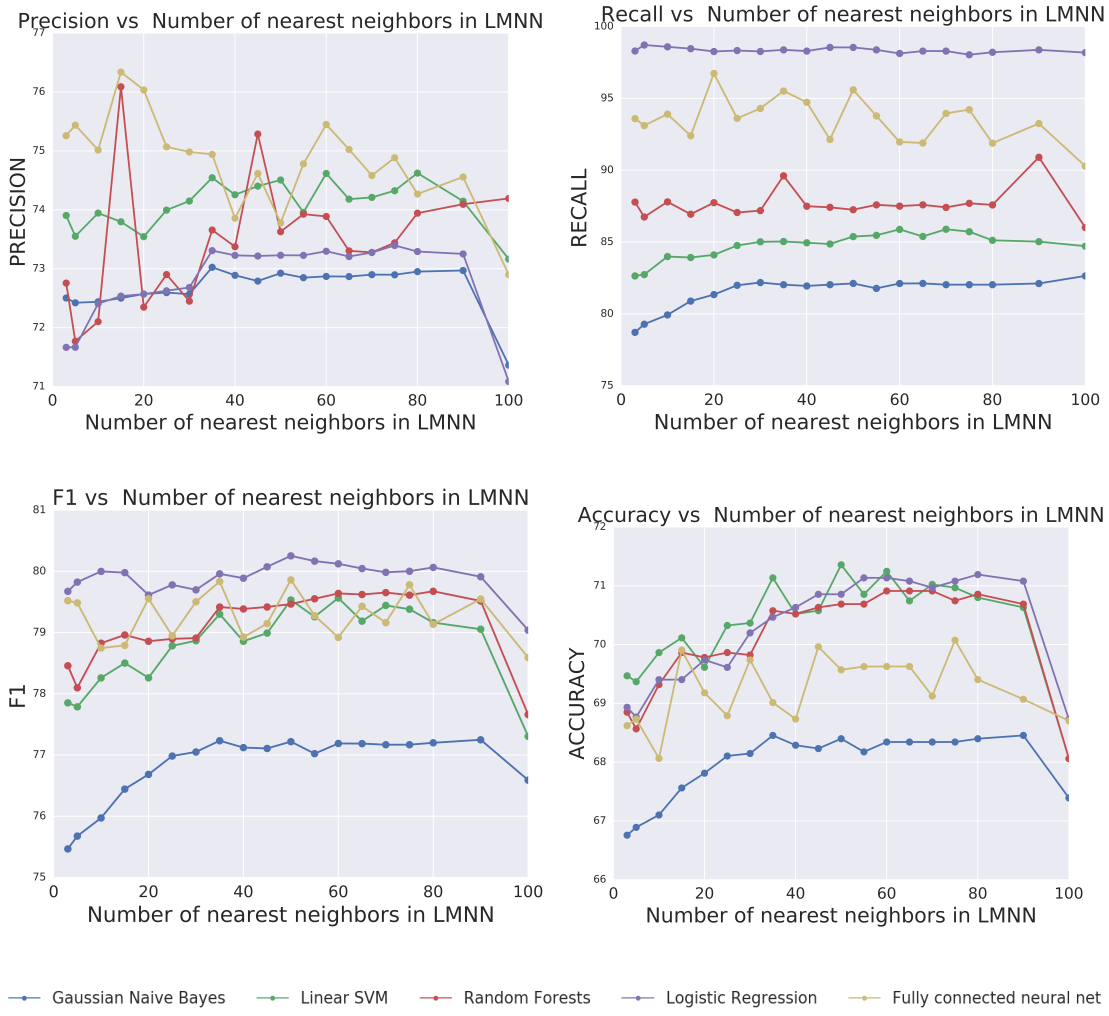


Figure B.5: Measures of performance vs number of LMNN constraints for Gives opinion/Gives Suggestion/Gives orientation vs Asks for opinion/Asks for orientation/Asks for suggestion

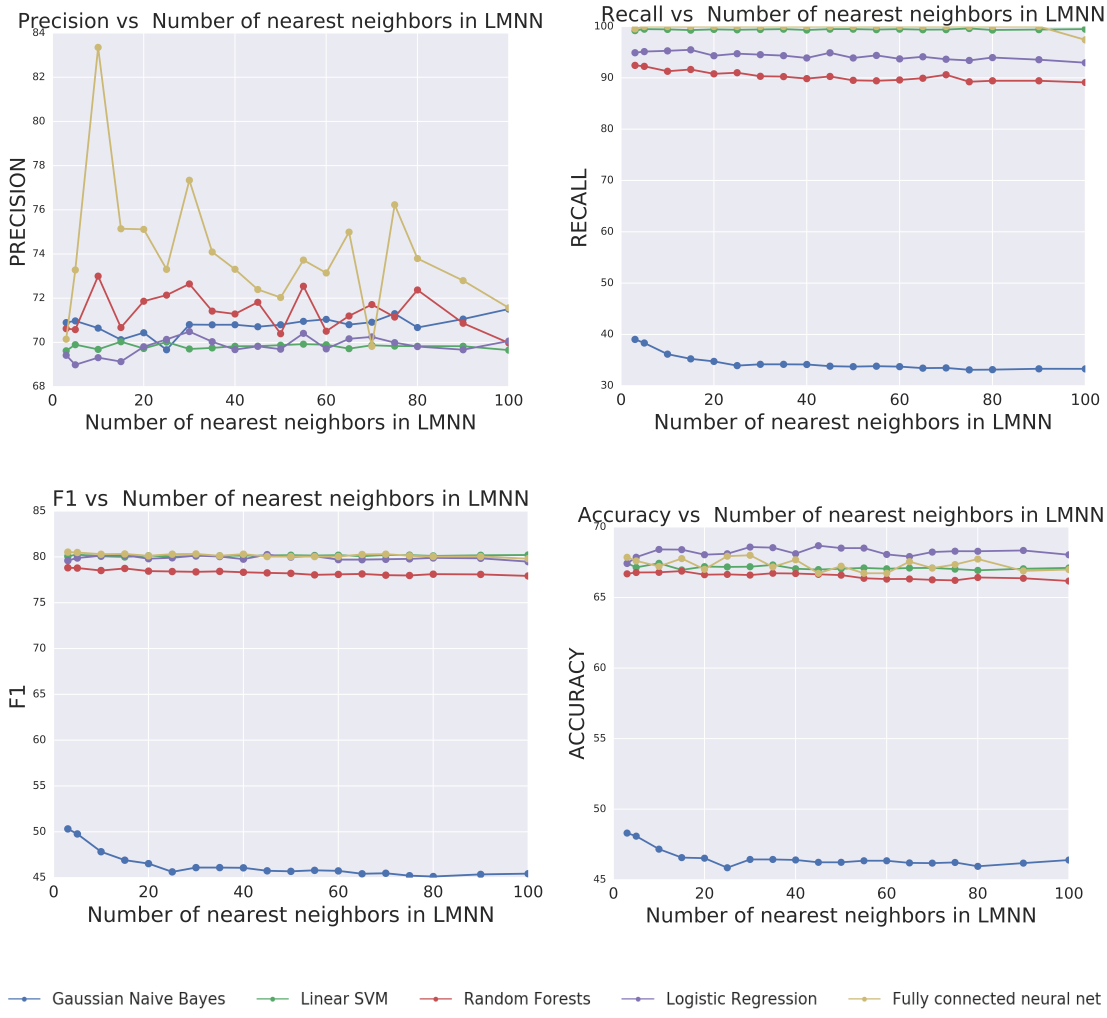


Figure B.6: Measures of performance vs number of LMNN constraints for positive vs negative emotions