Manuscript drafts

Gerbrich Ferdinands

1/14/2020

Introduction

Methods A convenience sample of 5 existing systematic reviews on varying topics was collected.

Results

Discussion

Introduction

Systematic Reviews (SR's) are booming - but they are a lot of work Various machine learning tools have been proposed to reduce workload in abstract screening.

- objectives - to demonstrate effectiveness of ml algorithm in reducing abstract classification for systematic reviews
- justification -
- background
- guidance to reader
- summary/conclusion

This study is about how machine learning algorithms can increase efficiency in systematic reviews. I will write about what SRs are, and how workload can be reduced.

Systematic reviews are top of the bill in research. As more and more papers are published and reproducibility crisis has emerged, science calls for more meta. It is important reflect on research by giving an overview of research areas which is typically done by a systematic review [...]. Performing a systematic review is a tedious and time-consuming task. To review a specific research area, one starts out with an initial search of thousands of academic papers. All these papers abstracts need to be screened to find an initial batch of possibly relevant papers. With now hopefully only a couple of hundred papers left, the researcher needs to read these papers full-text to arrive at a final selection of papers that are relevant for the final systematic review [this is prisma process?]. This whole processes costs this and this much time [shelmilt].

The stage of abstract screening where abstracts are systematically screened is where a lot is to be gained. This stage is the target of possible learning algorithms that can assist the reviewer in selecting the relevant papers. Together with the reviewer /human machine interaction. The algorithm aims to compute which papers in the pool need to be excluded and which need to be included, based on the reviewers decisions. It learns from the reviewers decisions and asks the reviewer to provide more labels, incrementally improving its class predictions.

The goal of the algorithm defined in the current study is to reduce to number of abstracts needed to screen (maybe not right term, bit biomedical). To be more specific, the algorithm aims to present the reader with the primary studies as soon as possible. This means that at some point you probably have seen all relevant abstracts and are only viewing excluded papers, which means you can stop reviewing much earlier (theoretically spoken). Also reviewing is now much more fun. As compared to when you have to review all abstracts and you perhaps see only one relevant abstract every other week/day.

So you might wonder, how does such an algorithm actually work? Active learning strategy, starting with a pool of unlabeled abstracts (U). The reviewer starts labeling some instances in U, creating L. The algorithm utilizes L to predict labels for all abstracts (classifier), by using a set of features from the papers called X, for example the text in the abstract (feature extraction method). Now, it made an initial classification. The algorithm now aims to improve its classification by which paper from U will be presented to the reviewer next. By labeling the next paper, the reviewer provides the algorithm with new information which the algorithm uses to update its prediction.

We approach asr as a classification problem: All papers obtained in the systematic search form a pool of instances x. All instances x need to be classified, e.g. we want to give them a label y. all x are now part of U,. Binary classification, either inclusion or exclusion. We want to classify based on some features from the instances x, feature vector $\mathbf{X}$.

By starting of with L, the algorithm utilizes characteristics of X to predict labels in U.

We want to classify the papers in Where the whole collection of papers in the systematic search form a pool of instances x, with unknown label y. <x,y>.

```
input: a pool of unlabeled abstracts $\mathcal{U}$
oracle labels a few initial papers, from $\mathcal{U}$, $\mathcal{L}$,

  M = train($\mathcal{L}$)
  query x \isin
```

- pool unlabeled abstracts $\mathcal{U}$
- labeled data set $\mathcal{L}$,
- instance x, label y
- utility measure $\phi_A(\cdot)$
- x_A^* best query instance according to $\phi_A(\cdot)$

Now there are a few technical details. Many different versions of such algorithms exist. Many of such algorithms have been described in the active learning literature and have been applied in the systematic reviewing process. Not exhaustive, but the algorithm can apply many different strategies to arrive at its predictions, which can be divided in following parameters: classifier, feature extraction strategy, balancing, query strategy.

Most often the SVM classifier is used, popular and very good results. Also lots of other configurations. However, other classifiers have not been tested a lot (polygon thing by cohe, naïve bayes and random forest by ...), but mostly SVM still. Also, most research in the medical sciences (well there are some exceptions of course [conversation between cohen and matwill]

In the current study, we aim at exploring the performance of several classifiers on reducing workload while maintaining performance in abstract screening process. This is done by performing simulations on existing systematic reviews. Performance is evaluated by how much time can be saved ... while maintaining accuracy.

Present research questions. RQ1 – which classifiers perform best? - RQ1a – does classifier performance vary over different research areas? (in what terms does it perform best)

RQ2 different hyperparameter optimizations? Classifiers come with hyperparameters. We have to make a choice on how to set these hyperparameters. What we do is create 3 sets, optimizing in three ways, aggregate results obtained.

The goal is to gain insight in classifiers other than the widely applied SVM, overall various research areas. So not only medical sciences.

To perform all these computations the research was carried out using the ASReview software by Utrecht University, which has a simulation mode that you can just input your labelled review file into and perform a simulation study with it. To be found on GitHub. It has many adjustable components/is very versatile.

Then if we still have time left we explore the effect of another feature extraction method, namely doc2vec. This might possibly increase performance as it performs better at grasping structure/hidden relations/hierarchy between words in the texts. So, it could be interesting in more 'fuzzy' research areas. A downside is that it takes more computing time.

Then if there's even more time we might want to compare a different balance strategy.

A SR can be divided into phases. Everything starts with a **systematic search**, leading to then citation screening is performed, then full-text screening [**PRISMA-PGroup2015**]

What must be the objective of our tool?: It is the tedious task citation screening part where loads of time can be saved.

models are designed in a 'realistic' way (you have some inclusions)

Selecting papers is a two-step process: abstract & fulltext screening

Binary classification problem. predict whether a paper in the pool is an inclusion or exclusion, based on labeled instances from $\mathcal{L}$ and use training data to understand how input variables are related to class.

We're building active learning model who takes X as an input an predicts class using labels from training set. Model improves by deciding asking more information from the reviewer (oracle), accounting for imbalance.

Assumptions

1) decisions of the original SR are **ground truth** (benchmark) (oracle)

The inclusion rate is ... data is imbalanced. what is the philosophy False negatives must be avoided ... The cost of a false negative outweighs the cost of a false positive. Note that we assume the oracle/original user to hold the truth. This is of course not always the case.

There are two classes in the data: exlusions and inclusions. The inclusions are clearly the minority class.

Datasets from the medical and social sciences, software engineering and public administration. Medical sciences SRs are viewed as more 'strict'/'structured' and social sciences more messy.

active learning for systematic reviews

corpus = all the text:

Active learning = increasing classification performance with every query. The query strategy determines the way unlabeled papers are queried to the researcher.

[**modAL2018**]

RQ1 - what are good classifiers RQ2 - what are good optimization strategies

Background

[**OMara-Eves2015**] literature review

[**Yu2018a**], [**Yu2019**] simulated 32 svm classifiers, on software engineering. A popular classifier is SVM. succes with HUTM (fastread), uncertainty, mix of weighting and agressive undersampling, In terms of Yu et al, we adopt .CT.

SVM - tf-idf on medical data, uncertainty sampling, agressive undersampling. [**Wallace2010**]

abstrackr

SVM + Weighting + uncertainty (bow) produced good methods [**Miwa2014**] Also include social sciences data besides medical data.

[**Cohen2006**] perceptron-based classifier (neural network)

SVM on legal documents (no balancing, certainty) [**Cormack2014**] in limitations section mentions that LR yields about same results, nb inferior results.

[**Kilicoglu2009**] - SVM, naive bayes, boosting and combinations. future work should optimize parameters. "Regarding the base classifiers used in identifying method- ologically rigorous studies, boosting consistently strikes the best balance between precision and recall, whereas naive Bayes in general performs well on recall

(demonstrating a tradeoff between recall and precision), as does polynomial SVM on precision. The AUC results are mixed, although boosting has a slight edge overall. These results demonstrate that different classifiers can be used to satisfy different information needs (SVM for specificity, naive Bayes for sensitivity, and boosting for balance between the two, for example)."

Our extensions is that we try different classifiers, on more datasets.

This study was approved by the Ethics Committee of the Faculty of Social and Behavioural Sciences of Utrecht University, filed as an amendement under study 20-104.

All simulations were run using through cartesius EINF-156

Methods

Goal: evaluate performance of different models of the ASReview tool. The screening process is simulated using ASReview, seeing if the original inclusions replicate. What would happen if the citation screening would have been performed using asreview? All datasets accompanying the systematic reviews are openly published.

Datasets

The algorithm will be tested on five systematic reviews from various research areas. test datasets serve as systematic search results, then perform active learning to detect inclusions.

Cohen et al. collected systematic review datasets from the medical sciences [**Cohen2006**]. All systematic reviews in this database are on drug efficacy. The *ace* dataset used in the current study comes from a systematic review on the efficacy of Angiotensin-converting enzyme (ACE) inhibitors.

a machine learning-based citation classification tool to reduce workload in systematic reviews of drug class efficacy. Using a perceptron classifier, WSS@95% = 56.61 in [**Cohen2006**]. (5x2 crossvalidation). Can we beat this? The data

The *software* dataset is retrieved from [**Yu2018a**], who collected datasets on literature reviews from the software engineering field. This dataset is on fault prediction in software engineering by [**Hall2012**].

The *nudging* dataset comes from a review from the behavioural public administration area. The SR includes studies on nudging healthcare professionals [**Nagtegaal2019**]. The data was stored on the Harvard Dataverse [**Nagtegaal2019a**].

A literature review from field of psychology, *ptsd*. The SR is on studies applying latent trajectory analyses on posttraumatic stress after exposure to trauma [**vandeSchoot2017**]. The corresponding data can be found on the Open Science Framework [. . .].

wilson, a dataset from the medical sciences [**Appenzeller-Herzog2020**]. TA review on effectiveness treatments of Wilson disease [**Appenzeller-Herzog2019**].

All datasets started with an initial pool of thousands of papers. A fraction of these papers where deemed relevant for the SR, with inclusion rates around 1-2 percent with one outlier of about 5 percent (Table 1).

The raw datafiles published with the SRs were preprocessed into a test dataset. The test datasets contain title information on all citations obtained in the search strategy. The instances in the data consist of a title, an abstract and were labeled to indicate which citations were included in the systematic review. Instances with missing abstracts were removed. Duplicate instances were removed. Preprocessing scripts can be found on the GitHub[1]

Table 1: Statistics on datasets from original systematic reviews.

Dataset	Original study			Test collection		
	Candidate studies	Final inclusions	Inclusion rate (%)	Candidate studies	Final inclusions	Inclusion rate (%)
ace	2544	41	1.61	2235	41	1.83
nudging	2006	100	4.99	1848	100	5.41
ptsd	6185	38	0.61	5031	38	0.76
software	8911	104	1.17	8896	104	1.17
wilson	3453	26	0.75	2334	24	1.03

[1] https://github.com/GerbrichFerdinands/asreview-thesis

Models

Five different active learning models were build. Every model m was used to perform an automated systematic review on SR dataset d. The models all apply a different classifier c with its own set of (hyper)parameters h.

Classifiers The classifier predicts the class of all papers, given the training dataset/labeled data set $\mathcal{L}$.

Given the labels and some features from all abstracts in the pool, the classifier To predict whether a paper should be an exclusion or an inclusion, different classifiers

Logistic Regression (L) -

Naive Bayes (B) - predicts the class of an instance given input features $\mathbf{X}$. Naive Bayes assumes all features are independent given the class value. This is obviously not the case but still the algorithm performs impressively [**Zhang2004**]. Especially at ... tasks.

Random Forests (R) is where a large number of decision trees are fit on bootstrapped samples of the original data. All trees cast a vote on the class, which are aggregated into a class prediciton for each input $\mathbf{X}$ [**Breiman2001**].

Support Vecor Machine (S) - finds a multidimensional hyperplane to separate classes. [**Tong2001**]

Dense Neural Network (N) -

Besides c, components of m are fea

To summarize, every model m consists of the following key components: classifier, feature extraction strategy, query strategy, balance strategy.

For example M_1 BTMD (naive bayes, tfidf, certainty sampling, double balance) STMD

Word representation To be able to predict whether a paper needs to be included or excluded (e.g. to predict class), the classifier needs some features from the papers.
As features we use the title and abstract from every paper - this is what is prescribed by prisma, (check!) The classifier cannot predict the paper class from the raw titles and abstracts as they are. Therefore, the content of the texts needs to be transformed into numerical representations called feature vectors.

A classical example is a 'bag of words' representation. For each each text, the number of occurrences of each word is stored. This leads to n features, where n is the number of distinct words in the texts [**scikit-learn**]. The bag-of-words method is simplistic and will highly value often occuring but otherwise meaningless words such as "and". Term-frequency Inverse Document Frequency (TF-IDF) [**Ramos2003**] circumvents this problem by adjusting a term frequency in a text with the inverse docuement frequency, the frequency of a given word in the entire corpus.

(T) (D)

The next abstract to be queried The model has various ways of deciding which instance should be queried next.

Rebalancing the training set To account for class imbalance in the data, the model can apply several strategies to rebalance the training set.

A reweighting strategy is applied where inclusions (the minority class) are weighted more heavily than the exclusions.

When no balancing is applied, the training data set = labeled data set $\mathcal{L}$

Starting point The initial training set consists of 5 inclusions and 5 exclusions, randomly sampled from the dataset. We use 5 inclusions and exclusions as we assume the researcher has some prior knowledge on this. The researcher has some prior knowledge about the pool, some papers ought to be included in the SR.

```
c = naivebayes
f = tfidf
q = max
n_prior_included = 5
n_prior_excluded = 5
```

Retraining We can choose to retrain the model after labeling n instances. - n_instances=10 (number of papers each query)

Simulations

To evaluate performance of the five models described above, the model is used to simulate five existing systematic reviews.

Optimizing hyperparameters

For every model*data combination, 3 sets of hyperparameters are generated to ... parallel Every model has its own hyperparameters. For every model, the hyperparameters are optimized three times, arriving at three versions of the model:

We now have 75 combinations. for every for every model (5), for every dataset (5) and for every set of optimized hyperparameters (3), a simulation study consisting trials is performed. From these $5 * 5 * 3 = 75$ simulation studies, performance of the different models is evaluated.

A simulation is of one model on one dataset. The simulation is repeated for 10 trials t.

Every simulation study consists of 10 trials, to account for the randomness of prior inclusions and exclusions. So every trial the prior inclusions and exclusions are randomly selected. Results are aggregated (?)

Statistical analysis

measures

comparison over conditions

The software

ASReview takes the following parameters/arguments:

	Configurations
Models	2-Layer Neural Network, Naive Bayes, Random Forest, Support Vector Machine, Logistic Regression
Query Strategies	Cluster Sampling, Maximum Sampling, Cluster * Maximum Sampling, Maximum * Uncertainty Sampling, Maximum * Random Sampling, Cluster * Uncertainty Sampling, Cluster * Random Sampling
Feature extraction strategies	Doc2Vec, TF-IDF, sbert, embeddingIdf

Use these inputs to predict relevance of papers.

Stage 1: hyperparameter optimization

Or, more specific:

Models	Feature extraction strategies
dense_nn	doc2vec
nb	tfidf
rf	tfidf
svm	doc2vec
lr	tfidf

Hyperparameters

Every model has its own set of hyperparameters:

Optimization

The hyperparameters are optimized on the 5 datasets in three different ways:

- 1 on 1: maximum performance

$$d = D$$

- 4 on 1: cross-validation

$$d \notin D$$

$$D = 1, 2, 3, 4$$

- 5 on 1: more data = more better?

$$d \in D$$

This results $(5 + 5 + 1) * 5$ sets of hyperparameters.

Performance metrics For each model, Several metrics are used to compare performance of different models over datasets,

Dataset	Naive Bayes	Random Forests	Support Vector Machine	Logistic Regression	Dense Neural Network
ptsd	?				
ace	?				
hall	?				
nagtegaal	?				
....	?				

The goal is twofold: we want to identify all relevant papers, as fast as we can. Tradeoff: identifying all relevant papers and reducing workload. A good metric to evaluate this is..

What is more important: recall or precision?

Recall more highly valued than precision.

What about class imbalance?

RRF Amount of relevant references found after having screened a certain percentage of the total number of abstracts.

Work saved over sampling (WSS) Indicates how much time can be saved, at a given level of recall. WSS is in terms of the percentage of abstracts that don't have to be screened by the researcher. Typically, WSS is measured at a recall of 0.95. Reasonable because..

$$\texttt{WSS} = \frac{TN + FN}{N} - (1 - recall)$$

Raoul

Utility?

F-measure

ROC/AUC Is performance related to some characteristic (n, inclusion rate, ...)

? How to compare outcomes of 3 different optimization strategies?

Results

Discussion

Appendix A - list of definitions

Feature Extraction Strategies

split_ta = overall hyperparameter

TF-IDF

hyperparameters

```
ngram_max: int
        Can use up to ngrams up to ngram_max. For example in the case of
        ngram_max=2, monograms and bigrams could be used.
```

Doc2Vec Predicts words from context. Aims at capturing the relations between word (man-woman, king-queen). [**Le2014**]. Using a neural network.

using Continuous Bag-of-Words (CBOW), Skip-Gram model, ... Word vector W and extra: document vector D, trained to predict words in the text.

From gensim [**Rehurek2010**].

```
    Arguments
    ---------
    vector_size: int
        Output size of the vector.
    epochs: int
        Number of epochs to train the doc2vec model.
    min_count: int
        Minimum number of occurences for a word in the corpus for it to
        be included in the model.
    workers: int
        Number of threads to train the model with.
    window: int
        Maximum distance over which word vectors influence each other.
    dm_concat: int
        Whether to concatenate word vectors or not.
        See paper for more detail.
    dm: int
        Model to use.
        0: Use distribute bag of words (DBOW).
        1: Use distributed memory (DM).
        2: Use both of the above with half the vector size and concatenate
        them.
    dbow_words: int
        Whether to train the word vectors using the skipgram metho
```

SBERT BERT-base model with mean-tokens pooling [**Reimers2019**]

embeddingIdf This model averages the weighted word vectors of all the words in the text, in order to get a single feature vector for each text. The weights are provided by the inverse document frequencies

Models

Naive Bayes Naive Bayes assumes all features are independent given the class value. [**Zhang2004**]

ASReview uses the `MultinomialNB` from the scikit-learn package [**scikit-learn**], that implements the naive Bayes algorithm for multinomially distributed data. `nb`

Hyperparameters

- alpha - accounts for features not present in learning samples and prevents zero probabilities in further computations.

Random Forests A number of decision trees are fit on bootstrapped samples of the original data, [**Breiman2001**] RandomForestClassifier from sklearn

Arguments ——— n_estimators: int Number of estimators. max_features: int Number of features in the model. class_weight: float Class weight of the inclusions. random_state: int, RandomState Set the random state of the RNG. """

Support Vector Machine Arguments ——— gamma: str Gamma parameter of the SVM model. class_weight: class_weight of the inclusions. C: C parameter of the SVM model. kernel: SVM kernel type. random_state: State of the RNG.

Logistic Regression

Dense Neural Network

Query Strategies

- Max - Choose the most likely samples to be included according to the model
- Uncertainty - choose the most uncertain samples according to the model (i.e. closest to 0.5 probability) [**Lewis1994**]
- Random - randomly selects abstracts with no regard to model assigned probabilities.
- Cluster - Use clustering after feature extraction on the dataset. Then the highest probabilities within random clusters are sampled

The following combinations are simulated:

- cluster
- max
- cluster * random
- cluster * uncertainty
- max * cluster
- max * random
- max * uncertainty

Balance Strategies

amount of training data

- n_instances = number of papers queried each query
- n_queries = number of queries
- n_prior_included: 5
- n_prior_excluded:

Combinations

This leads to 119 combinations of configurations.

- Naive bayes only goes with tfidf feature extraction.
- For the feature extraction strategies we will focus on doc2vec and tfidf. (but will compute all 4)
- This leads to 3 * 7 * 4 * 3 + 1 * 7 * 1 * 3 = 273 combinations.

See appendix A for a table containing all 273 combinations.

Cross-validation

Should give an accurate estimate of maximum performance / future systematic reviews to be performed.

Appendix B - combinations

Model	Query Strategy	Feature extraction strategy
dense_nn	cluster	doc2vec
dense_nn	max	doc2vec
dense_nn	max * cluster	doc2vec
dense_nn	max * uncertainty	doc2vec
dense_nn	max * random	doc2vec
dense_nn	cluster * uncertainty	doc2vec
dense_nn	cluster * random	doc2vec
dense_nn	cluster	tfidf
dense_nn	max	tfidf
dense_nn	max * cluster	tfidf
dense_nn	max * uncertainty	tfidf
dense_nn	max * random	tfidf
dense_nn	cluster * uncertainty	tfidf
dense_nn	cluster * random	tfidf
dense_nn	cluster	sbert
dense_nn	max	sbert
dense_nn	max * cluster	sbert
dense_nn	max * uncertainty	sbert
dense_nn	max * random	sbert
dense_nn	cluster * uncertainty	sbert

(continued)

Model	Query Strategy	Feature extraction strategy
dense_nn	cluster * random	sbert
dense_nn	cluster	embeddingIdf
dense_nn	max	embeddingIdf
dense_nn	max * cluster	embeddingIdf
dense_nn	max * uncertainty	embeddingIdf
dense_nn	max * random	embeddingIdf
dense_nn	cluster * uncertainty	embeddingIdf
dense_nn	cluster * random	embeddingIdf
nb	cluster	tfidf
nb	max	tfidf
nb	max * cluster	tfidf
nb	max * uncertainty	tfidf
nb	max * random	tfidf
nb	cluster * uncertainty	tfidf
nb	cluster * random	tfidf
rf	cluster	doc2vec
rf	max	doc2vec
rf	max * cluster	doc2vec
rf	max * uncertainty	doc2vec
rf	max * random	doc2vec
rf	cluster * uncertainty	doc2vec
rf	cluster * random	doc2vec
rf	cluster	tfidf
rf	max	tfidf
rf	max * cluster	tfidf
rf	max * uncertainty	tfidf
rf	max * random	tfidf
rf	cluster * uncertainty	tfidf
rf	cluster * random	tfidf
rf	cluster	sbert
rf	max	sbert
rf	max * cluster	sbert
rf	max * uncertainty	sbert
rf	max * random	sbert
rf	cluster * uncertainty	sbert
rf	cluster * random	sbert
rf	cluster	embeddingIdf
rf	max	embeddingIdf
rf	max * cluster	embeddingIdf
rf	max * uncertainty	embeddingIdf
rf	max * random	embeddingIdf
rf	cluster * uncertainty	embeddingIdf
rf	cluster * random	embeddingIdf
svm	cluster	doc2vec
svm	max	doc2vec
svm	max * cluster	doc2vec

(continued)

Model	Query Strategy	Feature extraction strategy
svm	max * uncertainty	doc2vec
svm	max * random	doc2vec
svm	cluster * uncertainty	doc2vec
svm	cluster * random	doc2vec
svm	cluster	tfidf
svm	max	tfidf
svm	max * cluster	tfidf
svm	max * uncertainty	tfidf
svm	max * random	tfidf
svm	cluster * uncertainty	tfidf
svm	cluster * random	tfidf
svm	cluster	sbert
svm	max	sbert
svm	max * cluster	sbert
svm	max * uncertainty	sbert
svm	max * random	sbert
svm	cluster * uncertainty	sbert
svm	cluster * random	sbert
svm	cluster	embeddingIdf
svm	max	embeddingIdf
svm	max * cluster	embeddingIdf
svm	max * uncertainty	embeddingIdf
svm	max * random	embeddingIdf
svm	cluster * uncertainty	embeddingIdf
svm	cluster * random	embeddingIdf
lr	cluster	doc2vec
lr	max	doc2vec
lr	max * cluster	doc2vec
lr	max * uncertainty	doc2vec
lr	max * random	doc2vec
lr	cluster * uncertainty	doc2vec
lr	cluster * random	doc2vec
lr	cluster	tfidf
lr	max	tfidf
lr	max * cluster	tfidf
lr	max * uncertainty	tfidf
lr	max * random	tfidf
lr	cluster * uncertainty	tfidf
lr	cluster * random	tfidf
lr	cluster	sbert
lr	max	sbert
lr	max * cluster	sbert
lr	max * uncertainty	sbert
lr	max * random	sbert
lr	cluster * uncertainty	sbert
lr	cluster * random	sbert
lr	cluster	embeddingIdf

(continued)

	Model	Query Strategy	Feature extraction strategy
lr		max	embeddingIdf
lr		max * cluster	embeddingIdf
lr		max * uncertainty	embeddingIdf
lr		max * random	embeddingIdf
lr		cluster * uncertainty	embeddingIdf
lr		cluster * random	embeddingIdf

Appendix C - supercomputer Cartesius

500,000 SBU

Running on Cartesius is charged in System Billing Units (SBUs), and charging is based on the wall clock time of a job. On fat and thin nodes, an SBU is equal to using 1 core for 1 hour (a core hour), or 1 core for 20 minutes on a GPU node. Since compute nodes are allocated exclusively to a single job at a time, you will be charged for all cores on that node - even if you are using less.