Answering Geosciences Research Questions at a Global Scale via a Hybrid Machine-Human Learning Approach: A Case Study of the Link between Climate and Volcanism

Seongjin Park, Barbara Carrapa, *University of Arizona, Tucson, Arizona 85721, USA;* ***Mihai N. Ducea,*** *University of Arizona, Tucson, Arizona 85721, USA, and Faculty of Geology and Geophysics, University of Bucharest, 010041, Bucharest, Romania;* ***Mihai Surdeanu, Robert Hayes, Dan Collins,*** *University of Arizona, Tucson, Arizona 85721, USA*

Supplemental Document 1

Temporal Expression Normalization

To convert mentions of temporal expressions (i.e., names of geological eras or epochs) to temporal intervals, we created a spreadsheet that contains the relations between date intervals and these temporal expressions. The file contains the name of the geological time era (e.g., *Jurassic*) and the time period (e.g., from 201.3 million years ago to 145 million years ago). The following table shows a subset of this spreadsheet:

Era/Epoch	From	To
Eoarchean	4,000,000,000	3,400,000,000
Paleoarchean	3,400,000,000	3,200,000,000
…	…	…
Jurassic	201,300,000	145,000,000
…	…	…

Table 1. A subset of the spreadsheet file that map names of geological eras/epochs to actual time intervals.

Algorithm 3: Sample rules for temporal expressions

```
name: time-period-1
    priority: 1
    label: TempExpr
    type: token
    pattern: |
        /Z?(Tertiary|Maastrichtian|Danian|
        Guadalupian|Triassic|Cenomanian|
        Cretaceous|Paleogene|Palaeocene|
        Pliocene|Pleistocene|Holocene|
        Zanclean|Cambrian|Paleozoic|
        Palaeozoic|Ordovician|Neogene|
        Phanerozoic|Silurian|Devonian|
        Carboniferous|Permian|Neoproterozoic|
        Mesozoic|Quaternary|Precambrian|
        Jurassic)/

name: time-period-2
    priority: 1
    label: TempExpr-Ago
    type: token
    pattern: |
    entity=/^NUM/+
      /(ma|myr|mya|ka|Ma|Myr|Mya|Ka|m.y.r)/
```

We extracted temporal expressions from text using two Odin rules, listed in Algorithm 3. The first rule (*time-period-1*) captures names of known geological epochs and eras. Note that, since the publications mined were automatically converted from PDF files to text files using Science-Parser[1], the result text files often had spelling mistakes. This rule captures the most common ones. The second rule (*time-period-2*) captures numeric temporal expressions such as *500 mya*, using common temporal abbreviations in geoscience papers.

[1] https://github.com/allenai/science-parse

Algorithm 4: Normalize the temporal expression and calculate the frequency

```
counter = count frequency of temporal
  expressions
for sentence in documents do
    TempExpr = [list of TempExpr in the
      sentence]
    Ago = [list of TempExpr-Ago in the
      sentence]
    for expr in TempExpr do
        counter[expr] += 1
    for expr in Ago do
        if expr starts with "M or m" then
            actualTime = NUM × 1000000
            timeExpr = find era using
              actualTime
            counter[timeExpr] += 1
        if expr starts with "K or k" then
            actualTime = NUM × 1000
            timeExpr = find era using
              actualTime
            counter[timeExpr] += 1
```

After capturing temporal expressions using the two rules summarized above, we used an additional script to convert and normalize the actual times to the corresponding geological times. The process is listed in Algorithm 4. For example, when one sentence contained a phrase *150 million years ago* or *150 m.y.r*, the script first converts the temporal expression to the time (in years) *150,000,000*, and then normalizes it to *Jurassic* using the spreadsheet listed in Table 1. After that, we counted the occurrence of geological eras/epochs in the document for later use, in the visualization. The following output shows an example of the statistics acquired from one paper, where lines 3 – 4 show the frequency of geological eras that occurred in the target paper.

```
1 "synthetic_data_on" : {
2 "time" : {
3       "Neogene" : 3,
4       "Pliocene" : 2    }
```

Answering Geosciences Research Questions at a Global Scale via a Hybrid Machine-Human Learning Approach: A Case Study of the Link between Climate and Volcanism

Seongjin Park, Barbara Carrapa, *University of Arizona, Tucson, Arizona 85721, USA;* ***Mihai N. Ducea,*** *University of Arizona, Tucson, Arizona 85721, USA, and Faculty of Geology and Geophysics, University of Bucharest, 010041, Bucharest, Romania;* ***Mihai Surdeanu, Robert Hayes, Dan Collins,*** *University of Arizona, Tucson, Arizona 85721, USA*

Supplemental Document 2

Spatial expression normalization

The second critical component necessary for the contextualization of geoscience results (in addition of the recognition of temporal expressions) handles the identification and normalization of location expressions. Similar to the recognition of temporal expressions, there are domain-specific spatial expressions that are not captured by existing Named Entity Recognition (NER) tools (e.g., Stanford CoreNLP). Further, some of these expressions (i.e., all IODP sites) do not contain direct information about the actual locations that they indicate. Thus, we wrote scripts to extract spatial expressions, disambiguate geoscience-specific spatial expressions (e.g., *IODP Site U1360*), and normalize those expressions. In this section, we will provide the algorithms used for site identification and normalization.

Recognition of location expressions

First, we applied the named entity recognizer in Stanford CoreNLP to check how many spatial expressions it recognizes. CoreNLP captures most of the well-known locations, such as *Bering Sea* or *Aleutian Islands*, but it does not recognize geoscience-specific locations (e.g., *IODP Site U1360* or *Deccan Traps*). To quantify these errors, we analyzed the annotation results from 100 sample documents using CoreNLP.

Algorithm 5: Named Entity Recognition using Stanford CoreNLP

```
for sentence in document do
    words = [word in sentence]
    for word in words do
        entity = Recognizer(word)
        if entity == "Location" then
            return entity
```

For this analysis, we used Algorithm 5 to deploy Stanford's CoreNLP to recognize named entities in a given sequence of words. In particular, the document was tokenized into sentences, and then, each sentence was split into words using the word-tokenizer in the CoreNLP package. Next, the recognizer processes each sentence, and returns named entity categories (*Location, Person, Organization, Number, Date, Miscellaneous*) when the input word is (part of) a named entity, or *O* otherwise.

Our analysis indicated that CoreNLP does recognize: (1) specific geological locations (e.g., DSDP Site,

IODP Site), (2) Traps[1], and (3) other specific locations that do not usually appear in general, open-domain texts. In addition, since the data were text files converted from PDF files, there were some misspelled words which made them unrecognizable.

To compensate for these limitations, we wrote a series of custom Odin rules to capture the above geological locations that are missed by this general-purpose tool. These rules are listed in Algorithm 6.

Algorithm 6: Rules for geological sites

```
name: geo-site-Site
    priority: 1
    label: SpatialExpr
    type: token
    pattern: |
        /DSDP/ /Site/ /U?[0-9]+[A-Z]?/
        |
        /IODP/ /Site/ /U?[0-9]+[A-Z]?/
        |
        /Site/ /U?[0-9]+[A-Z]?/

name: geo-site-Name
    priority: 1
    label: SpatialExpr
    type: token
    pattern: |
        /(Deccan|ParanaEtendeka|Karoo|Siberian)
          (Traps)?/
        |
        /(?i)flood/ /(?i)basalts?/
        |
        /Stevns/ /Klint/
        |
        /Tethyan/
```

Disambiguation of location names

As a result of the previous step, our location recognizer identifies both generic locations and locations specific to geoscience discourse. While the former can be disambiguated using existing resources, the latter cannot. For example, there is no resource to indicate the actual location for *IODP Site U1360*. To remedy this limitation, we implemented a data-driven algorithm that infers the actual location of those recognized terms. Our algorithm disambiguates these locations based on their collocation with other, known location names in the same document. In particular, we calculate the frequency of co-occurrence between a geological location (e.g., *IODP Site U1360*) and an actual location (e.g., *South Atlantic*).

[1] Here, *Trap* means a structural trap, which is a type of geological trap that forms as a result of changes in the structure of the subsurface, due to tectonic, diapiric, gravitational and compactional processes.

Then, we extract the distance between the two names based as the number of words between the names. Each geological location is disambiguated to the location with each it co-occurs the most in a collection of geoscience publications. In case of ties, we used distance information for disambiguation, i.e., we chose the actual location that tends to be closest in text. This algorithm is summarized in Algorithm 7. Table 2 shows some sample output for this disambiguation algorithm.

Algorithm 7: Get co-occurrence frequency and distance between geological location and actual location

```
site = HashMap for disambiguation
site_result = Hashmap for the result
for sentence in document do
    if (geo-site-Site or geo-site-Name) in
     sentence then
        entity = geo-site-Site or
          geo-site-Name
        site_idx = index of entity
        locations = [actual location in
          sentence]
        loc_idx = [indices of actual location
          - index of entity] for location in
          locations do
            site_result[entity][location]["freq"]
              += 1
            site_result[entity][location]["dist"]
              += loc_idx of location
    else
        pass;
for entity in site_result do
    site[entity] = location which has the
      highest frequency
```

Site	Location
Site 397	Africa
IODP Site U1341	Bering Sea
DSDP Site 216	Kerguelen
…	…

Table 2. Example results from the site inference component. The first column lists the unidentified sites; the second lists the most frequent co-occurring location.

The next step for the site identification is location normalization. Since there are multiple ways to describe the same location (e.g., *China* vs. *People's Republic of China*, or *Seoul* and *the capital city of South Korea*), the locations extracted from papers must be normalized. We used an external natural

language processing tool, *geonorm*[2], for this purpose.

Algorithm 8: Extracting normalized entities and recalculating frequencies

```
geonorm = location normalizer
counter = frequency of locations
site = dictionary from site disambiguation
for sentence in document do
    for word in sentence do
        if word.entity == Location then
            norm_loc = geonorm(word)
            counter[norm_loc] += 1
        if word is in site then
            convert_site = site[word]
            norm_loc =
              geonorm(convert_site)
            counter[norm_loc] += 1
```

Lastly, Algorithm 8 summarizes our process to calculate the frequency of location expressions in a given document. If a given word was recognized as *Location* with CoreNLP, then we fed the recognized word into the location normalizer, and added one to the frequency of the normalized location. When the given word was in the result of site inference, then we converted the recognized word into the actual location using the result from site disambiguation, and fed the converted word into the location normalizer. We compute the frequencies of all normalized locations. Figure 1 shows an example output of this process for one paper.


```
"synthetic_data_on" : {
    "location" : {
      "Atlantic County" : 1,
      "Republic of France" : 5,
      "Aquitaine Basin" : 1,
      "Kingdom of Morocco" : 4,
      "Bretagne" : 2,
      "Portuguese Republic" : 1,
      "Mediterranean" : 1,
      "Kingdom of Spain" : 1,
      "Montenay" : 1,
      "Cahuzac" : 1
    }
```


Figure 1. The result of the site normalization for one sample publication.

[2] https://github.com/clulab/geonorm/

Answering Geosciences Research Questions at a Global Scale via a Hybrid Machine-Human Learning Approach: A Case Study of the Link between Climate and Volcanism

Seongjin Park, Barbara Carrapa, *University of Arizona, Tucson, Arizona 85721, USA;* ***Mihai N. Ducea,*** *University of Arizona, Tucson, Arizona 85721, USA, and Faculty of Geology and Geophysics, University of Bucharest, 010041, Bucharest, Romania;* ***Mihai Surdeanu, Robert Hayes, Dan Collins,*** *University of Arizona, Tucson, Arizona 85721, USA*

Supplemental Document 3

Document classification

To determine whether a given geoscience paper supports (or not) the hypothesis investigated, i.e., that volcanism affects climate change, we built multiple document classifiers to automatically label a collection of papers with this information. To have the ability to investigate the details of the model such as the contribution of features to a prediction, we used two classifiers that provide this functionality: a linear support vector machines (SVM) classifier, and a Naïve-Bayes SVM (NB-SVM), using unigram and bigram features for both. In this section, we describe how the training documents were annotated, and how we trained and tested the two different SVM classifiers.

Paper Annotations

To have training and test data to build the proposed classifiers, 200 papers out of the 1,164 downloaded papers were presented to annotators, and they annotated whether the given paper supports or negates the hypothesis that volcanism impacts climate change, or are unrelated to the hypothesis. Two of the authors served as annotators. From each paper to be annotated we automatically extracted the title, abstract, introduction, and conclusion[1]. We used the crowd-sourcing platform FindingFive[2] to collect annotations. As a result, there were 400 responses (200 papers × 2 annotators), from which we constructed separate training and test partitions through cross-validation.

During the annotation, we allowed the annotators to choose more than one label per paper to encode more complex discourse. For example, the same paper could be annotated with SUPPORT and NEGATE labels, when a part of the given text supports the investigated hypothesis, but another negates it. However, this ambiguity tends to confuse machine learning methods, so we simplified multi-label annotations into a single label as follows:

1. We prioritized SUPPORT and NEGATE labels over UNRELATED. That is, when the annotator chose SUPPORT and UNRELATED, then the document would be labeled as SUPPORT. When the annotator chose NEGATE and UNRELATED, then the document would be labeled as NEGATE.
2. When SUPPORT and NEGATE were chosen at the same time (i.e., when the part of the given paper supports the idea and the other part does not), both labels would be kept as joint label NEGATE&SUPPORT.

---

[1] Since the papers were originally PDF files and converted to text files, some of the papers did not have correct section headings, or even any section heading in some situations. When the converted file did not have proper section headings, we extracted the first 300 words from the content to be presented to the annotators.

[2] https://www.findingfive.com

3. When the annotator chose all possible labels (SUPPORT, NEGATE, and UNRELATED), UNRELATED is ignored, and the two remaining labels are merged into NEGATE&SUPPORT.

As a result, the responses from the annotators were normalized into four labels: SUPPORT, NEGATE, NEGATE&SUPPORT, and UNRELATED. The distribution of 400 labels are as described in Table 1.

Label	Number
Negation	2
Negation & Support	6
Support	85
Unrelated	307

Table 1. The number of labels for 400 annotations

Evaluation Measures

To evaluate the classifiers discussed below, we used three evaluation measures commonly used to evaluate the performance of text classification algorithms:

- *Precision*: measures the proportion of predicted positives that are truly positive for a *given class*. For example, precision for the Support class computes the percentage of correct predictions among the predictions labeled by the machine as Support. *Micro precision* generalizes this formula to *all classes* in the dataset. That is, the formula for micro precision is:
    - Micro P = TP / (TP + FP), where TP indicates the number of true positives (i.e., correct predictions) for each class, and FP indicates the number of false positives (i.e., incorrect predictions) for each class.
- *Recall*: measures the proportion of actual positives in the test partition that are correctly predicted as true for a given *class*. For example, recall for the Support class computes the percentage of correct predictions among instances of the Support class in the evaluation data. *Micro recall* generalizes this formula to *all classes* in the dataset. More formally, the formula for micro recall is:
    - Micro R = TP / (TP + FN), where FN indicates the number of false negatives (i.e., labels in the evaluation data that are missed in the prediction) for each class.
- *F1*: harmonic mean of precision and recall for each *class*. *Micro F1* is the harmonic of micro precision and micro recall:
    - Micro F1 = 2PR(P + R), where P and R stand for micro precision and micro recall, respectively.

Linear SVM Classifier

With the annotated data, we created a linear SVM classifier using the *scikit-learn*[3] package in the Python programming language. First, we extracted unigram and bigram features (e.g., from the sentence "*The dog chased the cat*", the unigram features are the individual words in the sentence, [*the, dog, chased, cat*], and the bigram features are consecutive sequences of two words, i.e., [*start-the, the-dog, dog-chased, chased-the, the-cat, cat-end*]). After extracting features, training and test data were converted to feature matrices, which contains the frequency of each feature (unigram and bigram) in the given document.

Table 1 shows an example of such a feature matrix. The first column shows the generated labels (e.g., UNREL. (UNRELATED) and SUP. (SUPPORT)), and the other columns show the frequency of each feature (e.g., geology (*unigram*) and *volcanic-eruption* (bigram)). For example, Table 2 shows that the first document is labeled as UNRELATED; the document does not contain the word *geology*, nor the sequence of *volcanic* and *eruption*. The second document is labeled as SUPPORT; in this document the word *geology* occurrs once, and the sequence of *volcanic* followed by *eruption* occurrs three times.

label	geology	...	volcanic-eruption	...
UNREL.	0	...	0	...
SUP.	1	...	3	...
...	...	...	...	...

Table 2. Formatted response data for the classification task.

With the coded data, we evaluated the performance of the model using 10-fold cross-validation. In other words, we first split the data into 10 partitions, and trained the model with 9 partitions and evaluated it with the remaining partition. This process was repeated 10 times such that each partition serves as a testing partition once. Algorithm 1 summarizes this process.

The performance of this classifier is summarized in Table 3, using standard precision, recall, and F1 (i.e., the harmonic mean of precision and recall) measures, on all the 400 annotated papers. All in all, the F1 score was 82.4%, which we consider an encouraging result, especially considering the small size of the annotated dataset.

label	precision	recall	F1	N
NEG.	0.000	0.000	0.000	2
NEG.&SUP	0.333	0.333	0.333	6
SUP.	0.707	0.680	0.695	85
UNREL.	0.907	0.919	0.913	307
Overall	0.851	0.854	0.853	400

Table 3. Performance of the linear SVM classifier. N indicates the number of papers in each class.

With the linear SVM classifier, one can inspect the feature weights for each label to be predicted (i.e., the relative importance of each feature on each label). Table 4 shows the top 10 features for each label in the trained model. Even though not all top 10 features are strongly related with volcanism or

[3] https://scikit-learn.org/stable/

climate change, we find that some features were related with either volcanism (e.g., *volcanic CO*) or climate change (e.g., *cooling trend, fire regime, of flood*).

Algorithm 1: SVM classifier with 10-fold cross-validation

```
CV = 10 batches of the data
predictions = []
true_label = []
for test_data in CV do
    train_data = CV - test_data
    train_feature =
      get_features(train_data(text))
    train_label =
      get_labels(train_data(label))
    classifier = SVM()
    SVM.train(train_feature, train_label)
    prediction =
      SVM.predict(get_features(test_data(text)))
    true_label.append(get_labels(test_data(label)))
    predictions.append(prediction)
print(classification_report(prediction,
  true_label))
```

Ranking	NEGATE	NEGATE&SUPPORT	SUPPORT	UNRELATED
1	We found	may be	nannoplankton	montane
2	after tephras	both	that	lacustine
3	and increases	little	tree	Sweden
4	and that	The authors	Our	study
5	and vegetation	best correlation	10	oceanic
6	consistent statistically	efficiency	biological	history Received
7	conspicuous	extinctions the	the atmosphere	driven
8	cooling trend	of flood	from	2012 Accepted
9	deposition of	volcanic CO	anoxia	Ordovician
10	fire regime	1999	detection	12 December

Table 4. Top 10 feature weights for each label extracted by the linear SVM classifier.

NB-SVM Classifier

The linear SVM classifier uses the frequency of unigrams/bigrams as the feature values. However, Wang & Manning (2012) showed that using instead the log-count ratios produced by a Naïve Bayes (NB) model performs better for a binary classification task. Here we adapt this idea to multi-class classification, as detailed below.

Log-count ratio

Let $f^{(i)} \in R^{||V||}$ be the feature count vector for training example i with label $y^{(i)} \in \{\boldsymbol{negate}, \boldsymbol{negate}\&\boldsymbol{support}, \boldsymbol{support}, \boldsymbol{unrelated}\}$. V is the set of features, and $f_j^{(i)}$ represents

the number of occurrences of feature V_j in training case i. For example, define the count vectors as $\boldsymbol{p} = \alpha + \sum_{i:y^{(i)}=negate} f^{(i)}$ and $\boldsymbol{q} = \alpha + \sum_{i:y^{(i)}=negate\&support,support,unrelated} f^{(i)}$ for smoothing parameter $\propto$. For example, the log-count ratio for the label **negate** is:

$$\mathbf{r_{negate}} = \log\left(\frac{\mathbf{p}/||\mathbf{p}||_1}{\mathbf{q}/||\mathbf{q}||_1}\right)$$

As a result, we have four different **r** ratios for NEGATE, NEGATE&SUPPO RT, SUPPORT, and UNRELATED.

SVM with NB features

This classifier, henceforth referred to as NB-SVM, is similar to the previous linear SVM, with the exception that we use $\boldsymbol{x}^{(k)} = \tilde{\boldsymbol{f}}^{(k)}$ where $\tilde{\boldsymbol{f}}^{(k)} = \hat{r}_i \circ \hat{f}^{(k)}$ is the element-wise product and $i \in \{negate, negate\&support, support, unrelated\}$ (e.g., the element-wise product of the ratio $\boldsymbol{r}_{negate}$ and $\boldsymbol{f}^{(k)}$).

With the given parameters, four different SVMs (**NEGATE** vs. rest, **NEGATE&SUPPORT** vs. rest, **SUPPORT** vs. rest, and **UNRELATED** vs. rest) were trained using different ratios. As a result, for SVM_i *where* $i \in \{negate, negate\&support, support, unrelated\}$, $x^{(k)} = \tilde{f}^{(k)} = \hat{r}_i \circ \hat{f}^{(k)}$ and w_i, b_i could be obtained using the *linearSVC* module in *scikit-learn* package.

The original paper suggested the model $\boldsymbol{w}' = (1-\beta)\underline{w} + \beta\boldsymbol{w}$ where $\underline{w} = \left|\left|\boldsymbol{w}\right|\right|_1/|V|$ is the mean magnitude of $\boldsymbol{w}$ *and* $\beta \in [0,1]$ is the interpolation parameter. In the current model, $\boldsymbol{w}_i'$ could be obtained by using $\boldsymbol{w}_i'$ of $\boldsymbol{SVM}_i$ where $i \in \{negate, negate\&support, support, unrelated\}$.

For the prediction, each $\boldsymbol{SVM}_i$ classifier makes a prediction $y_i^{(k)} \in \{-1, 1\}$. For example, $\boldsymbol{SVM}_{negate}$ returns 1 if the prediction is true (in this case, the classifier would return 1 if prediction for the test *k* is NEGATE) and -1 elsewhere. For $\boldsymbol{SVM}_i$, the prediction for the test case *k* is

$$y_i^{(k)} = sign(\boldsymbol{w}_i^T \boldsymbol{x}^{(k)} + b)$$

where $i \in \{negate, negate\&support, support, unrelated\}, \boldsymbol{w}_i$ *is* $\boldsymbol{w}_i'$, *and* $\boldsymbol{x}^{(k)}$ *is* $\boldsymbol{r}_i \circ \tilde{\boldsymbol{f}}^{(k)}$. After that, **argmax** is applied to the result of the SVMs to obtain a prediction with the highest score. Thus, $i = argmax\ y_i^{(k)}$ will be the prediction for the test case *k*.

As in the evaluation of the previous linear SVM classifier, we also evaluated the performance of the NB-SVM classifier using 10-fold cross-validation. The difference here is that we tried four different NB-SVMs (i.e., four one-vs-rest NB-SVM classifiers) for each label, and we applied **argmax** over the 4 predictions at the end to select the best one, i.e., the one with the highest score (see Algorithm 2).

Algorithm 2: NB-SVM classifier with 10-fold cross-validation

```
CV = 10 batches of the data
predictions = []
true_label = []
for test_data in CV do
    train_data = CV - test_data
    train_feature =
      get_NBfeatures(train_data(text))
    train_label =
      get_labels(train_data(label))
    test_feature =
      get_NBfeatures(test_data(text))
    test_label = get_labels(test_data(label))
    temp_prediction = []
    SVMs = [4 NB-SVM classifiers]
    for svm in SVMs do
        SVM.train(train_feature,
          train_label)
        pred = SVM.predict(test_feature)
        temp_prediction.append(pred)
    prediction = argmax(temp_prediction)
    true_label.append(test_label)
    predictions.append(prediction)
print(classification_report(prediction,
  true_label))
```

Table 5 lists the results of the NB-SVM classifier. Similar to the observations of Wang & Manning (2012), we observed that this classifier performs better than the "vanilla" SVM, but, in our case, the improvement was not large. For example, the F1 score of the NB-SVM classifier was 83.75%, while the linear SVM's F1 score was 82.4%.

label	precision	recall	F1	N
NEG.	0.000	0.000	0.000	2
NEG.&SUP	0.250	0.167	0.200	6
SUP.	0.705	0.647	0.675	85
UNREL.	0.903	0.909	0.906	307
Overall	0.846	0.837	0.842	400

Table 5. Performance of NB-SVM classifier.

Multi-Layer Perceptron Classifier

This classifier uses a feed-forward neural network, or multi-layer perceptron (henceforth, MLP) that operates on the same features as the previous classifiers, i.e., unigrams and bigrams with values set to their frequency in the corresponding document.

With the coded data as shown in Table 2, we evaluated the performance of MLP model using 10-fold cross-validation. In other words, we first split the data into 10 partitions, and trained the model with 9 partitions and evaluated it with the remaining partition. This process was repeated 10 times such that each partition serves as a testing partition once. To build the MLP model, we used the *Keras* package in the Python programming language. The MLP model had two hidden layers with 256 hidden units, and predicted one of the four classes (support, negate, support and negate, unrelated). The *Adam* optimizer was used with a learning rate of 0.0001.

Table 6 shows the confusion matrix of MLP model prediction, and Table 7 lists the results of the MLP classifier. As the table indicates, the MLP classifier showed the better overall performance, outperforming both the SVM and NB-SVM classifiers. For example, the F1 score of the NB-SVM classifier was 83.75%, while the linear SVM's F1 score was 82.4%. However, as the confusion matrix in Table 6 indicates, this classifier does not perform well on underrepresented classes such as NEG. and NEG.&SUP. Handling underrepresented classes remains an open research issues in machine learning.

True \ Prediction	NEG	NEG.&SUP.	SUP.	UNREL.
NEG.	0	0	1	1
NEG.&SUP	0	2	2	2
SUP.	0	0	58	27
UNREL.	0	2	21	284

Table 6. Confusion matrix of true and predicted labels.

label	precision	recall	F1	N
NEG.	0.000	0.000	0.000	2
NEG.&SUP	0.500	0.333	0.404	6
SUP.	0.707	0.682	0.695	85
UNREL.	0.904	0.925	0.915	307
Overall	0.853	0.863	0.856	400

Table 7. Performance of Multi-Layer Perceptron (MLP) classifier.

Ensemble Model

Lastly, we build an ensemble model that combines the predictions of these three individual classifiers. Our ensemble method uses a simple voting scheme:

1. When the predictions of three models are the same (e.g., NEGATE, NEGATE, NEGATE), then that label (e.g., NEGATE) becomes the final output.

2. When the predictions from the three models are different, and one of the predictions is UNRELATED (e.g., SUPPORT and UNRELATED), then the prediction which is not UNRELATED becomes the final output (e.g., SUPPORT).

3. When the predictions from the three models are different and neither of them is UNRELATED, then choose the prediction from MLP.
4. When the predictions from the three models are different and any of them is UNRELATED, then choose the first prediction that is not UNRELATED, inspecting the classifiers in the following order: MLP, NB-SVM, and, lastly, SVM.

Note that the above ensembling heuristics may bias the model against predicting the UNRELATED label. However, this potential bias has no practical effect as it cannot override the heavy bias *towards* the UNRELATED class in the training data.

Table 8 lists the performance of this ensemble model. The ensemble performs better than NB-SVM models, but it performs slightly worse than the vanilla SVM and MLP models. For example, the micro-F1 score of the ensemble model was 85.2% and that of MLP model was 85.6%. Further, the performance of the ensemble model, vanilla SVM, and NB-SVM are all lower than the performance of the MLP model. Because the MLP method was the best overall, we used the MLP model output to classify the 957 remaining papers in our dataset, and generate the visualizations discussed in the main body of the paper.

label	precision	recall	F1	N
NEG.	0.000	0.000	0.000	2
NEG.&SUP	0.333	0.333	0.333	6
SUP.	0.713	0.671	0.691	85
UNREL.	0.900	0.912	0.906	307
Overall	0.850	0.855	0.852	400

Table 8. Performance of the ensemble model that combines the SVM, NB-SVM, and MLP classifiers.