UPPSALA
UNIVERSITET

Självständigt arbete i informationsteknologi
20 juni 2018

Improving armed conflict prediction using machine learning

ViEWS+

Valeria Helle
Andra-Stefania Negus
Jakob Nyberg

Civilingenjörsprogrammet i informationsteknologi

Master Programme in Computer and Information Engineering

UPPSALA
UNIVERSITET

Institutionen för informationsteknologi

Besöksadress:
ITC, Polacksbacken
Lägerhyddsvägen 2

Postadress:
Box 337
751 05 Uppsala

Hemsida:
http:/www.it.uu.se

Abstract

Improving armed conflict prediction using machine learning

ViEWS+

Valeria Helle
Andra-Stefania Negus
Jakob Nyberg

Our project, ViEWS+, expands the software functionality of the Violence Early-Warning System (ViEWS). ViEWS aims to predict the probabilities of armed conflicts in the next 36 months using machine learning. Governments and policy-makers may use conflict predictions to decide where to deliver aid and resources, potentially saving lives.

The predictions use conflict data gathered by ViEWS, which includes variables like past conflicts, child mortality and urban density. The large number of variables raises the need for a selection tool to remove those that are irrelevant for conflict prediction. Before our work, the stakeholders used their experience and some guesswork to pick the variables, and the predictive function with its parameters.

Our goals were to improve the efficiency, in terms of speed, and correctness of the ViEWS predictions. Three steps were taken. Firstly, we made an automatic variable selection tool. This helps researchers use fewer, more relevant variables, to save time and resources. Secondly, we compared prediction functions, and identified the best for the purpose of predicting conflict. Lastly, we tested how parameter values affect the performance of the chosen functions, so as to produce good predictions but also reduce the execution time.

The new tools improved both the execution time and the predictive correctness of the system compared to the results obtained prior to our project. It is now nine times faster than before, and its correctness has improved by a factor of three. We believe our work leads to more accurate conflict predictions, and as ViEWS has strong connections to the European Union, we hope that decision makers can benefit from it when trying to prevent conflicts.

Extern handledare: Mihai Croicu, Håvard Hegre, Frederick Hoyles
Handledare: Virginia Grande Castro, Anne-Kathrin Peters, Björn Victor
Examinator: Björn Victor

Sammanfattning

I detta projekt, vilket vi valt att benämna ViEWS+, har vi förbättrat olika aspekter av ViEWS (Violence Early-Warning System), ett system som med maskinlärning försöker förutsäga var i världen väpnade konflikter kommer uppstå. Målet med ViEWS är att kunna förutsäga sannolikheten för konflikter så långt som 36 månader i framtiden. Målet med att förutsäga sannoliketen för konflikter är att politiker och beslutsfattare ska kunna använda dessa kunskaper för att förhindra dem.

Indata till systemet är konfliktdata med ett stort antal egenskaper, så som tidigare konflikter, barnadödlighet och urbanisering. Dessa är av varierande användbarhet, vilket skapar ett behov för att sålla ut de som inte är användbara för att förutsäga framtida konflikter. Innan vårt projekt har forskarna som använder ViEWS valt ut egenskaper för hand, vilket blir allt svårare i och med att fler introduceras. Forskargruppen hade även ingen formell metodik för att välja parametervärden till de maskinlärningsfunktioner de använder. De valde parametrar baserat på erfarenhet och känsla, något som kan leda till onödigt långa exekveringstider och eventuellt sämre resultat beroende på funktionen som används. Våra mål med projektet var att förbättra systemets produktivitet, i termer av exekveringstid och säkerheten i förutsägelserna. För att uppnå detta utvecklade vi analysverktyg för att försöka lösa de existerande problemen.

Vi har utvecklat ett verktyg för att välja ut färre, mer användbara, egenskaper från datasamlingen. Detta gör att egenskaper som inte tillför någon viktig information kan sorteras bort vilket sparar exekveringstid. Vi har även jämfört prestandan hos olika maskinlärningsfunktioner, för att identifiera de bäst lämpade för konfliktprediktion. Slutligen har vi implementerat ett verktyg för att analysera hur resultaten från funktionerna varierar efter valet av parametrar. Detta gör att man systematiskt kan bestämma vilka parametervärden som bör väljas för att garantera bra resultat samtidigt som exekveringstid hålls nere.

Våra resultat visar att med våra förbättringar sänkes exekveringstiden med en faktor av omkring nio och förutsägelseförmågorna höjdes med en faktor av tre. Vi hoppas att vårt arbete kan leda till säkrare föutsägelser och vilket i sin tur kanske leder till en fredligare värld.

Contents

1 Introduction

Though it might not be apparent, machine learning is something that we often encounter in our daily digital lives. Functions such as targeted ads, spam filters and image recognition all use the results of machine learning algorithms. The power of machine learning comes from using often large sets of data to create models and teach software to predict or make decisions that are hard or even impossible for a human to make. Machine learning can be used to predict many things, such as stock values, weather, movie preferences, and, as is the case for this project, armed conflict.

The ViEWS team is a research group working with conflict prediction using machine learning. They use conflict data (with variables such as past conflicts, access to water and child mortality) to try to predict where in the world armed conflict will occur. This is done by feeding the conflict data to machine learning algorithms, which are then used to predict probabilities of conflict. Conflict prediction may be used by governments and policy-makers in order to decide where to deliver aid and resources, potentially saving lives.

The need for the improvements we implemented stems from a number of issues that exist within the current system. Currently, the ViEWS team uses their research and theoretical experience to manually select algorithms, their parameters and variables from the dataset. Manually selecting what parts of data are important may lead to missing less obvious relationships within the data. Likewise, manually choosing which parameters to feed a machine learning function is certainly viable, but better results can be obtained by analysing which parameters lead to the best algorithm performance. Furthermore, only a small number of machine learning algorithms had been tested, and it was possible that others would perform better.

Thus, we split our project, which we call ViEWS+, into three major parts. In the first part, we created an automatic method for feature selection. This tool finds the features from the ViEWS dataset that are the most important for predicting conflict. Using fewer, more important features should make the predictive process more efficient and perhaps even more correct by leaving out irrelevant features.

In the second part of the project we implemented a tool to evaluate the performance of different machine learning algorithms together with the conflict data. Testing the two together would make it possible to identify which features are best suited to the task of conflict prediction.

In the final part we selected parameter values for the algorithms, with the goal of increasing the efficiency of the proposed tools even further. We wanted to find ways to automatically optimize the chosen machine learning algorithms for the task of conflict prediction. This is in order to make sure they perform well without becoming too computationally expensive to use.

These improvements made the system more efficient, enabling not only faster but also better predictive ability. It is now nine times faster, and its correctness has improved by a factor of three compared to the results obtained prior to the implementation of our tools.

2 Background

Conflict and war have been the focus of large amounts of research throughout history. Håvard Hegre et al. [HS16] explain that there are certain factors that can be used to reliably predict conflict. These include average income, political instability, low rates of economic growth and other similar socio-economic data.

The increase in volume of data available to conflict researchers and the improvements of statistical and computational tools, such as machine learning, make predictions gradually more reliable.

Beyond providing information on the basics of machine learning, this chapter also addresses conflict prediction, our stakeholder and the dataset we are using for this project.

2.1 Machine Learning

Machine learning can be defined as the science of programming computers so that they can learn from data: machine learning software must get better at a certain task the more it performs it [Gé17, p. 4]. One of its main advantages is that instead of having to manually configure a program so that it can handle new data as it becomes available, a machine learning program can be trained to handle it without the need for such configuration. For example, manually creating a spam filter for any emails that contain "free money", means that once the emails start coming with "fr33 money" the filter will not work for these cases. The user would need to add the new text to the filter. If instead the spam filter is implemented using machine learning techniques, it can notice that emails containing "fr33" are often marked as spam by users, and can start filtering out such messages without user input [Gé17, p. 4-7].

One of the ways to classify machine learning systems is by the amount of supervision involved in the training of the system [Gé17, p. 7]. The two most common categories in this case are: supervised and unsupervised learning. Supervised machine learning is the more widely used of the two and is so called because the data scientist acts as a supervisor teaching the algorithm what conclusions it should reach [Cas18]. In supervised learning, the data set used for training comes labeled with the correct outputs expected from it [Gé17, p. 8]. For example, if we are trying to create an algorithm that identifies spam, the training set would be emails which are marked as either spam or not spam. In unsupervised learning, the machine is simply provided with data and it tries to find patterns within that data [Gé17, p. 10]. Take for example an online store that wants to suggest new products to customers based on their purchases. They could use unsupervised learning algorithms to discover that customers who purchase high-heeled shoes also buy adhesive bandages or occasionally earrings, and could suggest these two products on the page for purchasing shoes.

Supervised machine learning problems can be divided into two types: regression and classification problems. Regression problems deal with quantitative, continuous outputs, while clas-

sification problems deal with qualitative, discrete outputs [JWHT13, p. 28-29]. For example, determining a stock price is a regression problem since the stock price can take on values from a continuous range. Predicting whether a person likes a song or not is a classification problem as the outcome is one of the two discrete values, like or dislike.

2.1.1 Hyper-parameters

The way machine learning algorithms "learn" is by finding parameters to a function or model that will fit the input data, and this process is usually called "training" the model. The internal parameters of the model are not set by the users but by the algorithms during training, using different techniques depending on the algorithm [JWHT13, p. 21-22].

In addition to internal parameters, some machine learning functions have *hyper-parameters* that have to be chosen manually. In order to explain hyper-parameters we can imagine a machine learning function that makes its decisions based on the values of neighboring data.

For example, a simple machine learning system would determine whether a strawberry is ripe by looking at the status of strawberries in (a small number) of nearby bushes that were checked by the (human) growers in advance. The distance around the investigated strawberry bush that the system will consider "nearby" is the hyper-parameter in such a system.

A higher distance would certainly give more data to the system to look at, as there will be a higher number of bushes that the system will check, and thus the more certain (statistically) the system will be about the status of the bush. However, it comes with trade-offs: the higher the distance, the more different the conditions for strawberry growing will be (e.g. sunlight or soil). Thus, it can potentially lead to errors from over stretching the area where the conditions are comparable.

Such trade-offs are common with all machine learning systems, and thus, running an algorithm to test and best determine these hyper-parameters helps in improving final results for the machine learning system.

2.1.2 Feature Selection

A set of statistical data is usually divided into features, or variables. These are the columns of the table, the different categories of data collected. Machine learning needs such sets of data in order to identify patterns, similarities and be able to predict the outcomes we are interested in, such as the price of a stock, or whether or not someone will like a movie.

It is possible that in the set of features used for machine learning some of them do not contribute to or even decrease the quality of the predictions. To illustrate this, take the example of a biologist attempting to predict the height of trees using statistical models. It may then be relevant to use input data such as the type of tree, climate conditions and soil composition since these

are factors known to contribute to the height of a tree. Less relevant data may be the fact that there was a car parked next to the tree, whether the tree has birds living in it or the height of surrounding grass. By introducing these redundant features to the statistical model it becomes harder to determine the actual relationships within the data, since the irrelevant features may obscure them.

With the tree example it was fairly obvious what features were irrelevant, but this is not always the case. There is therefore a need for feature selection methods in order to algorithmically determine which subsets of the input variables contribute most to the output. There are different methods for achieving this such as removing features with a certain amount of variance, comparing different combinations of features or using machine learning methods that automatically choose the most important features [JWHT13, p. 204-207].

A simple way of approaching feature selection is to look for linear relationships between the feature we are predicting and the other features in the dataset. A correlation matrix shows these kinds of relationships. It measures, on a scale from -1 to 1, the linear correlation between two features. To give a simple example: if there is a positive correlation between feature_1, *amount of annual rainfall* and feature_2, *tree height*, then when the value *amount of annual rainfall* increases so does that of *tree height*. A negative correlation in this case could be between feature_3, *number of logging industry workers in the area*, and the *tree height*.

2.1.3 Evaluating Performance

There are several performance metrics available to evaluate the performance of a machine learning algorithm. Before examining them, we will first look into cross-validation which is a method used when evaluating the performance of such algorithms.

After training an algorithm, the next step would be to test it. This can be done by splitting the dataset into a training set and a test set. Once the algorithm is trained using the training set, it can check the quality of its predictions against new values: those in the test set. However, if there are several algorithms to be evaluated, repeatedly "showing" them the test set will lead to eventually adapting the algorithms not only to the training but also to the test set [Gé17, p. 30]. A possible solution to this is to split the data into three sets: training, test and *validation* sets. After training on the training set, the validation set is used to check the performance of the resulting algorithms. Then the test set is used as a single final test [Gé17, p. 30].

Splitting the dataset into several portions with dedicated purposes means that the amount of training data will decrease to allow for separate testing and validation sets. This is problematic because usually a larger training set leads to algorithms that produce better results. To avoid this, cross validation is used: the training set is split into subsets or folds, then the algorithms are trained with different combinations of these folds [Gé17, p. 30]. As a last step, the resulting algorithm is trained on the complete training set and its quality is evaluated against the test set [Gé17, p. 30].

In terms of performance metrics, the most common, and perhaps the most intuitive of them, is accuracy. Accuracy is the percentage of correct classifications out of all classifications, see Equation 1 [Kir17, p. 74]. Accuracy is usually a good metric for classification methods. However, when dealing with *sparse* data, a concept we explain in the next paragraph, its usefulness drops. Assume that we have a bag with 2500 black and ten white balls, and we are building an algorithm that predicts when we pick a white one. In this example, the cases when the algorithm guesses a white ball and we pick a white ball are called *true positives*. The cases when the algorithm guesses a black ball and we pick a black ball are called *true negatives*. There are also cases when the algorithm is wrong. If the algorithm guesses a white ball, but we pick a black ball this is called a *false positive*. Finally, the times when the algorithm guesses a black ball, but we pick a white ball are called *false negatives*.

The 2500 black and ten white balls in the bag give us a proportion of 250:1. Therefore we are dealing with a so called *sparse* dataset for our purpose of guessing whites. Because of this sparsity, if we train a machine learning algorithm with this data, even an algorithm always guessing black would get an accuracy score of 99.9%! However, if the goal is to predict whites, this is not a very good algorithm, thus accuracy is not the best measure of success in this case.

$$\text{Accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{true positives} + \text{false positives} + \text{true negatives} + \text{false negatives}} \tag{1}$$

Precision and recall are very important performance metrics. Unlike accuracy, they give a better evaluation of how the algorithm is performing when dealing with sparse data. Precision is the percentage of our positive predictions that are actually correct (see Equation 2), whereas recall is the percentage of positive results we managed to correctly predict out of all true positives (see Equation 3) [Kir17, p. 74].

In order to give these concepts a better explanation we are going to look into the black and white balls example again. Precision in this case is the percentage of times when the algorithm guessed a white ball and the guess proved to be correct. If our algorithm guessed a white ball ten times and six times the ball we picked was white, the precision is 60 %, as the algorithm made six correct predictions out of ten.

Recall is also known as the *true positive rate* of the function. If we picked several white balls, recall will be the percentage of times the algorithm said the ball will be white. If we picked ten white balls, but the algorithm guessed three balls would be black (*false negatives*) and seven balls would be white (*true positives*), then the recall is 70 %, as the algorithm made seven correct predictions out of ten.

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \tag{2}$$

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \tag{3}$$

Now, after illustrating precision and recall, it is time to explain what these values represent in the context of conflict prediction.

Take the example of an algorithm that predicts conflict in all the countries in the world. As there are few countries at war, there are very few true positives that can be identified, but they would all be correctly identified. The algorithm would identify no false negatives, since it does not mark anything as a negative. Finally, it would falsely mark as being at war the countries that are at peace, so there would be an extremely large amount of false positives.

Recall measures the proportion of the actual countries at conflict that have been correctly identified by our algorithm. As such, recall would be 100%, but despite the high performance, this would not correctly describe reality.

Precision, on the other hand, measures how many of the predicted countries at war are actually in conflict. If we take the same example, where we predict conflict everywhere in the world, then precision would be very low (close to zero). That is because most of the countries we have predicted to be at war are actually at peace, and the algorithm produced a large number of false positives.

Thus, there is a constant trade-off between precision and recall. The more countries the algorithm labels as ”at war”, the higher the recall (as the chance of identifying all the countries in conflict increases), but the lower the precision (as it will have labelled many peaceful countries as being in conflict by mistake). The ideal then is to have a balance between precision and recall, and this is where the *F score* is useful. The F score (Equation 4) is the weighted average of the precision and recall, and is scaled from zero to one, from worst to best [PVG+11a]. As it takes into account several measures of success, the F score provides a good single measure for the quality of our solutions.

$$\text{F score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4}$$

2.2 Decision Trees and Tree Based Methods

Tree based methods are a subclass of machine learning algorithms. These are algorithms based on *decision trees*, hence the name. Such a method was used by the ViEWS team prior to our work.

A decision or classification tree is a way of categorizing things by repeatedly separating them into smaller classes [Kir17, p. 70]. The tree of life is a classification tree, splitting organisms

into kingdoms, families and eventually species. At each split in the tree, data is separated into two categories with the portions that fulfil some criterion ending up in one branch, and those that do not ending up in the other. For example, cats, dogs and platypuses have spinal cords, so they are placed in the vertebrates branch, whereas insects, molluscs and crustaceans are placed in the other branch, invertebrates.

The goal of decision tree based learning is to make an algorithm that creates classification trees out of any data. During training a tree is constructed, one split at a time. At each split of the tree the algorithm will divide the data based on a single feature [Kir17, p. 71], such as the size of an animal. An index function calculates the size of the threshold that decides whether or not data will fall into one category or the other. The index function tries to minimize the amount of data that will fall into the "wrong" category by setting the threshold to a certain level. The process will gradually create smaller and smaller categories of data. New data supplied after training will simply follow the path in the tree that matches the features in the data and eventually fall into a category.

A lot of tree based learning is done by using multiple decision trees. Methods using many trees are called *ensemble methods*. The main hyper-parameter (see Section 2.1.1) contributing to performance for ensemble methods are the number of trees used.

2.2.1 Bagging

One of the problems with using ensemble methods is that in order to train many decision trees a lot of data is needed, preferably one independent dataset for each tree. In reality, this is not feasible since ensemble methods often use hundreds of trees. *Boostrap aggregating*, shortened as *bagging*, was created by Leo Breiman [Bre96] and is a way to avoid this problem. Bagging works by sampling data with replacement from the original dataset, thus creating multiple random training sets [Bre96]. Since some data will be repeated in the training sets it also creates the probability that other parts will be unique for each set. The training sets produced with bagging are then used to train a number of decision trees in parallel, and at the end the decisions of each tree are aggregated [Bre96] .

2.2.2 Random Forest

In 2001, Breiman published a paper [Bre01] about an improvement to the bagging algorithm called Random Forest. Similarly to bagging, Random Forest is based on bootstrap aggregation of data and multiple decision trees, but with an adjustment [Bre01]. Instead of choosing from all the variables/features when making splits, in Random Forest each tree only selects from a random subset of variables/features. This adjustment leads to Random Forests having the ability to measure variable importance as it can check which ones lead to the most correct predictions. They are thus more resistant to disturbances or errors created by irrelevant data [Bre01].

2.3 Scikit-learn: Machine Learning in Python

To implement a lot of different machine learning techniques from scratch would be time consuming. Luckily there are code libraries that make the process easier. *Scikit-learn* is a Python package containing a large number of tools for machine learning, including both the actual machine learning functions as well as the tools for preparing data and measuring performance [PVG+11b]. All machine learning methods in Scikit-learn adhere to interfaces which makes them interchangeable since they use the same method names. For example, regardless of the type of function they all have a `fit(X, y)` function for training and a `predict(X, y)` function for predicting. Another useful feature is the ability to pipeline objects, where different stages of preprocessing of data, model fitting and evaluation can be strung together into a single object.

Scikit-learn provides functions for Bagging and Random Forest, as well as a modified version of Random Forest that introduces more randomness called Extra Random Forest.

2.4 Uppsala Conflict Data Project (UCDP)

The Uppsala Conflict Data Project (UCDP) is a data collection project focusing on armed conflicts all over the world, assembling data about the number of deaths caused by armed conflicts and where they occur [Upp18b].

UCDP defines an armed conflict as a disagreement between two parties where armed force is involved and that results in more than 25 battle-related deaths per year [Upp18a]. There are three mutually exclusive categories of armed conflict depending on the type of warring parties involved. A conflict where at least one state is involved (e.g. a traditional war such as World War II) is classified as a state-based conflict; one where none of the parties is affiliated with a government is classified as a "non state-conflict", and one where civilians are attacked by a warring party is classified as "one-sided-violence". In this report we will simply use the term "armed conflict" or simply "conflict" to refer to all of these three types of conflict.

2.5 Conflict Prediction at ViEWS

ViEWS is an EU-funded project at Uppsala University with the goal of providing an early warning tool for the types of political violence defined by the Uppsala Conflict Data Project. Predictions are made on different levels of locality with the lowest being on the upcoming actions of actors such as militias, rebel groups or militaries. On a less local level is a grid which spans across the globe that divides it in squares sized 0.5^o in latitude and longitude, where predictions are made for each square, enabling a sub-national resolution. Lastly, there are also country level predictions [ViE18].

The ViEWS software is implemented in Python and R. For full scale forecasts, it can be run

on supercomputers available through UPPMAX, the Uppsala Multidisciplinary Center for Advanced Computational Science. However, it is possible to run it on less powerful computers with less input data (subsets) and/or fewer features.

2.5.1 The ViEWS Dataset

The ViEWS dataset we worked with was created and populated by the stakeholder and is stored in a relational database, with a total size of 2.8 GB.

It has 4 228 092 entries or rows, and stores 89 variables. With the exception of one boolean variable, only numeric variables are stored in it. All the variable names used throughout this report are taken from the stakeholders' dataset. These dataset variables are the features that were used in the machine learning solution we implemented. For the features that are the most relevant for our project, brief explanations will be provided as necessary. The full definitions of the features are available in Appendix A.

2.5.2 One Step Ahead (OSA)

The ViEWS simulations are run in a so-called "one step ahead" system. This means that data from the past is used to create the predictive algorithm, and data from the present is used to evaluate the quality of the predictions it produces.

Doing so makes it possible to use the data for predicting the future. For example, if we wish to build a machine learning model to make predictions 12 months into the future, we can use input data from 2013. Then the quality of its predictions can be evaluated against actual outcomes from 12 months later, since data from 2014 is available in the dataset. This teaches our algorithm (model) to predict outcomes for one year ahead. Thus, when fed new data from the present, it will be able to predict one year into the future. See Figure 1 for a visual representation of the process.

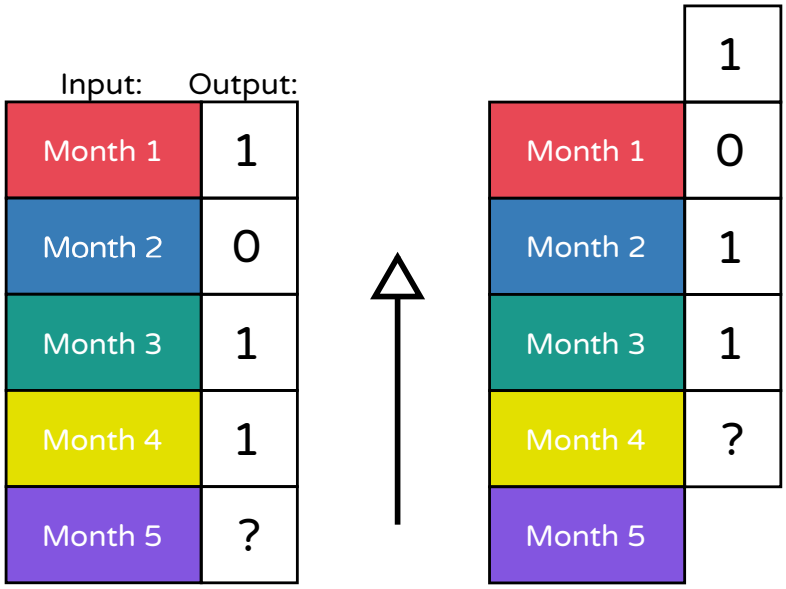

Figure 1 The "One Step Ahead" process. If we wish to predict the outcome 1 or 0 (1= conflict, 0= no conflict) in month 5 we can shift the output data one step backwards. We can train our model/algorithm using month 3 (but even 1,2 and 3). With the model we built we can predict month 4. To check that the model is correct we compare predicted 4 with actual month 4. Then we can use the data from month 4 to predict the outcome for month 5.

3 Purpose, Aims, and Motivation

The ViEWS+ project is aimed at expanding the functionality of an existing software system, ViEWS , by creating new tools for it. These include a feature selection tool, one that compares the performance of different machine learning methods, and a tool for identifying the appropriate hyper-parameters for the machine learning method used.

The tools were implemented using Python and supplied to the stakeholder, the ViEWS team, as a number of script files containing the requested functionalities.

The proposed tools did not yet exist in any form in the stakeholder's system. Due to time and human resource constraints, their current solution consists of researchers using their experience to hand pick features and parameters. As mentioned previously, automatically and systematically testing which features leads to better predictions. Also, using an unnecessary amount of input features or hyper-parameters that are too large leads to needlessly long execution times.

The initial system we created improvements for is aimed at predicting armed conflicts around the world. Thus, our improved system can be used for a higher precision conflict prediction research. Conflict predictions can, in turn, be used by decision makers to prevent conflict and deliver aid where possible.

3.1 Delimitations

We limited our work to only analysing prediction models within a time frame of 12 months. This limitation comes from the fact that the predictive capabilities of different features in the data sets may differ depending on the time frame. Adjustments to those differences require more changes in the code and more analysis. Similarly to weather forecasts, the more distant in time the prediction is, the less reliable it may become. If our tools work well for 12 months, they could then be expanded to work further into the future.

It is possible to predict multiple different variables in the dataset, such as the number of deaths, `ged_best`, or the number of violent lethal events, `ged_count`. However, we are limiting our work to predicting the variable `ged_dummy`, a variable indicating whether or not any type of conflict has occurred in that area.

One of the main delimitations for our project is working with a subset of the original database. This subset has a size of 2.8 GB, while the full set is about 80 GB. Only super-computers can handle this amount of data efficiently. As we are not working with all the data, the training performance may differ from what we might get if we were running it on the entire dataset. It is however the same in terms of sparsity and number of features, which allows us to design our tools to manage these attributes. Using a subset does not make the results less relevant. One of the key features of machine learning is its ability to be trained on a set of data to produce solutions that can then be reused on other sets of data of the same type, see Section 2.1.

4 Related Work

It is somewhat difficult to find works related to ours as it is both very specific, from the angle of enhancing a conflict prediction system, but also very general, since it deals with improving the performance of a machine learning system. Feature selection and parameter tuning are practices commonly used when working with machine learning and thus there are many different methods proposed for how it can be done.

One of the more general problems we dealt with is that the dataset has many features that need to be filtered. It is thus logical to look at other projects that deal with large numbers of features. In their paper [YP97], researchers Yang and Pedersen look at different feature selection methods for text recognition. A dataset for text recognition may contain as many as 10000 features, therefore the need for the feature selection is high. The researchers tested different scoring functions and concluded that the x^2 *statistic* and information gain were the most effective for feature selection in their case. Information gain is a measure used by the Random Forest algorithm in feature selection which lends credibility to us choosing to use the method for our purposes.

The `scikit-learn` documentation [PVG+11a] contains examples and articles where different concepts and functions are explained. One of these reports addresses hyper-parameter tuning. They showcase the functions `GridSearchCV` (explained further in Section 5.1) and

`RandomizedSerchCV` each representing a different way of parameter tuning. The former function is a comprehensive, brute-force search that can cover entire ranges and combinations of parameters. The latter is instead a random search, where random combinations of values are tested. They conclude that the comprehensive search is better when dealing with functions that have a low number of parameters, as the computational expense grows quickly when more are introduced. The random search is unaffected by the number of parameters, and thus better for functions with many parameter. Since the Random Forest algorithm only has one major parameter that contributes to performance, we chose to use a comprehensive search using `GridSearchCV`.

The unique challenges we faced were to implement fairly standard procedures into the existing system with all its peculiarities, ensuring that they perform correctly in spite of them. Since this part of the project is so specific to the system we are working with it is hard to find other similar projects. Instead, it may be worth looking at other projects that deal with conflict prediction in general.

4.1 Conflict Prediction Projects

In addition to ViEWS there are other groups aiming to predict the locations of armed conflicts. One of these is the aerospace and global security company Lockheed Martin. The company is currently developing software for predicting where in the world armed conflicts will occur through its Integrated Crisis Early Warning System (ICEWS) [Sub12, Chapter 2]. iCAST, one of the elements of the system, states that it provides six month forecasts for events such as domestic political crisis, international crisis, ethnic/religious violence, insurgency, and rebellion [Sub12, Chapter 2]. Their method combines forecasts from heterogeneous statistical and agent-based models, and as a result their aggregate forecast has an accuracy greater than 90% [Sub12, Chapter 2]. The tool can also evaluate user defined hypothetical scenarios.

In 2015 its dataset was made public [BLO+18]. A large portion of the iCAST system remains classified, including its methodology, predictions and evaluation strategy. Thus, our understanding of what it can do is limited to the brief description available on the company's website but it is worth noting that a major organization considers it an important field of research.

While a number of universities and organizations are conducting research in conflicts, not all researchers agree that the use of machine learning can make predictions more precise. One such case is the International Conflict Research at ETH Zürich. This department is carrying out research of violent conflict, as well as the ability to predict it. In an interview [Fab17], Professor Cederman of ETH Zürich states that the risk of a future armed conflict can be identified at an early stage, for example in regions with oppressed ethnic groups. It is very difficult though to actually predict exactly where and when armed conflict will occur in a region. He agrees that data science provides new tools that can be used in conflict research. Cederman is however critical of the optimistic view that predictions can be made much more precise and their temporal and spatial reach increased. In his view, amassing large quantities of potentially non-representative,

unverified data does not automatically lead to better results [Fab17].

In simpler terms, he does not believe that merely collecting lots of data and pushing it through algorithms will magically make it possible to predict conflict. What we may take from this, while still holding the assumption that conflict prediction is possible, is that feature selection should be used in order to keep the number of features as low, and important, as possible.

5 Method

We are using Python as our programming language. Python is already used by ViEWS which makes it easier for us to use existing code and integrate our improvements into the project. We are using several Python libraries for specific steps of our project, and their details can be found below.

An alternative to using Python would be using R, since it too is commonly used with statistics and has libraries for machine learning. However, the machine learning libraries for R are not as unified as Python's Scikit-learn module where all estimator functions have an interface they conform to (see Section 2.3). Scikit-learn's adherence to a single interface makes it easy to interchangeably use different functions that in R would require larger rewrites of code every time we add a new one.

5.1 Scikit-learn Functions

The previously mentioned Scikit-learn module is our source for most of the machine learning related functions used. We use the `sklearn.Pipeline` module to create pipelines. Pipelines are objects which let users string together different functions to transform the data before building a model. For example, we use a pipeline to string together a scaler, an imputer and a feature selector. The scaler normalizes the data, meaning that it changes the scales that different variables use so that the resulting scales are comparable. An imputer replaces empty data rows with a predefined value such as a mean or a median of existing values in the respective row.

The module `sklearn.model_selection` was used for comparing which parameters generate the best results, using the method `GridSearchCV`. It is a powerful tool, especially in combination with pipelines since it allows for cross validation with different functions in the pipeline. This means that if we wish to determine which combination of classifier and feature selector is best we can supply `GridSearchCV` with the appropriate parameter grid and it gives the results. Because of the sparsity in the data, the cross validation uses F score (Equation 4, Section 2.1.3), the combined measure of precision and recall, instead of accuracy for scoring the different alternatives.

5.2 Maintaining Reasonable Performance Needs

Running simulations on entire datasets is intensive for a computer's memory and processor. If a forecast uses a selection of features, only those specific parts of the database table need to be downloaded and stored in memory. However, in our case where all features are needed for testing, the entire database table has to be obtained. Carelessly storing too many copies of the data in memory quickly leads to memory space running out. By storing functionality that deals with data in functions, the Python garbage collector will quickly remove data from the stack as the function finishes.

In order to reduce processor use, we use Jupyter Notebook, an open-source web application that lets the user create and share documents that contain live code, equations, visualizations and narrative text [Pro18]. Jupyter makes it possible to run sections of our code selectively. This means that we can change and run less computationally intensive parts of our code without having to rerun the slow portion, such as loading the data, every time. Tree based ensemble methods like Random Forest demand more computation time as parameter magnitude, like the number of trees, increases. Choosing parameter values that give acceptable results without being needlessly large reduces unnecessary CPU usage.

As mentioned in Section 2.1.3, there is a problem with the rarity of data points that are classified as conflict. By removing a number of rows where the predicted outcome is "not conflict" from our dataset (a process called downsampling), we not only improve performance scores but also reduce the size of the training data. Downsampling is performed by a function that existed in the framework prior to our work.

We use the `pandas` Python library for its high performance data structures, such as data frames, and data analysis tools [AQR18]. Data frames are labeled tabular data structures that ease the process of dataset manipulation when dealing with large amounts of data [McK11]. The `numpy` library is used for its scientific computing functionalities and `pyplot` for creating plots of our results. The alternative to using data frames, provided the data is uniform, are matrices. Since matrices are uniform data structures, like arrays, they take up less memory than data frames which store any data type as well as information about the data frame. We have used both data frames and matrices at different points in the process, mostly using data frames when loading the data and selecting which columns to drop. After processing the data frames, they are converted to matrices for improved performance, and then fed to the machine learning algorithms.

6 System Structure

The conflict data ViEWS uses is stored in a PostgreSQL database. Any clients wanting to run predictions will have to connect to this database to fetch data. There are some alternatives for how to run the predictions. One alternative is to download the data directly to the client computer and run the predictions locally. Since the process is fairly computationally intensive there is also

the alternative to run the whole process remotely on a more powerful machine, which the ViEWS team supplies. The client machine will then interact with the remote server via, for example, a Jupyter interface which will return the results after the predictions are finished. A diagram of the system structure can be seen in Figure 2.

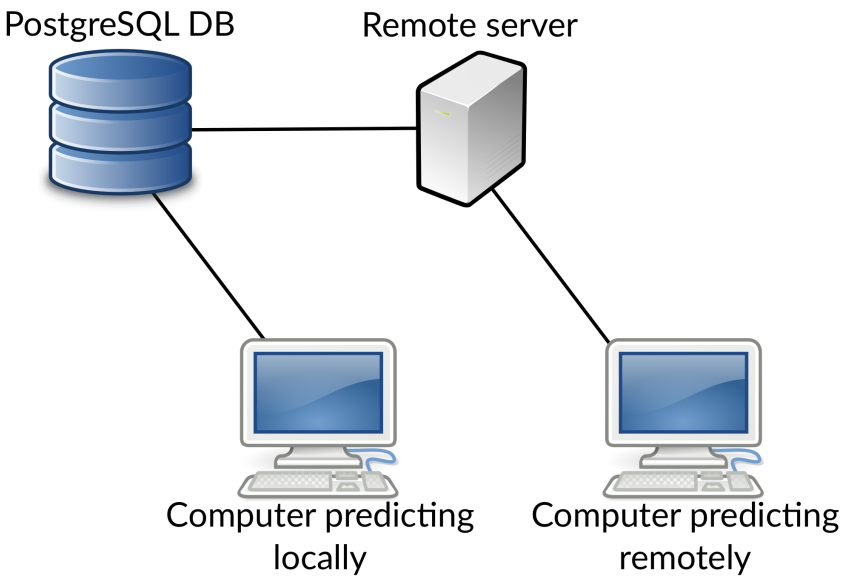

Figure 2 A simple structural description of how the ViEWS software can be run. The left computer fetches conflict data directly from the database and runs predictions locally. The right computer uses a more powerful remote server which fetches the data and runs the predictions. It then sends the results to the connected computer.

Although ViEWS has a fairly large codebase dealing with different aspects of prediction and data management, our work has focused on the components that deal with running the OSA forecasts, see subsubsection 2.5.2 for explanation of the OSA process.

To run an OSA forecast the researcher first creates a “model” object, represented by a Python dictionary, containing a number of data fields with relevant information to the forecast, such as the forecast length, which features to use and the feature to predict. The model is then supplied to a forecasting function which first fetches the data from the ViEWS database and shifts the input data by a specified number of steps. Another step in preprocessing is downsampling (see subsection 5.2) the data, removing a share of rows where the outcome is zero, zero representing ”not conflict”. The preprocessed data is then used for fitting a machine learning model which after training performs the prediction for the requested time frame. Prior to our project, the user had to manually enter features and hyper-parameter values for the model they wanted to run. A goal of our functions is to generate the model object and automatically fill certain fields with optimal estimators and features. The model is then passed along to the existing framework for predictions.

Our system roughly has three layers of analysis that the model can be passed through. Initially, the best features are selected, then the best classifier is decided and lastly the best hyperparameters for that classifier are decided. Afterwards, the model object can be used multiple times for prediction in the future by saving the settings to a file. A graphical representation of the process can be seen in Figure 3 and an illustration of how the improvements fit into the general system can be seen in Figure 4.

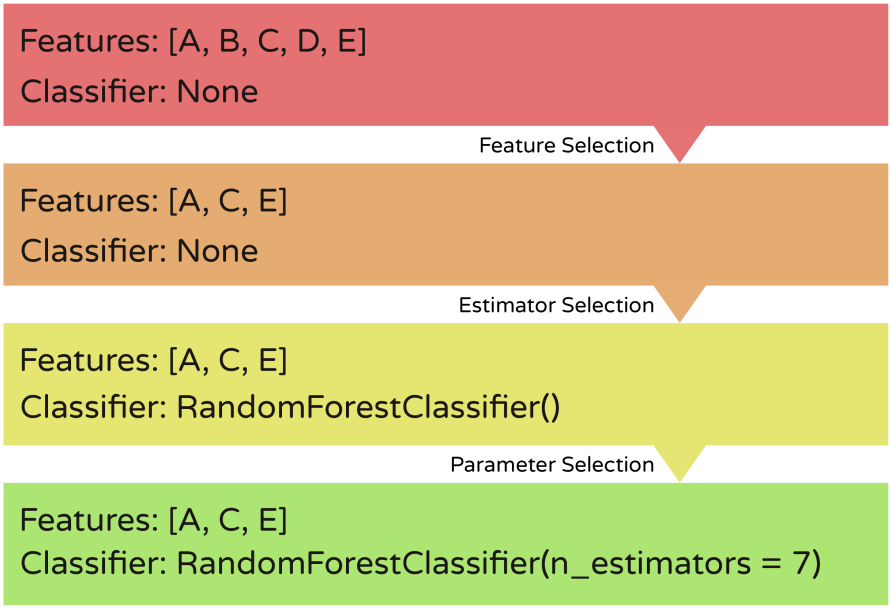

Figure 3 The steps we want to enable with our improvements. First, the relevant features are picked, then the best classifier can be chosen and finally appropriate parameters for that classifier can be found

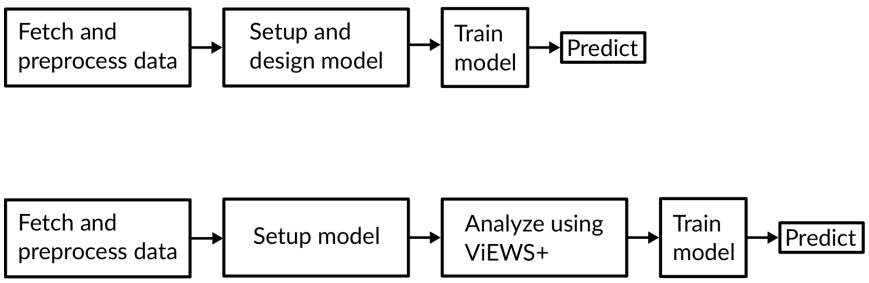

Figure 4 The steps of the prediction process and how ViEWS+ fits into it. Prior to our project researchers had to both set up their model and choose parameters and features in the same step. Now they only set up the basic information, such as when to predict, and then analyse the model with our tools to obtain the best features and parameters.

7 Requirements and Evaluation Methods

In order to identify the requirements for our project we looked at the results produced by ViEWS. Whatever performance they achieved we would be naturally required to improve.

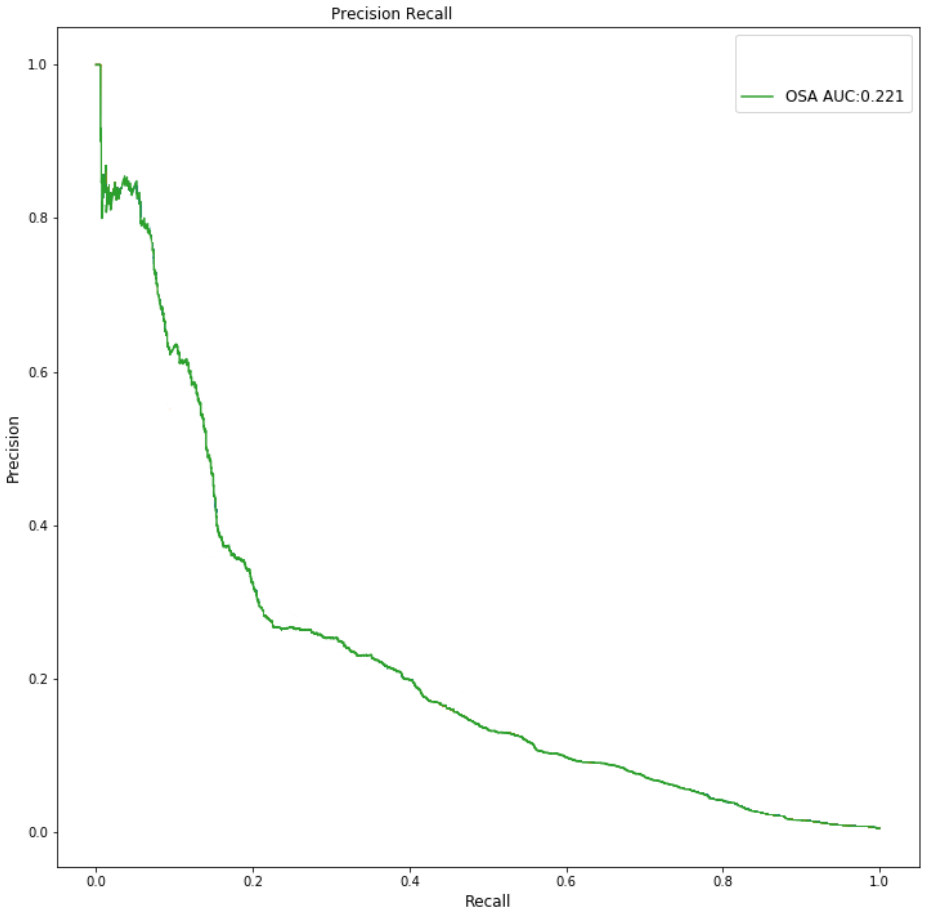

Figure 5 The precision-recall curve describing the performance of the current ViEWS system. The average precision, given by the area under the curve, is 0,22. OSA AUC stands for One Step Ahead Area Under Curve

The ViEWS team evaluates the performance of their work by using a precision-recall curve. The concepts of precision and recall are explained in detail in Section 2.1.3. A precision-recall curve

is normally used as a measure of success for predictions when the classes are very imbalanced. Precision shows how relevant the result is, while recall shows how many truly relevant results are returned [Sci18]. The curve can be described as a single number: the area under the curve, the *average precision*.

The formula for *average precision* or AP is shown in Equation 5 where P_n and R_n are the precision and recall values at the nth threshold [Sci18].

$$AP = \sum_n (R_n - R_{n-1}) P_n \tag{5}$$

The average precision that is obtained from this graph is 0.221. This number is the baseline for our improvements in terms of predictive performance. The area below the curve is on a scale from 0 to 1. A value of 0.221 was considered low by our stakeholders.

The stakeholder believes that the lack of performance is due to three factors: lack of feature selection (ViEWS uses over 100 features), lack of classifier tuning (as the hyper-parameter space was not explored), as well as structural factors such as high unbalance between classes (a very small proportion of ones to zeroes). The goal the stakeholders set was defined as at least 10% improvement in a standard evaluation metric taking into account precision and recall for the "1"-class (as the goal is predicting conflict).

In order to evaluate the results of our project in a comparable way, we use the same evaluation methods as our stakeholder: average precision.

The parameters of the model the ViEWS team was using before we started our work can be seen in the table below. We were required to improve these values.

Limit	0.5	0.386	0.1	0.9
F Score	0.313	0.269	0.093	0.051

Table 1 Evaluation scores for Random Forest with 250 trees provided by stakeholder, using different decision boundaries

Our stakeholders did not have any concrete requirements regarding reduction of execution time for forecasting, any reduction is considered a valuable addition.

Our functions are tested on a machine with 4 Xeon-class virtual cores and 32 GB RAM.

8 Descriptions of Implemented Functions

Our codebase is roughly split into four portions:

1. Old code created by the ViEWS team consisting of utility functions for connecting to the database and converting datasets, such as for shifting data for using the OSA method
2. A collection of new utility functions created by us, it is mainly used for loading conflict data from file.
3. Jupyter notebooks containing code that performs feature selection, parameter searching etc. Also displays result graphs.
4. A file containing the functions which perform the feature selection and parameter searches.

8.1 Revised Model Object

We decided to redefine the "model" objects (see Section 6), which were Python dictionaries before, to Python classes. Unlike dictionaries, classes can have their own functions to provide additional utilities, such as validating data that is sent to the model. Using a class instead of a dictionary also allows enforcing the type of the parameters ensuring data consistency. A Model class object is initiated using the same value fields that the dictionary, as class parameters, was using but with an added parameter `bad_features`. `bad_features` is a researcher-supplied list of features not to be used when doing feature selection. These include indices, erroneous columns produced by the database and other unwanted features.

8.2 Loading the Data

The file `views_download.py` uses some of the pre-existing functions for connecting to the database and downloads the entire table to a file. Having a database in a file makes it possible to work with the code even without an internet connection, if necessary. It also speeds up prototyping as the data does not have to be re-downloaded every time the program is run.

After the data is downloaded, it is loaded as a `pandas` DataFrame. From it, the any irrelevant features, such as error indices added by the database, are removed. Data prior to 1995 and after 2016 is also removed. Pre-1995 data has a lot of NaN (Not a Number) values, while data after 2016 has not been fully verified by the ViEWS team, reducing its reliability. In order to deal with the sparsity of the dataset for the target feature, ged_dummy, and reduce execution times, we kept all the ones but only a 10% random sample of the zeros, since too many zeros affect the accuracy of the predictions. A more detailed explanation of why this is done is provided in Section 2.1.3.

8.3 Feature Selection

In the ViEWS dataset we worked with there exist several variables whose values directly affect the value of the outcome *ged_dummy*, as this was how the database was designed. If any of the variables *ged_dummy_sb, ged_dummy_os, ged_dummy_ns*, *ged_count_sb, ged_count_os*, *ged_count_ns* or *ged_best_ns* has a value different from 0, then the outcome variable, *ged_dummy*, will be 1. All *ged_dummy* variables listed above record the existence of armed conflict. All *ged_count* variables record the number of violent lethal events, and the *ged_best* variable records the best available estimate of number of deaths. Their suffixes indicate the actors that were involved in the events recorded by the variable. The *_ns* suffix means that this was a non-state conflict where non-state armed groups are in conflict with one another. The *_sb* suffix refers to conflicts where the state was involved, that is intra-state, inter-state, internationalized conflicts. The *_os* suffix indicates one-sided violence, meaning cases where government forces or non-state armed groups engage in violence against civilians [ViE18].

The variable we want to predict is *ged_dummy*[1], a variable that records the existence of armed conflict. Its values can be either "1" or "0"; a one means that an armed conflict was registered in its geographical grid cell (see Section 2.5), and a zero that there were no such conflicts in that cell. There are far fewer areas affected by conflict than at peace, resulting in a ratio of zeros to ones of 255:1 (or 4 211 304 zeros versus 16 788 ones).

The low number of recorded conflicts, apparent in the ratio of zeros to ones in *ged_dummy*, and the direct connections of some columns to the outcome variable was taken into account when implementing our solution.

Feature selection is performed in two ways. The first is by including the ged features (*ged_dummy _sb, ged_dummy_os, ged_dummy_ns*, *ged_count_sb, ged_count_os*, *ged_count_ns* or *ged_best_ns*), but lagged by 12 months. Since for any time *t*, if any of these ged features is not zero then ged_dummy is one, there is no reason to check that connection. We know it exists, so testing it would be a waste of resources. Instead, we look for connections between the ged features at time *t-12* and *ged_dummy* at time *t*, to see whether a history of armed conflict or one of violent lethal events can predict current or future conflicts. In the second iteration, the ged features are dropped to check for other connections within the dataset completely outside those that are known.

In both cases, once the ged feature are dropped or lagged, we extract the labels for *ged_dummy*: a list with the values that it assumes within the sample dataset we are working with. We separate ged_dummy from the dataset since it is the target feature, the one we are trying to predict. The remaining data then goes through a pipeline with an imputer that fills in NaN fields with median values for the respective features. This is done because most machine learning algorithms do not work with missing values [Gé17] and thus the code will not run. We replaced missing

[1] A dummy variable is a boolean variable that originates in the transformation of a continuous/numeric variable (in the stakeholder's case this was the number of battles) into a categorical one (that can only take the values "1" or "0") [DS98] indicating in this case the presence/absence of conflict .

values with the median because this is the standard method [Gé17]. The pipeline also applies a standard scaler provided by `scikit-learn` in order to scale the features on a scale from 0 to 1, because machine learning algorithms do not work well when the input variables have very different scales [Gé17]. The importance of imputer and scaler functions is further explained in Section 5.1.

After undergoing these transformations, the dataset can now be used for feature selection. The feature selection process uses `GridSearchCV` with a 5-fold cross-validation, see Section 2.1.3, and performance scoring that uses the F score for a Random Forest Classifier. This produces, among other things, a list of the best features together with the feature importances obtained from the Random Forest. Some results from running this function can be seen in Section 10.1.

8.4 Parameter Search

Like with the feature selection, the data is preprocessed before it is used for parameter tuning. The main part of the parameter searching function is a call to the function `GridSearchCV` that compares different classifiers and parameters.

The grid search tests a series of classifier functions each with a range of trees, to be able to observe at which number of trees the function performs best. As in the feature selection process, F scoring was used for evaluation. The classifier functions used in the cross validation are all different variants of tree based methods: RandomForestClassifier, AdaBoostClassifier, GradientBoostClassifier, BaggingClassifier and ExtraTreesClassifier.

After running the `GridSearchCV`, the results are saved to file so that they can be used for later analysis. We implemented a function for plotting the scores of the classifiers compared to the number of estimators. An example of such a plot can be seen in Figure 10.

8.5 Classifier Comparison

Classifier Comparison is a Jupyter notebook that serves to compare a number of different machine learning functions. This is done by performing a number of predictions with each function and comparing the final scores to one another. All functions use their default parameter settings. The final results are presented as a bar graph, which can be seen in Figure 9.

9 Evaluation Results

In order to evaluate our work, we compared our results with the results of the system before the improvements. We generated our own precision-recall curve to compare with the one provided by the ViEWS team. The original curve can be found in Section 7.

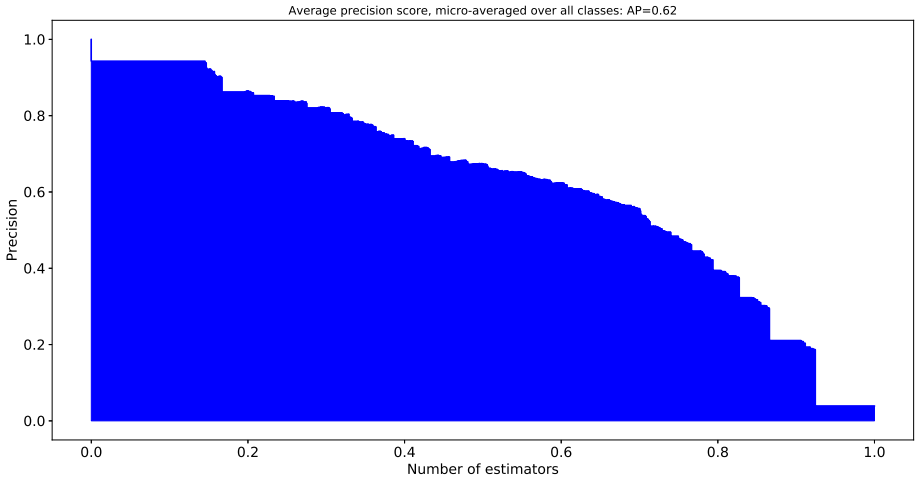

Figure 6 Precision-recall curve showing results after using our tools

The precision-recall curve has an area below the curve of 0.62 from a scale from 0 to 1. This is a significant increase from the initial value of 0.221 and indicates that the classifier has an improved performance.

After parameter searching, we found that the Random Forest classifier could be used with an amount of trees significantly lower than 250, which was the number originally used by the ViEWS team. The evaluation scores and execution times from our tests can be seen in Table 2. The table shows that using 25 trees results in an average precision of 0.62. Using 250 trees does increase the score slightly, but this comes at a cost of a nine times longer training time. The decrease in performance as the number of trees increases is shown in Figure 10.

(FS Off)/(FS On)	Score [AP]	Time [s]
25 trees	0.62/0.63	1.418/1.4
250 trees	0.65/0.65	13.24/13.58

Table 2 The comparative results for performance when using results from our tools. These show average precision (AP) and execution time depending on the number of trees and whether or not feature selection (FS) was used.

10 Results and Discussion

In this section we describe the results of our project, the tools we implemented to compare classifiers, select features and optimize parameters.

10.1 Feature Selection

For feature selection we first looked into the linear relationships of the outcome feature, *ged_dummy*, using a correlation matrix (see Section 2.1.2). A sample of the results for our dataset is listed below: the positive and negative correlations are in Table 3, and in full in Appendix B. One of the design decisions made by our stakeholders can be observed in these results. The built-in correlations between *ged_dummy* and *ged_dummy _sb / _os / _ns , ged_count_sb / _os / _ns*, and *ged_best_ns* (see Section 2.5.1), can be observed here: most of these features appear at the top of the correlation matrix.

As one of our project's goals was to increase the efficiency of the stakeholder's system, we can use the correlation matrix to select the features that with the strongest connections, positive or negative, to the one we are aiming to predict. We would thus choose to use in our machine learning model only a sample of the 89 features in the dataset: the ones with the highest positive and negative correlations. For example, the matrix indicates that *ged_dummy_sb* and *ged_dummy_os* have strong positive correlations with *ged_dummy*. As such, whenever they grow so does *ged_dummy*, the risk of conflict. The strongest negative correlations, with features like *ttime* (travel time to nearest city), mean that whenever they decrease then the risk of conflict increases.

A correlation matrix was used at the start of the feature selection process, but it can disregard non-linear relationships, such as when *feature_1* is at a certain threshold, *feature_2* decreases abruptly[Gé17].

Feature	Correlation	Feature	Correlation
ged_dummy	1.000	ttime_min	-0.099
ged_dummy_sb	0.748	ttime_max	-0.101
ged_dummy_os	0.604	dist_goldplace_s_wgs	-0.108
ged_count_sb	0.439	ttime_mean	-0.110
ged_count_os	0.417	barren_ih_li	-0.125

Table 3 Correlation matrix: positive and negative correlations

A more complex algorithm for feature selection, such as using the feature importance values of a Random Forest (see Section 2.2), can produce better, more nuanced results and can identify relationships beyond the merely linear ones. We created such a feature selection tool using a Random Forest to determine which features are the most significant. The function can return the relevant features and also provide a graph showing the importances, as seen in Figure 7. The bars show the relative importances of the different features. A high importance score means that the feature was important to the algorithm when making predictions about *ged_dummy*.

As the connection between the various *ged_variables* is built-in (see Section 8.3), strong and could skew the results, we first ran feature selection without these. A sample of the results is in

Figure 7, with the full results in Appendix D.

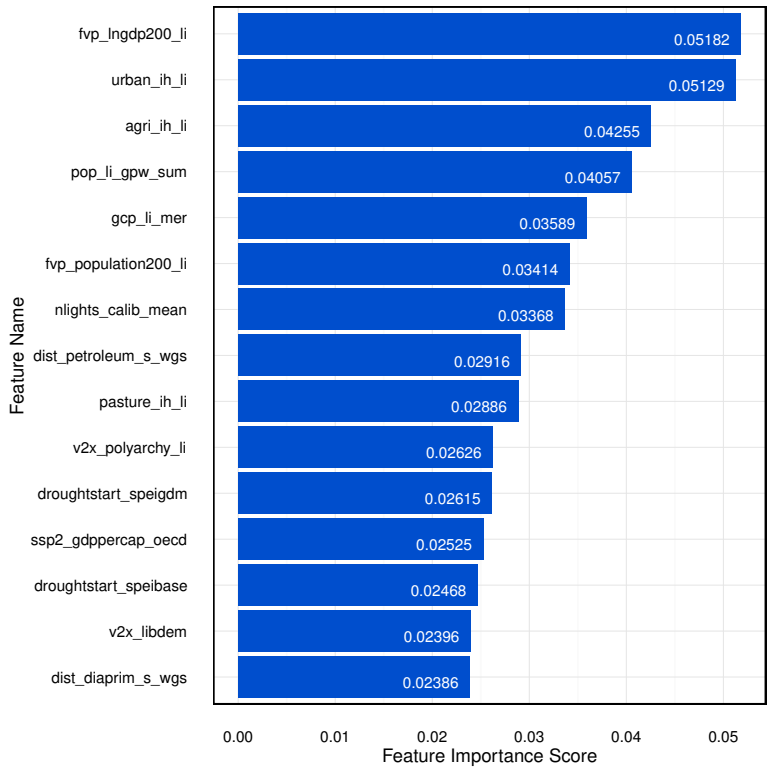

Figure 7 Example of feature importance graph produced by our tool. The bars visualize the relative importances of the different features, and a high importance score means that the feature was important to the algorithm when making predictions.

The sample graph above shows that features such as total gross domestic product for the country (*fvp_lngdp200_li*), percentage of cell occupied by urban areas or agricultural terrain (*urban_ih_li* and *agri_ih_l*i) were valuable in predicting armed conflicts (*ged_dummy*).

We ran the feature selection procedure but kept the ged features (ged_dummy_sb, ged_dummy_os, ged_dummy_ns, ged_count_sb, ged_count_os, ged_count_ns and ged_best_ns). In order to produce relevant results that can be used together with the OSA procedure, all the ged variables were lagged by 12 months. Keeping these features with their values at time t while trying to predict ged_dummy also at time t would be irrelevant, since the only thing we would find is the built-in relationship between ged_dummy and the other ged features (Section 2.5.1). Lagging these features can show if having conflict in the area 12 months before increases the risk of conflict in the present. A sample of the results is in Figure 8, with the full results in Table 6.

The sample graph above shows that features such as the number of violent lethal events 12 months prior (*ged_count_sb_lag12*) or population density (*pop_li_gpw_sum*) were valuable in predicting armed conflicts (*ged_dummy*).

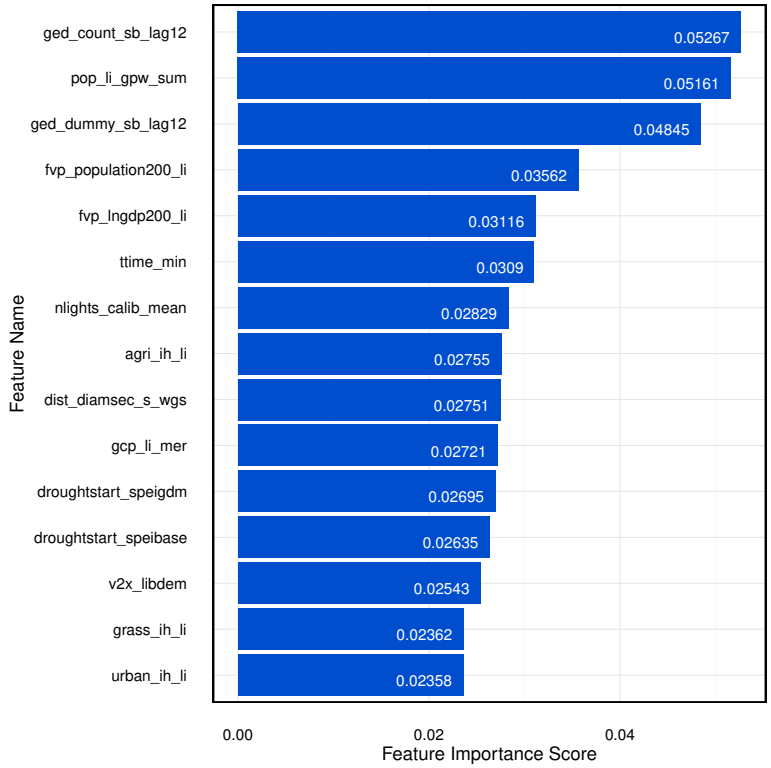

Figure 8 Example of feature importance graph: top 15 features (with ged features lagged by 12 months)

These results can be used in the same way as the results of the correlation matrix: to select from the a large pool of features only those features that have the highest importance when predicting *ged_dummy*.

The feature selection capability is a new one for the ViEWS project, and one that should be rather useful given that they are planning to add a large amount of new features to their dataset in the coming months.

10.2 Classifier Comparison

We implemented a tool for comparing different classifiers or functions. The graph produced by our tool can be used for comparing the performances of the different functions and selecting which one to use for predictions, see Figure 9 for an example.

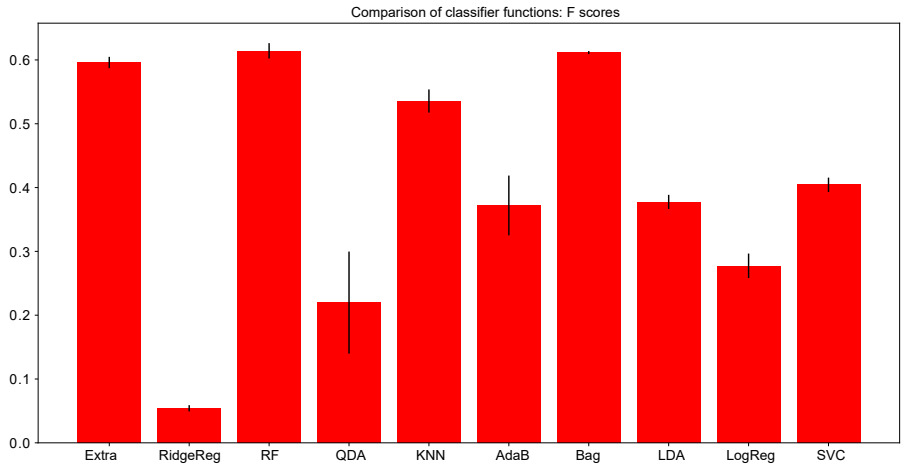

Figure 9 Example of output from classifier comparison script. The scores of different machine learning algorithms are compared side by side. Each bar (and label) represents a classification function in the Scikit-learn library. Error bars represent variance of score between cross validation sets. In this example, the Random Forest (RF) has the highest F score, so it should be selected as the function to be used for predictions.

10.3 Parameter Selection

The parameter selection function we implemented can be used to analyse the performance of different classifiers compared to the values of their parameters. In its current state the graphs it produces show the performance in relation to a single parameter. An example of a produced

graph can be seen in Figure 10. The graph can be used to determine at which value of the parameter the performance of the function, here measured using the F score, peaks or stops increasing.In the case of tree based methods, this prevents the use of an unnecessarily large amount of trees, which would give the same results but at a much higher computational cost.

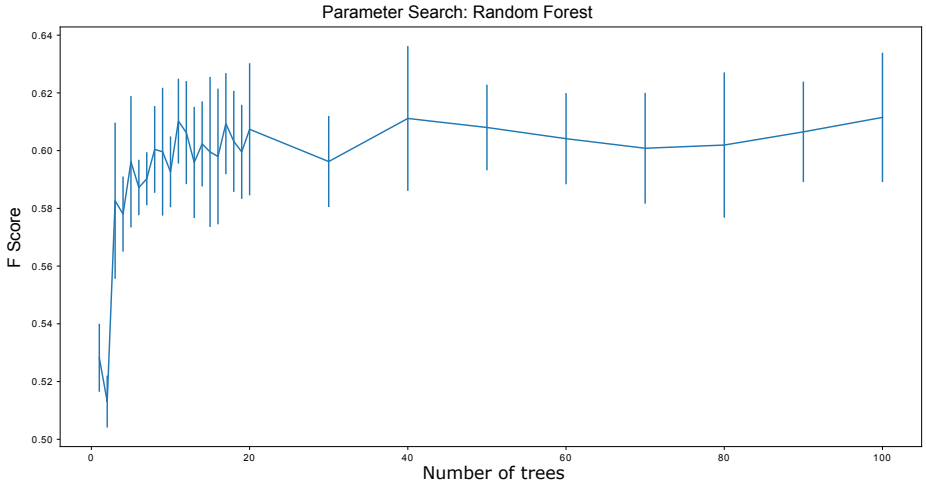

Figure 10 Example output of the parameter selection script analysing the performance of the Random Forest algorithm with different numbers of trees. The graph can be used to observe that the F score stops increasing significantly at around 20 trees. The error bars represent the variance of the the F score between cross validation sets.

11 Conclusions

This project has extended the machine learning pipeline of a large social-science prediction project attempting to forecast armed conflict. We had to deal with extreme levels of imbalance, large levels of noise, various levels of missing data, as well as multiple (sometimes conflicting) measurements of the same variables. Further, the dataset was very large (over four million observations and 89 features), requiring advanced data processing as well as sampling to make the problem viable.

Our tools enable the researchers to make more educated decisions in their implementations of the model. The results of these tools have been shown to decrease execution time significantly compared to the execution time before improvements. The time comparison can be found in Section 9.

We also implemented other machine learning models that the stakeholder had not implemented, and a tool to compare their performances. Further, we implemented evaluation routines for the

data and presented the stakeholder with extra performance evaluation metrics that work well with the type of data that was given to us.

The proposed improvements will have to be fully integrated in the overall ViEWS prediction framework for a complete evaluation as well as inclusion in the production environment. This complete integration will be done together with the stakeholder in the following months, as the current ViEWS code-base is frozen in preparation for a launch, and does not accept new features at this time.

We believe we have helped Uppsala University's Peace and Conflict Department to create an improved prediction toolkit, thus helping in turn with improving their forecasting capacity. Further, with the improved toolkit, we expect better computational performance, as ViEWS was using significantly conservative defaults against what we found were best-working models.

Thus, our work will lead to more accurate predictions of conflict. Further, given that ViEWS is deeply connected to the European Union, we additionally hope that certain decision makers will be able to benefit from our work by forecasting conflicts, and hopefully using this information to prevent them.

From these results we conclude that the project is successful.

12 Future Work

Due to time constraints, we did not implement a way of automatically assigning the optimal parameters to the functions. This this has do be done manually by analyzing the resulting graphs from the parameter selection tool. We also did not implement a tool for evaluating the effect on performance of different combinations of parameters and their values, such as tree depth and count. Both of these are functionalities that could be implemented in the future.

One of the possible directions for the future work will be refining predictions even further by reducing inconsistencies between predictions made on the sub-national grid level and national level. To better illustrate what that means think of the following situation. After an analysis the possibility of a conflict in country A might be calculated to be 0,0001%, so the chance of a conflict is fairly low. At the same time, the possibility of a conflict in A's capital is calculated to be 0,5%. This is because the state of the overall region is not represented in the data, only the state of the cell. Such results are inconsistent in comparison to each other, as the chance of a conflict for the whole country cannot be small if its capital has a likelihood to develop one.

Using machine-learning to eliminate these inconsistencies could be a future development of the current project. As to the scope of that future development, it will depend on the time frames of the project and the particular needs of the ViEWS team at the time.

References

[AQR18] AQRCapitalManagement. (accessed: 30.04.2018) Package overview. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/overview.html

[BLO+18] E. Boschee, J. Lautenschlager, S. O'Brien, S. Shellman, J. Starz, and M. Ward, "Icews coded event data," 2018. [Online]. Available: https://doi.org/10.7910/DVN/28075

[Bre96] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[Bre01] ——, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[Cas18] Castle, Nikki. (accessed: 22.04.2018) Supervised vs. unsupervised machine learning. [Online]. Available: https://www.datascience.com/blog/supervised-and-unsupervised-machine-learning-algorithms

[DS98] N. R. Draper and H. Smith, *Applied Regression Analysis*, 3rd ed. New York, USA: Wiley-Interscience, 1998.

[Fab17] Fabio Bergamin. (2017) Can conflict be predicted? [Online]. Available: https://www.ethz.ch/en/news-and-events/eth-news/news/2017/02/can-conflict-be-predicted.html

[Gé17] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*, 1st ed. Sebastopol, USA: O'Reilly Media, 2017.

[HS16] H. Hegre and N. Sambanis, "Sensitivity Analysis of Empirical Results on Civil War Onset," *Journal of Conflict Rresolution*, vol. 50, no. 4, pp. 508–535, 2016.

[JWHT13] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[Kir17] M. Kirk, *Thoughtful Machine Learning with Python*, 1st ed. Sebastopol, USA: O'Reilly Media, 2017.

[McK11] W. McKinney, "pandas: a foundational python library for data analysis and statistics," *Python for High Performance and Scientific Computing*, pp. 1–9, 2011.

[Pro18] Project Jupyter. (accessed: 01.05.2018) The jupyter notebook. [Online]. Available: http://jupyter.org

[PVG+11a] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[PVG+11b] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[Sci18] Scikit-learn developers . (accessed: 13.05.2018) Precision-recall. [Online]. Available: http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html

[Sub12] V. Subrahmanian, *Handbook of computational approaches to counterterrorism*. Springer Science & Business Media, 2012.

[Upp18a] Uppsala Conflict Data Program. (accessed: 12.04.2018) Definitions. [Online]. Available: http://www.pcr.uu.se/research/ucdp/definitions/

[Upp18b] ——. (accessed: 14.04.2018) About UCDP. [Online]. Available: https://www.pcr.uu.se/research/ucdp/about-ucdp/

[ViE18] ViEWS. (accessed: 04.04.2018) Views: a political Violence Early-Warning System. [Online]. Available: http://www.pcr.uu.se/research/views/

[YP97] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Icml*, vol. 97, 1997, pp. 412–420.

A Definitions of ViEWS Variables

Definitions are obtained from the ViEWS webpage [ViE18].

agri_ih_li = percentage agricultural terrain in cell

barren_ih_li = amount of barren land in cell

dist_diamprim_s_wgs = distance to nearest primary diamond resource

dist_diamsec_s_wgs = distance to nearest secondary diamond resource

dist_goldplace_s_wgs = distance to nearest gold resource

dist_petroleum_s_wgs = distance to nearest petroleum resource

droughtstart_speigdm = month of drought start, using SPEI GDM indicator

droughtstart_speibase = month of drought start, using SPEI Base indicator

gcp_li_mer = GDP per capita in cell, measured in USD.

ged_best_ns = Dummy variable indicating the best available estimate of number of deaths; ns=non-state conflicts, where non-state armed groups are in conflict with one another.

ged_count = Dummy variable indicating the number of violent lethal events.

ged_dummy_sb = Dummy variable indicating whether there was a conflict event (at least one battle related death) in a given grid cell in a given month; sb= state based conflicts, that is intra-state, inter-state, internationalized and extrasystemic conflicts

ged_dummy_ns = Dummy variable indicating whether a non-state conflict event (at least one battle related death) had occurred within the given grid cell. Non-state conflicts, where non-state armed groups are in conflict with one another

ged_dummy_os= Dummy variable indicating whether there was a conflict event (at least one death) in a given grid cell in a given month; os= one-sided violence, regards cases where government forces or non-state armed groups engage in violence against civilians

grass_ih_li = percentage grasslands in cell

fvp_lngdp200_li = total GDP in country, natural logarithm, Future of Violent Conflict

fvp_population200_li = The total population of a given country.

nlights_calib_mean = Mean intensity of nightlights, NASA Defense Meteorological Satellite Program, re-calibrated for sensor anomalies

pasture_ih_li = percentage pastures in cell

pop_li_gpw_sum = Population density, linearly interpolated, NASA/SEDAC Gridded Population of the World

ssp2_gdppercap_oecd = OECD level of GDP per capita

ttime= Log-transformed estimate of the travel time to the nearest major city.

urban_ih_li = percentage of cell occupied by urban areas

v2x_libdem = level of liberal democracy in country

v2x_polyarchy = level of polyarchy (multi-party system) in country

B Feature selection: correlation matrix results

Feature	Correlation
ged_dummy	1.000
ged_dummy_sb	0.748
ged_dummy_os	0.604
ged_count_sb	0.439
ged_count_os	0.417
ged_dummy_ns	0.411
ged_count_ns	0.301
pop_li_gpw_sum	0.210
agri_ih_li	0.187
urban_ih_li	0.148
nlights_mean	0.146
ged_best_ns	0.143
nlights_calib_mean	0.143
gcp_li_mer	0.136
ged_best_os	0.109
growend	0.106
mountains_mean	0.095
dist_petroleum_s_wgs	0.092
acled_count_pr	0.090
fvp_population200_li	0.081
excluded	0.066
shrub_ih_li	0.055
col	0.052
longitude	0.052
imr_mean	0.050
dist_diamsec_s_wgs	0.050
imr_max	0.049
cmr_max	0.047
pasture_ih_li	0.044
excluded_li	0.042

Feature	Correlation
fvp_polity2	-0.007
gem_s	-0.007
Unnamed: 0	-0.007
savanna_ih_li	-0.010
gwcode	-0.016
grass_ih_li	-0.021
bdist1	-0.036
droughtyr_speibase	-0.039
bdist1_li	-0.048
fvp_lngdp200_li	-0.050
country_month_id	-0.050
dist_gem_s_wgs	-0.051
country_id	-0.056
bdist2	-0.056
dist_goldsurface_s_wgs	-0.057
country_year_id	-0.059
ssp2_gdppercap_iiasa	-0.060
capdist_li	-0.061
ssp2_gdppercap_oecd	-0.061
capdist	-0.066
v2x_polyarchy_li	-0.067
v2x_libdem	-0.074
ttime_sd	-0.079
bdist3	-0.081
ttime_min	-0.099
ttime_max	-0.101
dist_goldplace_s_wgs	-0.108
ttime_mean	-0.110
barren_ih_li	-0.125
in_africa	NaN

Table 4 Correlation matrix: positive and negative correlations

C Feature Selection Results: Ged Variables Lagged by 12 Months

	Feature Importance	Feature Name
0	0.05266723458503051	ged_count_sb_lag12
1	0.051609192925615	pop_li_gpw_sum
2	0.04844793050750885	ged_dummy_sb_lag12
3	0.035620888464551734	fvp_population200_li
4	0.031156691963603055	fvp_lngdp200_li
5	0.030899833857193068	ttime_min
6	0.028292328757211122	nlights_calib_mean
7	0.027552347990981413	agri_ih_li
8	0.02750987484673716	dist_diamsec_s_wgs
9	0.027213724327607627	gcp_li_mer
10	0.026951358447213025	droughtstart_speigdm
11	0.02635175779609428	droughtstart_speibase
12	0.025429392642802723	v2x_libdem
13	0.02362395076054753	grass_ih_li
14	0.023584379140552447	urban_ih_li
15	0.023230597417089157	ssp2_gdppercap_oecd
16	0.02248809553401365	pasture_ih_li
17	0.02126258801822266	v2x_polyarchy_li
18	0.020576782109709157	dist_petroleum_s_wgs
19	0.01906867121823576	nlights_mean
20	0.018831893785704976	capdist_li
21	0.01720973207026157	dist_goldvein_s_wgs
22	0.016524440925061876	dist_diaprim_s_wgs
23	0.015930872780234962	excluded_li
24	0.015879378389341475	savanna_ih_li
25	0.014820419388504038	dist_goldplace_s_wgs
26	0.01459604554082613	shrub_ih_li
27	0.013504768528448138	capdist
28	0.011830795685275425	ssp2_gdppercap_iiasa
29	0.01173069021453485	dist_goldsurface_s_wgs
30	0.011445181880017254	dist_gem_s_wgs
31	0.010927281022811241	acled_count_pr
32	0.010731529366044832	ged_dummy_os_lag12
33	0.010655260353689397	bdist1_li
34	0.01059978210454611	droughtyr_speigdm

Table 5 Feature selection

35	0.010265770461413184	droughtyr_speibase
36	0.010115723998103211	imr_min
37	0.01001730681276646	ttime_max
38	0.009994025795678436	fvp_polity2
39	0.009875840009745902	forest_ih_li
40	0.009770463101507988	bdist1
41	0.009755135796781255	ttime_mean
42	0.009198774675676055	bdist2
43	0.008768116885096761	ttime_sd
44	0.00869927144293016	droughtcrop_speigdm
45	0.008458276999269902	ged_count_os_lag12
46	0.008230781695446804	droughtcrop_speibase
47	0.00812903024988248	imr_mean
48	0.007829625580771193	bdist3
49	0.007097741183627125	cmr_mean
50	0.0070603451333210825	mountains_mean
51	0.0060400812769134254	imr_max
52	0.005802665662334071	gwcode
53	0.005703751322784709	cmr_min
54	0.005319694086884608	cmr_sd
55	0.004927696537748439	cmr_max
56	0.004868783876186867	barren_ih_li
57	0.004816716706683207	ged_best_os_lag12
58	0.003932072406004797	growstart
59	0.0037821245542582205	ged_count_ns_lag12
60	0.0030558370528607558	excluded
61	0.0024974492830457527	rainseas
62	0.0021612275160509003	ged_best_sb_lag12
63	0.0018137699043446198	ged_best_ns_lag12
64	0.0015677975149417525	growend
65	0.0009448147821917685	ged_dummy_ns_lag12
66	0.00018730954500872552	diamprim_s
67	0.0001638686244992177	goldsurface_s
68	0.0001498305715220485	goldvein_s
69	0.00012233437450695687	diamsec_s
70	7.981494072253015e-05	gem_s
71	4.043629266645438e-05	petroleum_s
72	0.0	goldplacer_s

Table 6 Feature selection: continued

D Feature Selection Results: No Ged Variables

	Feature Importance	Feature Name
0	0.05182102396813292	fvp_lngdp200_li
1	0.05129343912253889	urban_ih_li
2	0.04255416531457814	agri_ih_li
3	0.04057147290450391	pop_li_gpw_sum
4	0.035885749759569645	gcp_li_mer
5	0.034136117903784104	fvp_population200_li
6	0.03367770554750393	nlights_calib_mean
7	0.029157966702347913	dist_petroleum_s_wgs
8	0.028863229279401503	pasture_ih_li
9	0.026255953888804306	v2x_polyarchy_li
10	0.026149644894182058	droughtstart_speigdm
11	0.025253114471827376	ssp2_gdppercap_oecd
12	0.024677681221776354	droughtstart_speibase
13	0.023963553225711528	v2x_libdem
14	0.023862703426329502	dist_diaprim_s_wgs
15	0.022588282492407907	dist_diamsec_s_wgs
16	0.021788073842188352	grass_ih_li
17	0.021081267715737127	mountains_mean
18	0.020669005562980357	nlights_mean
19	0.019509975130341712	dist_goldplace_s_wgs
20	0.01909152062362391	savanna_ih_li
21	0.01900696866074424	dist_goldvein_s_wgs
22	0.018388429134130885	shrub_ih_li
23	0.01730138704716988	acled_count_pr
24	0.016901267534656303	excluded_li
25	0.016669184349932514	dist_goldsurface_s_wgs
26	0.01604066710698922	ttime_min
27	0.014926131060565765	ttime_mean
28	0.014763720212533355	capdist_li
29	0.013293821782230444	dist_gem_s_wgs
30	0.012892435630055203	capdist
31	0.012801348618762198	ssp2_gdppercap_iiasa
32	0.012211125852987057	bdist1
33	0.01205808135789655	cmr_min

Table 7 Feature selection no ged_variables

34	0.012014975269386696	droughtyr_speigdm
35	0.011560426899413218	ttime_sd
36	0.011547882458728634	imr_mean
37	0.011199810438990118	bdist1_li
38	0.011077514632841864	droughtyr_speibase
39	0.010634910003369571	forest_ih_li
40	0.010629673778916627	bdist3
41	0.010439702705961902	ttime_max
42	0.010394866976858513	fvp_polity2
43	0.00981890559593319	cmr_mean
44	0.009389033034721052	imr_min
45	0.009037756790873349	bdist2
46	0.008185398224930356	droughtcrop_speibase
47	0.007633276615134209	droughtcrop_speigdm
48	0.006916907883746059	cmr_max
49	0.006182515707741764	imr_max
50	0.005943142925040994	barren_ih_li
51	0.005332189831849677	cmr_sd
52	0.004477531998752002	rainseas
53	0.0019296838215288427	gwcode
54	0.0016984433678417536	excluded
55	0.0015995788267006712	growstart
56	0.0011340374130837051	growend
57	0.00042522873230942294	gem_s
58	0.00018281533122660684	diamsec_s
59	0.00017144935586163926	diamprim_s
60	0.00016950607784886132	goldsurface_s
61	0.00010363100842112056	goldvein_s
62	5.348590256043501e-05	petroleum_s
63	9.503042502130933e-06	goldplacer_s

Table 8 Feature selection no ged_variables: continued