Spring 2-12-2018

Machine Learning and Threat Intelligence

Jacob Vangore
Olivet Nazarene University, jbvangore@olivet.edu

Machine Learning and Threat Intelligence

Jacob Vangore

Olivet Nazarene University

Table of Contents

Introduction

Over the summer of 2017, I had the opportunity to work with a software security team at Intel Corporation. During my time there, I worked on a project that attempted to data mine for threat intelligence information related to any Intel created software. This system would assist developers in being aware of the latest threat intelligence information as soon as it is made available. This would be done by crawling and scraping security websites and RSS feeds.

However, problems can and did occur with this system of data mining. Often times our system would retrieve what are known as "false positives." This is when a test returns the wrong result based on returning that a certain attribute is present when it is not. When datamining, we initially used a word bank and look for a string match within the RSS feed or website. RSS feeds often contain news articles and other information that isn't directly related to threat intelligence. This is where things got tricky. String matches would return ads and articles about threat avoidance techniques and not actual vulnerabilities and viruses that developers could be aware of for their software. In order to combat this issue, I proposed the idea of implementing some type of machine learning into our system.

Using machine learning I was able to add a greater degree of "intelligence" to my software system. Instead of simply looking for a string match, I trained a classifier with a hundred of pieces of text. The classifier uses an algorithm to learn what the features of a "relevant" piece of text are as well as what might make something appear like it's relevant when it actually isn't.

Using this technique, I was able to greatly cut down the number of false positives that my program would return. For the sake of time, I was only able to test one machine learning algorithm: Naïve Bayes. The Python library that I used offered a few other options for text classification. They were: Max Entropy and Decision Tree. I was not able to fully understand the entirety of each algorithm and compare them all to find which was the most effective at properly classifying text snippets. This however, spawned an interest within me. I took this opportunity to learn more about machine learning and its background. For my project, I have researched all three of the algorithms offered by the Python library and then trained three different classifiers to classify some text snippets and see which is the most accurate and useful. This resulted in some interesting finds as well as a lot of insight into how these algorithms are able to predict labels on newly inputted data.

Machine Learning

Machine Learning is often used as a buzzword nowadays. Large companies such as Spotify, Netflix and Google have all tried their hand at implementing some form of the technology in one way or another. For Google, this can be seen in their search engine as well as the ads that it chooses to show its viewers. In Netflix and Spotify this can be seen in their recommendation systems. But even with all of its prominence in the media and culture, the true idea behind machine learning is abstracted from the way that it actually works. The process simply boils down to statistics.

Machine Learning's definition is often unknown to the general population. In layman's terms is giving computers the ability to learn and act without being explicitly programmed to do so. A programmer can feed in a set of training data and an algorithm will allow a program to behave and make predictions based off of that data.

The applications of machine learning are endless. For example, the popular video streaming service, Netflix will use machine learning algorithms to try and recommend new movies and TV shows to its users. The input data is how whether a user completed a series or movie, or what a user has chosen to rate a movie. Data like this enables an algorithm to predict what to type of content to recommend for the user next. The idea is to get the most personalized behavior from the program for every user (Plummer, 2017). Another application that uses machine learning in a neat way is Snapchat. In order to detect a face, hundreds of images had to be inputted into Snapchat. Using the pixels in those images as a reference, the algorithm can detect a new face in real-time by looking for patterns in the pixels (Cade, 2016).

Not only can machine learning be seen in modern consumer applications but in many different fields as well. The stock market is always fluctuating and many people try to find patterns to figure out when to buy and sell their stock. This process can be simplified through machine learning. For trading, developers used multiple data inputs to train a model to be able to identify trends and predict new behavior for stocks (Heinz, 2017). Another application can be seen in the medical field. Classifiers can be trained to detect deformities as well as diseases. The possibilities of this kind of technology are endless. MRI's can train a classifier to identify a whether an image has the characteristics of a cancer or some other type of growth or disease (Abder-Rahman, 2017). The main advantage of a technology like this is that computers can be used in the diagnosis process. This often is more accurate and is better at calculating risk than simply using the judgement of a human. A classifier is trained for one purpose and is greater at determining what the characteristics for what might be a tumor or other disease. Being trained with a variety of images, including what might be considered "false positives." These maybe images that would maybe have certain features that might trick the classifier into returning the wrong result. Training with enough of these can create a greater degree of accuracy and make for a more useful tool.

There are two types of machine learning techniques. One is known as supervised learning and the other is unsupervised learning (also known as reinforcement learning). Supervised learning takes input/output pairs as training data and makes new predictions based off that data. Reinforcement learning takes a different approach. It draws inferences from data that is unlabeled. Algorithms that are unsupervised will attempt to make the right decisions and will periodically be rewarded for making the right ones.

Supervised Learning

Supervised learning finds patterns and develops models that are used to make new predictions. Supervised learning can fall under two categories: classification and regression.

Classification is exactly what it sounds like. It uses a predictive model to assign a label to a new input (Brownlee, 2017). For instance, determining whether a user might find a movie enjoyable, or not. Or perhaps, whether or not a face is in the frame of a camera. This is often used to simply label an input. This is the method that I used for my project at Intel. Examples of this type of algorithm include Naïve Bayes, Support Vector Machines, and Discriminant Analysis.

Regression is similar to classification but takes things a bit further. Regression attempts to predict a continuous quantity (Brownlee, 2017. It maps a function from its input (x) to its output (y) which is continuous. This is usually used for specific numbers. Models are trained similarly to classification algorithms. However, the training data is interpreted as "x and y" pairs. New inputs are predicted to have some coordinate on the graph. Instead of predicting a "yes or no" type of label for a new input, a regression algorithm will predict a continuous value. This is used usually for some type of monetary prediction such as how much a house may sell for or what the price of a stock may be at a given time. Examples of regression algorithms include linear regression and nonlinear regression (Brownlee, 2017).

Reinforcement Learning

Reinforcement Learning (unsupervised learning) focuses on gaining the most reward possible in order to learn (Shaikh, 2017). The basic model behind reinforcement learning is known as the Markov Decision process. There are a set of states, S and a set of actions for the agent, A. As the agent moves from state to state, there is usually a reward or punishment associated with the transition. Instead of using input/output pairs, reinforcement learning attempts to learn as it goes. The concept lies behind exploitation and exploration. Exploration is attempting new transitions in order to learn and/or make progress and exploitation uses previously obtained knowledge to make decisions. The efficiency of a Reinforcement Learning algorithm depends on the problem as well as the tradeoff between exploration and exploitation (Shaikh, 2017).

The best applications of Reinforcement Learning can be seen in automated game playing. The best example is Super Mario or Pac Man. The all of the different states that Mario can be in would be S and all of the possible moves he can make would be the set A. Mario himself would be the agent. The ultimate goal is to reach Princess Peach's castle. If you are familiar with the game you know that there are many obstacles and enemies that inhibit Mario's progress. An algorithm may try to get rush Mario through the level gaining more experience after hitting each obstacle by being rewarded and punished accordingly. Some examples of Reinforcement learning algorithms include Q-Learning, Sparse-Sampling and Trajectory Trees.

Reasoning for Supervised Classification

The type of machine learning that I chose to use was supervised classification. In this situation, our problem was fairly straight forward. A piece of text was either "relevant" or "irrelevant." All that was necessary was creating training data with the appropriate labels and choosing an appropriate algorithm. Three different algorithms were utilized for this project: Naïve Bayes, Max Entropy and Decision Tree.

Naïve Bayes

The Naïve Bayes algorithm is much simpler than other algorithms, this project attempted to implement. Despite its simplicity, it is known to often outperform other more complex algorithms. Because of its simplicity to understand, the classifier for this algorithm becomes easier to code as well. The algorithm is based on what is known as Bayes' theorem which describes the likelihood of something based on previously obtained knowledge. Specifically, it says that the probability of an event, Y given X is equal to the probability of the even X given Y multiplied by the probability of X upon probability of Y. The equation itself looks like this: $P(X|Y) = P(Y|X) * P(X)/P(Y)$ (Ray, 2017).

The equation attempts to calculate what is known as a Posterior Probability, $P(X|Y)$. This is done by multiplying the likelihood of an event, $P(X|Y)$ by what is known as the Class Prior Probability, $P(X)$ and then divided by the Predictor Prior Probability, $P(Y)$. X is what can be called the proposition and Y is called the evidence (Ray, 2017). Without example, all of this sounds like gibberish. For that reason, let's look at the example of whether a particular day is good to play frisbee. The idea here is not too complex. Two versions of the Posterior Probability must be calculated since a particular day can only be "good" or "bad." We can take a look at all of the possible factors. Things like wind and temperature can be factors that affect our decision. In order to calculate our values, we need to create likelihood tables for each of the factors (wind, temperature) and calculate a Posterior Probability for each of the outcomes (good or bad). Let's say that our classifier had to label a new input with the values for wind and temperate being "mild" and "low." The classifier would need to compare two likelihoods. Whichever of the two (good day or bad day) has a higher likelihood, that is the label that the classifier would assign to the new input. In order to calculate the likelihood of a "good day" for this input, multiply the probability of the temperature being low when it's a good day by the windiness being mild when it's a good day [P(Temp=Low | Good Day)* P(Wind=mild | Good Day)]. Calculating the likelihood for the input on a bad day is a similar process [P(Temp=Low | Bad Day)* P(Wind=mild | Bad Day)]. After the two values are computed and normalized they can be compared. Whichever value is larger ends up being the label for the input.

In the case of text classification, the classifier depends heavily on individual words. Every time a new input is passed in, the unique words will be compared with a table created using the training data. If the input contains a word that is not in the training data, then a very small number is generated to represent its probability. The overall idea here is the same as the one described above. The unique words in the input are found in the table, their probabilities are multiplied together and then the values are normalized. Whichever label ends up having the largest value becomes the one assigned to the input (Sarkar, 2014).

Decision Tree

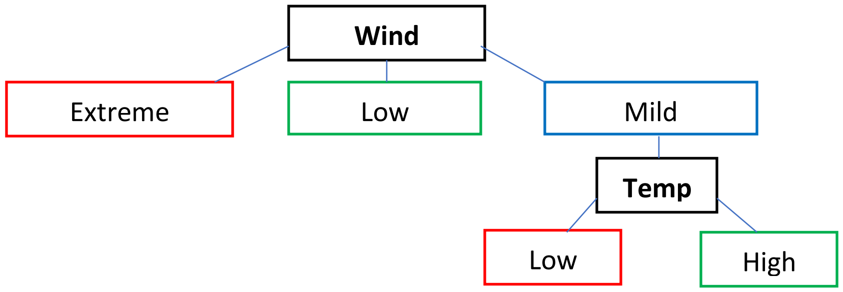
The Decision Tree algorithm is very different from the Naive Bayes algorithm. The code for the algorithm can be more intimidating and difficult to write. However, from an abstract view it is much easier to understand. It follows what is known as a "divide and conquer" strategy. In order to classify an input, Decision Tree will make a series of decisions in order to come to a conclusion. The learning for this particular algorithm takes place when it tries for infer rules from the training data's features. This process results in the creation of a decision tree (Analytics Vidhya, 2017).

When training data is used to create a classifier, a recursive process takes place in order to create the tree. The algorithm will look through the data and attempt to create what are known as "pure" subsets. Pure (or homogenous) subsets are sets where all of the values of a feature are the same. If the algorithm, returns a non-pure subset, it will recursively try again. Each time a pure, subset is created, a new node on the tree is formed (Analytics Vidhya, 2017).

Once again, the idea behind this algorithm can still seem unclear without an example. Let's use the previous example of the frisbee. This time, lets expand our training data. The following table describes the training data that the classifier will use. Ideally, a we would have thousands of entries in order to train, but for the sake of brevity, 5 should suffice. For our label, we have 3 for no and 2 for yes.

Wind	**Temperature**	**Play**
Mild	Low	No
Low	High	Yes
Extreme	Low	No
Mild	High	Yes
High	High	No

If we take the wind feature and break it down, we would result in three different branches. There would be one for mild, low, and extreme. The low branch would create a pure subset because all of the entries where the wind is mild (just one) have the same label (Yes). The same goes for the wind being extreme but this would result in a no. If a pure subset is not created, such as with wind equaling mild, there would be a split on temperature. If the temperature is low, then we would have a pure subset of just one entry (No). If the temperature is high, then we would also have a pure subset with just one entry (Yes). This process would repeat until all of the data in the training set is sorted. When new input is passed into the classifier, the process of labeling begins. The tree created with the training data provided above, would look something like this:



This diagram starts at the top with wind. The leaf-nodes in the tree represent pure subsets. If they are outlined in red, the label is "No" and if they are outlined in green, the label is "Yes." The bolded nodes represent a switch of features. The algorithm sees that the set created is not pure and will attempt to separate entries with a new feature and attempt to create a pure subset. This process keeps recursively occurring until the path of the tree contains all pure subsets.

Now that we have our tree, let's look at an entry with the following parameters: Wind=Low and Temperature=High. We would start off at the top of the tree and work our way down. Because we only have two parameters and our training set was fairly small this process becomes much easier. Out temperature was low and if we follow our tree down, we land at a green node, meaning that the classifier predicts our inputted data to be a good day to play frisbee. If we take a slightly more complex example (Wind=Mild, Temp=Low), then we would travel down the right-most path on our tree and come the prediction that it is not a good day to play frisbee.

Not only does decision tree keep track of the path leading to its “Yes” and “No”, outputs but it also needs to keep track of the ratio between the two types of entries in each node (positive and negative). This ratio is kept track of to measure the confidence of a prediction at every node that the classification algorithm tries to stop on. This ratio is also used in determining which attribute to split on. The algorithm will try to use entropy to decide which attribute to split on. A higher entropy, usually results in a higher number of pure subsets. And because pure subsets have higher confidence, than non-pure subsets, they are sought after when trying to create an efficient tree. Algorithms will try to maximize information gain when deciding with feature to split on (Analytics Vidya, 2017)

In the case of text classification, decision tree will attempt to use the existence and nonexistence of words or number of appearances of words. In the same way that the example of above used the value of features, a standard text classification system could use words and their existence and/or count to create its pure subsets (nodes).

Max Entropy

Max entropy, is used for a wide-variety of cases but is most commonly used for Natural Language Processing. Some examples of where the algorithm can be applied are: topic classification, and language detection. The Max Entropy algorithm is also the most complex out of all the algorithms tested and also takes more time to train. It follows the opposite principle that fuels the Naïve Bayes algorithm. Instead of assuming that all features of a training set are independent, Max Entropy assumes that they are dependent. The algorithm is based on the idea of Maximum Entropy. This principle selects the distribution with the largest entropy (uncertainty of a distribution). Because our training data can be very diverse, in terms of distributions, Max Entropy attempts to look for a uniform distribution which that has a high entropy (API Reference, 2017).

In order to calculate entropy, we must first calculate $\log(1/p_e)$, where e is p_e is the probability of an event. This is what is known as surprise. Or the odds of something unexpected happening instead of what is most probable. If the probability is 0, then the surprise is infinite. But, if the probability is high, then the surprise is a very small number. Entropy is calculated with the following computation: $-\sum p_x \log_2 p_x$ (Vryniotis, 2013).

The goal behind the algorithm is to minimize commitment while maximizing our entropy. The algorithm will try to select something that is consistent with the training data. When the training data is loaded in, the algorithm will attempt to find the empirical probability distribution. This is used to construct the model for which classifications are made. The process of random assignments will begin at this point. The pieces of data in the training set are randomly put into classes which are based on their context. Many models are created and the one with the most entropy is selected. This process is usually computationally expensive and an indicator function will be used to take in new input and determine which class a new input of text is most similar to. This is done through a similar calculation of probability. Words and their frequency are counted and compared to the stochastic model that has been created. The distribution’s highest probability will be the label assigned to the input (Vryniotis, 2013).

Implementation

In order to implement these algorithms and apply the research I had done on them, I decided to use Python as the language I would be coding in. Python has developed a reputation for being very simple to both read and write. It is said that if one can write pseudo-code, they can write in Python. Python also has many libraries that support machine learning. Libraries such as scikit-learn, matplotlib and pandas are just a few examples.

I did not have to write any of these algorithms from scratch since there were already so many implementations. The library that I chose to use is called TextBlob. TextBlob provides an easy-to-use API for Natural Language Processing. It holds capabilities for classification, noun-extraction and sentiment analysis. I, however, only needed classification capabilities. The process of creating a classifier is simple and boils down to only three steps: importing the package, instantiating the object and then train the object. Importing the package is simple after running "pip install TextBlob." At the top of the Python script, one needs to write "from TextBlob import *name of classifier*. After doing so, the classifier object needs to be initialized and trained. This is done by writing something like "with open('data.json', 'r') as fp: naive = NaiveBayesClassifier(fp, format="json")." A file filled with training data is loaded into the classifier object and the object uses that data to make predictions on new data. The training data is in JSON format and has a structure like: {'text', 'label'} (API Reference, 2017).

Gathering Data

The idea behind using classifiers for this project was to differentiate between security articles that were about actual threats, vulnerabilities and exploits and articles that were not but had string matches about "threats", "vulnerabilities", and "exploits." The best places to get data of this nature are RSS feeds. These feeds belonged to numerous security blogs and websites run by larger organizations like the US government as well as smaller companies and groups. These feeds are updated on a regular basis so security researchers monitor these feeds to stay updated on the latest news regarding their software and products. A website known as Packet Storm provided many helpful RSS feeds. Because the scope of my project is fairly small, I limited my data mining efforts to just four feeds.

Once again, I harnessed the power of Python to actually go into the feeds and grab the training data I needed. Having the feeds is one thing, but I needed to go through the feeds and parse for security data. I did this by using a library called feed parser. By creating an array to the links of the feeds and iterating through that array as well as each entry in a feed, I was able to access that data and store it in a file as JSON format. As the loop was iterating, I gave all of the entries in the file the same label: "relevant." After I collected my data, I had to go in and manually assign labels and make sure that there were no mistakes. I also issued a quota of having 100 pieces of training data and tried to balance it out with 50 relevant entries and 50 irrelevant entries. This did not happen the first time that I ran my Python data mining script, so I had to find about 20 entries manually. These 20 would qualify as "false positives."

Testing Data

For testing, I used similar feeds to gather data. I gathered about 30 pieces of text data that I manually labeled. Again, I used a 50/50 ratio. About half of my data was deemed "relevant" and half was "irrelevant".

The TextBlob Library makes it very easy. The classifier class has a method called "accuracy" that takes one parameter. The parameter is to pass in the test data. I took the test data that I collected and loaded it into an array of JSON objects. The "accuracy" method will take the inputted data and attempt to classify each of the entries. The classifier will compare the label that it predicted with the label that I assigned it and keep track of how many entries it labeled correctly. The process is similar to grading an exam. The method will then print data on how accurate it's predictions were. This is done simply by dividing the number of correct predictions by the total number of entries in the test data.

Results

I tested all three of the classifiers with the same test data. The Naïve Bayes classifier consistently returned the highest accuracy. The next came Decision Tree followed by Max Entropy. The Naïve Bayes classifier returned with a 100% accuracy for Test 1, 90% accuracy for Test 2 and 100% accuracy for Test 3. Decision Tree got 90% for Test 1 and Test 3 and 100% for Test 2. Max Entropy returned 70% for Tests 1 and 2. It return 60% for Test 3.

Analysis

Naïve Bayes is often noted to be the go to algorithm for text classification problems. In my case, it definitely lived up to its reputation. The Naïve Bayes classifier was the most reliable in my particular situation. This is also the classifier that I had initially predicted would be the most accurate. The algorithm behind this particular classifier, was not overly dependent on the amount of quality training data. For that, reason, Naïve Bayes was able to accurately make predictions on new data even though it makes "naïve" assumptions about the relationships between the features in the model. Decision Tree didn't perform badly by any means but did worse than Naïve Bayes. The algorithm here could possibly catch up to the accuracy of Naïve Bayes or even surpass it, given more training data. However, with a limited data set the accuracy of the Decision Tree algorithm falls short. Max Entropy falls under the same category. It tested the worst out of all three of the algorithms. However, many sources that I came across said that Max Entropy, would take a lot of time and power to train to become an accurate model. Max Entropy is also said to be an impressive classifier and would be able to make classifications work more efferently than Naive Bayes.

Conclusion

Machine Learning has worked its way into a great number of fields. It's applications to software applications is apparent and very easy to implement. Algorithms such as the one's in this paper offer solutions to complex problems that occur commonly in industrial and user applications. In the case of software threat awareness, piles of data and textual information bombards researchers. A lot of this data is relevant and is worth keeping around but a lot can turn out to be useless. Being able to differentiate between what is useful and what isn't is something human beings can be really good at. However, this is a long and tiresome process. Researchers are better able to use their time if a process like this was done automatically while collecting data. machine learning offers the perfect solution to this problem. To develop a small degree of "intelligence," a model can be trained to be able to predict whether a piece of text contains useful security information or not. Even though all three of the algorithms tested are well known and used for text classification, Naïve Bayes delivered the most accurate results considering the limited training set, computing power and time.

Works Cited

"API Reference¶." *API Reference — TextBlob 0.15.0 Documentation*, textblob.readthedocs.io/en/dev/api_reference.html.

Ali, Abder-Rahman. "Deep Learning in Oncology – Applications in Fighting Cancer -." *TechEmergence*, 1 Sept. 2017, www.techemergence.com/deep-learning-in-oncology/.

Brownlee, Jason. "Difference Between Classification and Regression in Machine Learning." *Machine Learning Mastery*, 25 Sept. 2017, machinelearningmastery.com/classification-versus-regression-in-machine-learning/.

Cade, DL. "A Look at How Snapchat's Powerful Facial Recognition Tech Works." *PetaPixel*, 30 June 2016, petapixel.com/2016/06/30/snapchats-powerful-facial-recognition-technology-works/.

Heinz, Sebastian. "A Simple Deep Learning Model for Stock Price Prediction Using TensorFlow." *Medium*, ML Review, 9 Nov. 2017, medium.com/mlreview/a-simple-deep-learning-model-for-stock-price-prediction-using-tensorflow-30505541d877.

Plummer, Libby. "This Is How Netflix's Top-Secret Recommendation System Works." *WIRED*, WIRED UK, 21 Aug. 2017, www.wired.co.uk/article/how-do-netflixs-algorithms-work-machine-learning-helps-to-predict-what-viewers-will-like.

Ray, Sunil, et al. "6 Easy Steps to Learn Naive Bayes Algorithm (with Code in Python)." *Analytics Vidhya*, 13 Sept. 2017, www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/.

Sarkar, S. D., Goswami, S., Agarwal, A., & Aktar, J. (2014). A Novel Feature Selection Technique for Text Classification Using Naïve Bayes. *ISRN Otolaryngology*, 1-10. doi:10.1155/2014/717092

Shaikh, Faizan, et al. "Beginner's Guide to Reinforcement Learning & Its Implementation in Python." *Analytics Vidhya*, 18 Aug. 2017, www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/.

Team, Analytics Vidhya Content, et al. "A Complete Tutorial on Tree Based Modeling from Scratch (in R & Python)." *Analytics Vidhya*, 1 May 2017, www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/.

Vryniotis, Vasilis. "Machine Learning Blog & Software Development News." *Datumbox*, 20 Nov. 2013, blog.datumbox.com/machine-learning-tutorial-the-max-entropy-text-classifier/.