A Machine Learning Approach to Real Time Earthquake Classification for the Southern California Early Response Warning System

Anshul Ramachandran (aramacha@caltech.edu)
Suraj Nair (snair@caltech.edu)
Ashwin Balakrishna (abalakri@caltech.edu)
Peter Kundzicz (pkundzic@caltech.edu)
Irene Wang (siwang@caltech.edu)

CS/EE 145

June 16, 2017

Abstract

The Southern California Early Response Warning System is currently responsible for alerting airports, trains, fire stations, etc. in the case of an incoming earthquake, differentiating seismological signals between those caused by local earthquakes and those caused by noise. The two main disadvantages of the current system are that multiple stations are required to make an earthquake classification, reducing the time preventive measures can be taken in, and more importantly, the current system raises hundreds to thousands of false triggers a day. We attempted to tackle both issues by using a machine learning approach to classify a trigger as either earthquake or noise induced from the signal of a single station, trying to minimize the false positive rate (noise signals classified as earthquakes) while still guaranteeing a low false negative rate (earthquake signals classified as noise). Our final system includes a prefiltering stage which filters out approximately 70% of all noise signals with extremely minimal misclassifications on earthquakes. Signals that pass the prefilter are then passed through three models of different architectures - an ensemble of tree-based models, a fully connected neural network, and a recurrent neural network. The results of these are ensembled and a classification is made on the resulting confidence in earthquake value. We were able to achieve a false positive rate of roughly 0.5% after just one second of waveform post-P-wave onset, a 2x improvement over the current standard (0.96%), while still guaranteeing a low (approx 2%) false negative rate.

1 Background and Motivation

Timely warnings of major earthquakes could provide the time needed to warn citizens or begin evacuation in vulnerable areas before too much damage is done. Although it is currently impossible to reliably predict earthquakes, technology is already in place to measure real-time seismic activity. Several countries like Japan have earthquake early-warning systems in place to enhance public safety, but in the United States no such system has been successful yet on a large scale. In California, this effort began in the south with the TriNet Project. There, Caltech, the California Geological Survey (CGS), and the USGS created a unified seismic system for Southern California. The integration effort expanded to the entire state with the formation of the California Integrated Seismic Network (CISN). Seismic stations exist all along the West Coast that monitor ground shaking intensity in real time, and transmit said information to an overarching system. A map of the stations in Southern California and all across the West Coast is shown in Figure 1. The central associator at Caltech thus receives signals from all stations and is responsible for recognizing and characterizing newly starting earthquakes.

The earlier a seismological signal is identified as an earthquake, the faster protective action such as stopping elevators and trains, shutting down critical processes, and opening doors can be taken. Currently, a real-time early warning system is in place in Southern California. The system can accurately distinguish earthquakes from background noise signals, but this is only possible after the seismic wave has been detected by multiple stations.

This is not ideal since waiting for multiple stations to receive a signal means that time is lost before preventive action is taken. Ideally, we would be able to achieve accurate earthquake classification using data from a single seismological station. The single station classifier works by first waiting for a 'trigger', which is defined as an uptick in the seismological signal, calculating a set of features from the time dependent signal (described in *section 2, Data*), and then making a classification between earthquake triggers and noise based on a set of simple thresholds on these features. Unfortunately, the current system raises hundreds to thousands of false positives a day depending on the station, i.e. claims of earthquakes from a noise-caused trigger. In the four month period between January 1, 2017 and April 30, 2017, a total of 4,066,504 triggers were detected, 39,137 of which passed the existing noise filtering criteria. Of these 39,137 signals classified as earthquakes, only 137 were associated with earthquake-caused events. Therefore, 39,000 of nominally 4,066,367 noise-caused signals were misclassified, for approximately a 0.96% false positive rate. It is hard to tell the current false negative rate because there are larger numbers of seismic signals that are from small earthquakes, magnitude < 3, that the real-time system does not detect.

The aim of the project is to improve the current earthquake early warning system by creating a system that can predict whether a seismological signal is an earthquake both quickly and accurately from a single station.

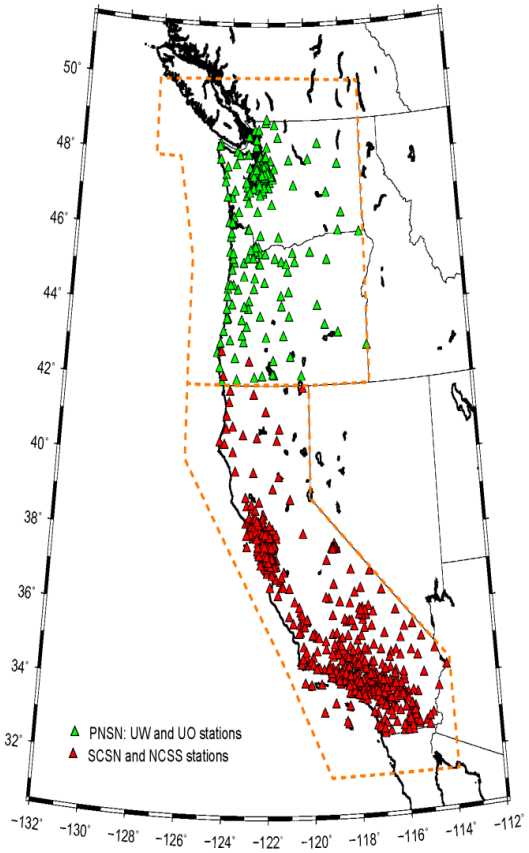


Figure 1: **Map of Seismology Stations Across the West Coast:** These stations collect the real time waveform data and each try and each raise an alarm if it detects an earthquake. The alarms from all stations are used to make the final decision regarding the existence of an earthquake

2 Data

2.1 Raw Data

As discussed in the background section, we consider only segments of the continuous time seismological data that correspond to regions that are classified as triggers. Specifically, a trigger is defined as a point in the signal for which the ratio of high frequency bank amplitude over a short-time range to that of a long-time range is above a certain threshold value. We use this definition for a trigger since we are confident that all P-wave onset signals will have this characteristic. We also this definition as it is part of the signal onset detection algorithm that is used by the real-time ShakeAlert algorithm (Given et al., 2014), and we want to accurately mimic the behavior of the real-time algorithm for which our signal noise classification scheme is designed. Of course, a lot of false noise signals from sources such as cars driving by, cows walking on top, and

weather fluctuations could lead to fluctuations in the seismological signal that also pass this simple threshold. In addition to these noise causes, other signals that we desire to classify as noise include those from regional and teleseismic earthquakes. We are interested in only raising alarms for local earthquakes, so both of these classes of earthquakes are, for this problem, considered as noise. Both would likely, however, have P-wave onsets that satisfy the trigger classification threshold, and may prove to be more problematic to distinguish from local earthquakes than some of the pure noise sources of seismological fluctuations.

The signals are labeled by hand retrospectively. When an earthquake is currently detected, the time the P-wave should have reached each station is calculated and the corresponding seismological signal closest to that time point (with some cap on the allowed difference between predicted and actual onset) is extracted. The same is done for regional and teleseismic earthquakes. Any trigger that is not labeled in this method is considered a noise-caused signal. Since this hand labeling strategy does not catch every earthquake that occurs in the Southern California region, it is possible that some signals labeled as noise-caused are truly earthquake-caused. However, these are likely low magnitude (< 3) earthquakes that we are not as concerned about from an Early Warning perspective, and are therefore alright with those being labeled as noise.

2.2 Calculated Feature Description

For our approach, instead of using the raw waveforms, we make use of meta-information of the waveforms, which are all features that are both believed to have seismological importance by geologists (domain-specific knowledge) and currently calculated in the real-time system (allowing for easier future integration of our approach). A detailed description of each feature is located in the Appendix. The features extracted from the raw seismological signals are time interval dependent, i.e. calculated over a given time range of signal, with the exception of rvar and presig. In total, 27 features were calculated for each possible trigger over the time interval starting at the P-wave onset (designated as the first point in the signal that passed the trigger threshold in place in the current early warning system) and ending at $0.5, 1, 1.5 \ldots, 5$ seconds after the P-wave onset. Therefore, 27 features were calculated for each of 10 increasingly longer waveform time intervals. In addition to these, the time independent features rvar and presig are used. We cumulatively append feature calculations as the time interval increases, resulting in 10 increasing-size sets of feature lists, depending on how much time after the P-wave onset we make the prediction on. For example, models trained on the first 2 seconds after P-wave onset use a total of $4 \times 27 + 2 = 110$ feature values, 27 time-dependent features calculated over each of the 0.5, 1, 1.5, and 2 second intervals, and the two time independent features rvar and presig.

It is generally accepted that false triggers have different dominant frequency components than earthquakes, and so different feature distributions should be found. A quick exploration of the distribution of values for various features, as seen in Figure 2, shows that earthquake-caused signals and noise-caused signals have points of difference. This is promising for machine learning models, that we can extract information and classification from the feature set that we are using.

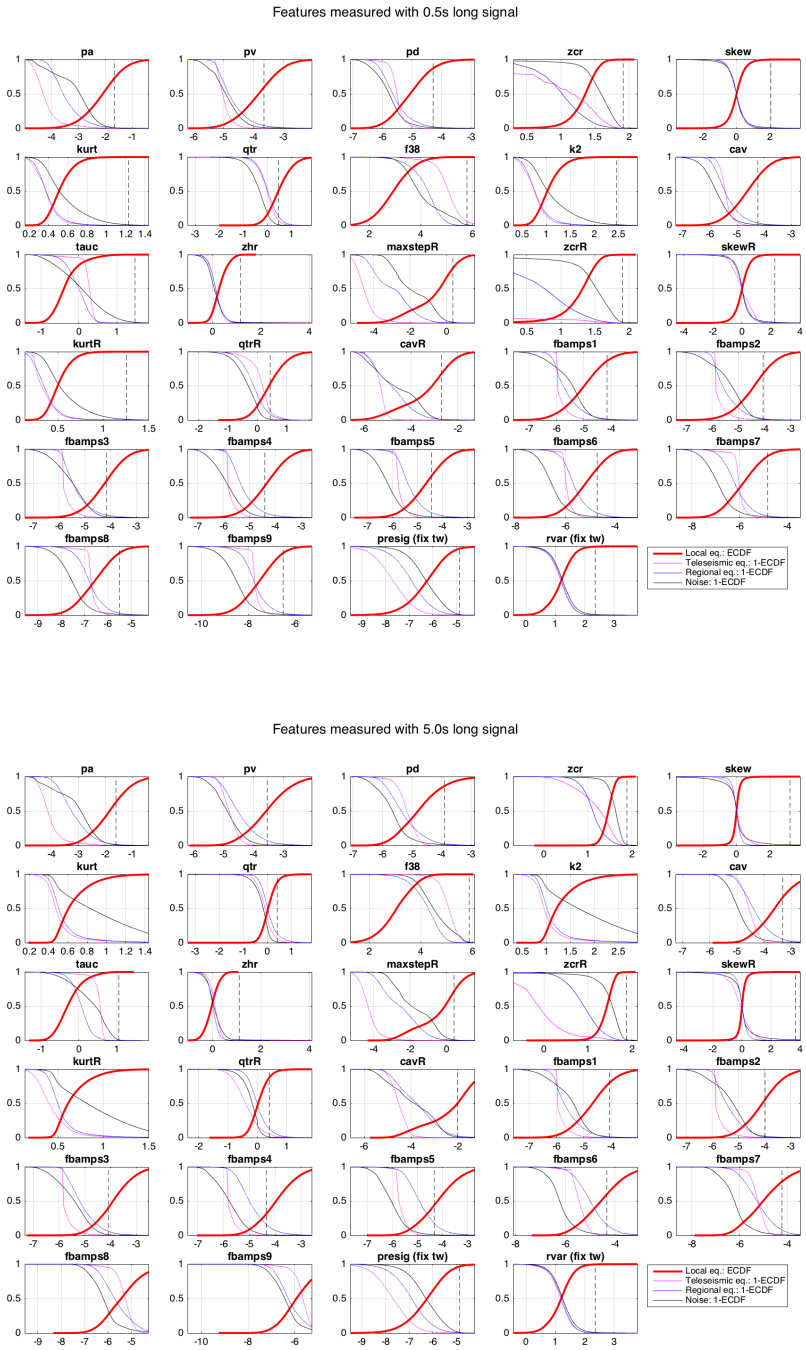


Figure 2: **Example Distributions:** The top figure shows the distributions of features extracted, grouped the four classes of signals if only the first timestep of the waveform (first 0.5 seconds post-uptick) is considered. The bottom figure shows the same feature distributions, except with calculating the features over the first 10 timesteps of the waveform (5 seconds post-uptick). We see in both cases, the earthquake-caused triggers do not necessarily have similar feature distributions as the others, although the differences seem visually to become more pronounced if a larger portion of the waveform is taken into account.

2.3 Final Dataset Specifications

Our final dataset has 435732 signals, 54617 of which are local earthquakes, 7487 are from teleseismic earthquakes, 11519 are from regional earthquakes, and 362109 are from pure noise sources. The collection of each class of data points varies slightly, and is described as following

- **Local earthquake data**: We are using all records with hypocentral distances shorter than 60km and catalog magnitudes >3 from the data set of Meier et al., 2015. This data set combines strong motion data from Japan (time period 1997-2015), broadband and strong motion data from Southern California (time period 2001-2014), as well as records from a global strong motion data compilation (1981-2009).
- **Teleseismic earthquake data**: Teleseisms are earthquakes that occur at large distances from the network, > 1,000km. We have used the Seismic Transfer Program (STP) to download records from all teleseisms with moment magnitudes Mw>=6.0 in the time period 2004-2016, recorded by the Southern California Seismic Network (SCSN).
- **Regional earthquake data**: Regional earthquakes are events that occur outside the seismic network of interest (here SCSN), but at shorter distances than teleseisms, e.g. events in northern Mexico or Nevada. We have downloaded records using STP for all regional earthquakes with Mw>=4 from the time period 2001-2017 recorded by SCSN.
- **Noise data**: We have used the log file data from the real-time ShakeAlert system to download waveforms around all impulsive onsets detected by the real-time system between January 2015 and April 2017 across the SCSN. We have removed all onset detections that occurred when real earthquakes have happened, in order to avoid having real earthquake records in the noise data set.

3 Feature Selection

3.1 Lasso Regression

Although certain features considered of geological importance were already extracted from the seismological signals, we conducted further analysis to determine which features were most critical in distinguishing background noise from earthquakes. Thus Lasso Regression was used to determine which features were of the most importance. Lasso Regression is a generalized linear model that estimates sparse coefficients. The weights corresponding to each feature are encouraged to go to 0, with the speed of approach to 0 governed by the amount of regularization used. Thus, Lasso Regression with a 0.01 regularization penalty was used and the features with nonzero weights are listed in Table 1 below. Thus, we see that all other features are probably of lower importance in determining whether a signal is coming from an earthquake. Furthermore, the weights in Table 1 are correlated with the predictive power of each of the features. We see that only fbamps5 and fbamps6 seem to be important of the filter bank amplitudes. Furthermore, for the features computed on both the raw signals and the high-pass filtered signals (skew, kurt, cav, qtr), we see that neither value of kurt and skew are very predictive while for cav and qtr, once one of the values is known, the other is not that predictive.

3.2 Possible Redundant Features

We eventually decided to remove a few features that we believed to be redundant, using logic from a seismological point of view. These features were skewR, kurtR, cavR, qtrR, tauc, and fbamps9. The first four we removed because we believed that there would be heavy redundancy between skew, kurt, cav, and qtr over the raw and high pass filtered signals, so only one would be necessary. Also, fbamps9 contains peak amplitudes in the frequency band 0.1-0.2Hz. Because causal filters introduce a phase delay that increases with decreasing filter frequency, signal energy at such low frequencies is strongly delayed by such filters. It will not show up in the initial couple of seconds, so there is almost no signal information contained in this

Table 1: **Features with Nonzero Weights with Lasso Regression** R denotes features computed on raw waveforms, while other features are computed on high-pass filtered waveforms. We use regularization penalty = 0.01.

Feature	Weight
maxstepR	0.492
cavR	0.446
presig	0.187
f38	0.180
qtr	0.162
fbamps5	0.161
fbamps6	0.072
k2	0.059
cav	0.044
zcrR	0.002
zcr	0.001

feature. Therefore, all prefiltering and training done in this paper do not involve these six time-dependent features.

4 Prefiltering Noise

The dataset we use consists of features from several signals that exhibited a sufficiently large uptick in their signal value to be considered as possible earthquakes. However, many of these signals can still be easily classified as noise signals without the use of sophisticated models. Thus, the idea behind prefiltering noise is developing a method to quickly identify and remove noise samples with simple models. Therefore these methods could be run on-site before data is sent to the central associated as described in the Individual Architectures Section. Then, signals that are more difficult to classify can be classified using more sophisticated models. Thus, both prefiltering models described below were trained on just the first 0.5 seconds of data so that prefiltering can be performed quickly in real-time. Furthermore, for both methods, a high penalty was placed on false negatives (misclassifying signals labeled as earthquakes) to ensure that the prefiltering methods could remove as many noise signals as possible from the dataset without discarding any earthquake signals.

4.1 Shallow Decision Tree

A shallow decision tree (depth 10) was trained on data from just the first time step (first 0.5s) with a high penalty placed on false negatives as described earlier. This was done by training the decision tree with a heavy weight on correct local earthquake classification. Here, teleseismic and regional signals were treated as noise. This method was used to filter the initial dataset, effectively removing as much noise as possible. The remaining difficult signals were used to train the more sophisticated models described in the Individual Architectures section. In our overall pipeline, the trained prefilter is applied to each signal first before performing classification using more sophisticated models. A visualization of the model accuracy and false negative/positive rates vs. the ratio of the penalty on false negatives to the penalty on false positives is shown in Figure 3.

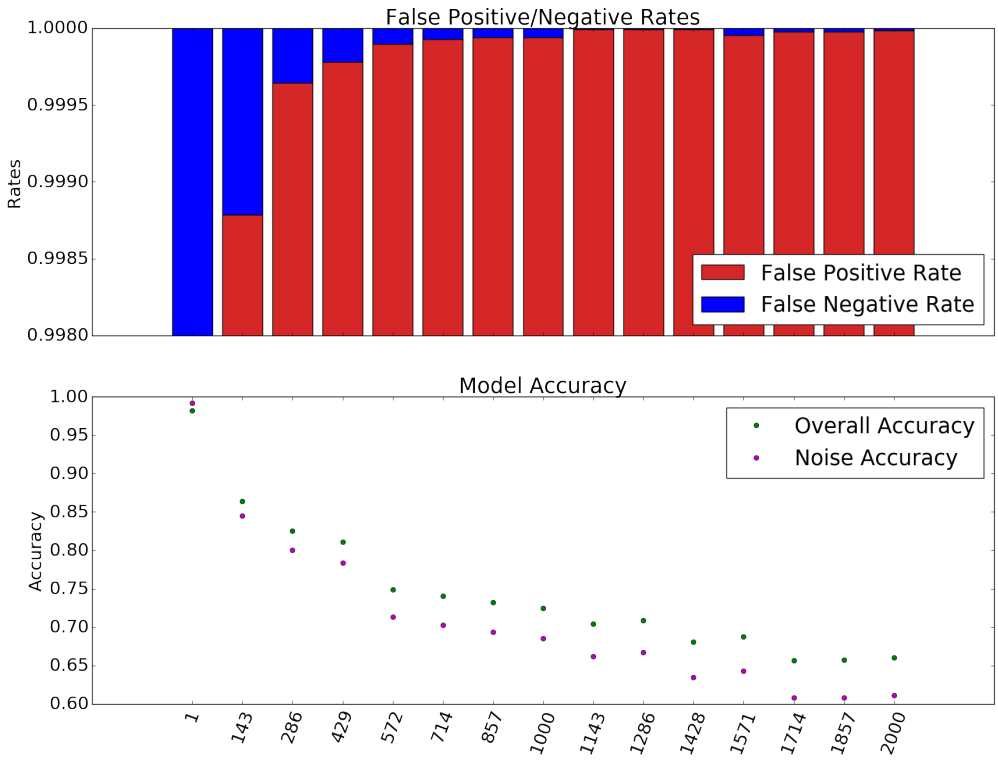


Figure 3: **Pre-Filtering Performance:** Several different shallow decision trees were trained with the same maximum depth of 10. Different class weights were tested in order to see how much noise could be removed without misclassifying a true local earthquake.

4.2 Perceptron Covering Algorithm

In this method, we took the 23 features in the first time step (21 time dependent features, 2 time independent features) and train a linear perceptron on every combination of n features (total of $\binom{23}{n}$ perceptrons) that weights classifying an earthquake signal incorrectly orders of magnitude more than classifying a noise signal incorrectly by a factor of W. We then find the set of k such perceptrons that maximize the number of noise signals that are labeled as such by at least one of the k perceptrons, along the lines of a union among the perceptrons (also why we call this a perceptron covering). We were attracted to using perceptrons as such since they are very computationally inexpensive yet still give more freedom than decision trees which use hard thresholds on individual features (no sense of covariance among features is taken into account). We show the dependence of percent noise detected as well as the number of misclassified earthquake samples thrown out on k and n by using $W = 10000$ and $k \in [1, 7], n \in [2, 3]$ in Figure 4. We also see the dependence of these quantities on k and W for $n = 3$ and $k \in [1, 7], W \in [10^4, 10^5, 10^6]$ in Figure 5. Discussions of the results are in the captions. At the end, however, given the promising results of the shallow decision tree and the ease to streamline the decision tree method into our automated training pipeline, we decided to use that method for prefiltering out noise.

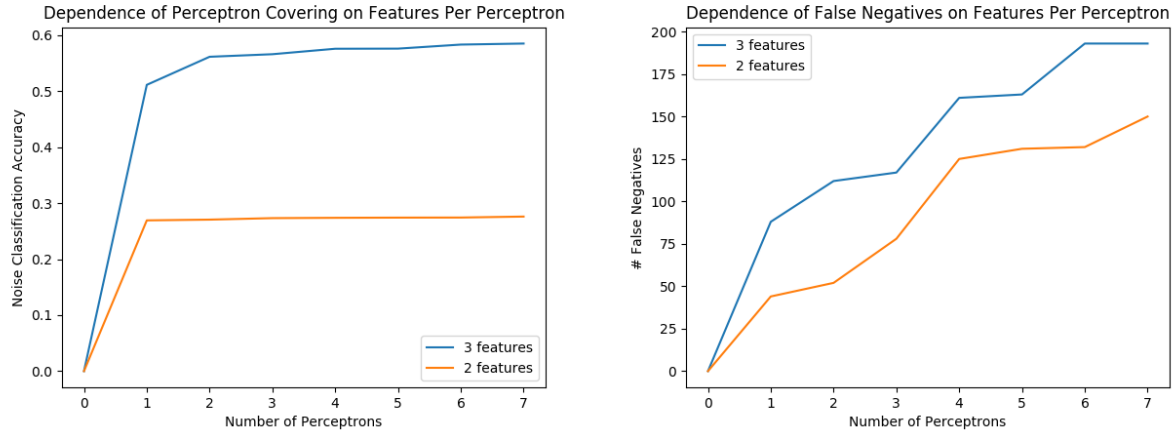


Figure 4: **Varying Number of Features per Perceptron:** We see on the left that when we went from 2D to 3D perceptrons, we see a vast increase in the percent of noise samples that we can correctly identify (almost to 60%). However, as shown on the right, the number of real earthquake signals that are thrown out also increases, although not as drastically. This makes some sense because perceptrons using more features can fit more complex partitioning surfaces, but as the potential to get many more noise samples correct increases, the penalty on a false negative may not be high enough to prevent additional earthquakes from being misclassified.

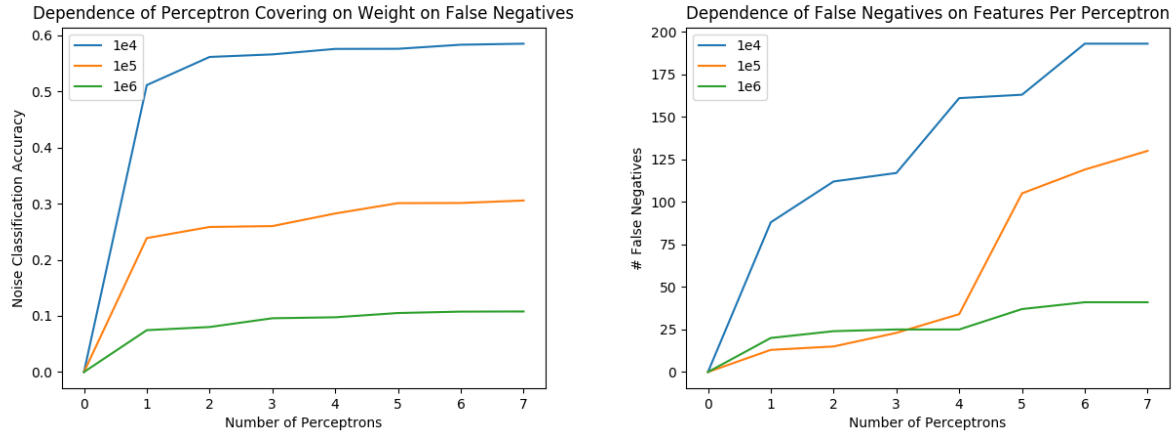


Figure 5: **Varying Weight of False Negatives:** As expected, the smaller the weight we place on getting earthquake-caused signals correct, the more noise-caused signals we are able to properly classify, as shown on the left. However, as shown on right, this equivalently means that as we decrease the weight on getting earthquake-caused signals correct, we increase the number of earthquake-caused signals that we misclassify, also as expected.

5 Individual Architectures

A combination of each of the individual architectures was run on the data and the results determined the classification of the signals.

5.1 Tree Ensemble

5.1.1 Description

When used for classification problems, tree based models generally work by finding splits in features that minimize some measure of impurity (gini impurity was used in this paper) in the resulting data partitions. Thus, the goal is to find splits in feature space that can separate differently labeled data as much as possible. Since tree based models have relatively low training time, they can also be used in ensemble based models to powerful effect. Particularly, with tree based models such as random forests and decision trees, a multilayer model can be constructed as follows. The training data can be split into two portions, one of which is used for training the tree based models on the bottom layer. Then, the trained models can be used to obtain classification outputs on the other portion. Then, the classifications of each of the bottom layer models can be used as training data for a top layer model, which uses the classification results of each of the bottom layer models and the tree label to learn the correct way to ensemble their outputs to obtain the best classification.

5.1.2 Architecture

Two main architectures were experimented with for the Tree Ensemble Models, both of which were multilayer models as described above. All models were implemented using the Python package Scikit-learn. The first architecture consisted of 2 random forest models, a decision tree, a bagging classifier, and an Adaboost classifier on the bottom layer with a decision tree on the top layer. The random forests, bagging classifier, and Adaboost classifier are so called 'meta' estimators, and thus aggregate the results of a variety of smaller models (base estimators), in this case decision trees. Thus, for the random forest and bagging classifier, hyperparameters corresponding to the number of base estimators, the maximum depth of these estimators, and the maximum number of samples used to train each of these base estimators were tuned to prevent overfitting. For the decision tree classifiers, hyperparameters corresponding to the maximum depth of the tree and the maximum amount of features split on were also tuned to prevent overfitting. Hyperparameter tuning for each model was performed by varying the hyperparameters systematically until the in sample error and out of sample error for that model on a variety of different training sets and testing sets were relatively close, making overfitting unlikely. The issue with the first architecture however was that the top layer decision tree model could only output a binary classification whether a signal corresponded to an earthquake rather than a confidence that the signal was an earthquake. A confidence score is much more useful for distinguishing signals that are clearly earthquakes and signals that are on the fence, so the decision tree model on the top layer was replaced with a Logistic Regression model, which can output the probability that a given signal corresponds to an earthquake. Thus, the final architecture for the Tree Ensemble model is described in Figure 6.

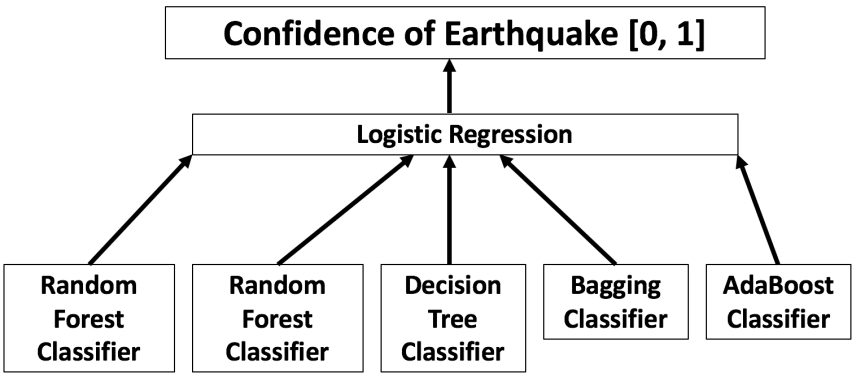


Figure 6: **Tree Ensemble Architecture:** Ensemble of several tree based models with a Logistic Regression Layer on top.

5.1.3 Results

In (Figure 7) we see the precision-recall curves for the fully connected model on an out of sample dataset. we observe that the are under the curve ranges between .90 and .98. However, we see effectively no increase as the number of time steps increases, which is counter intuitive. The cause for this is likely over-fitting, which can be remedied by placing more constraints on the complexity of the trees.

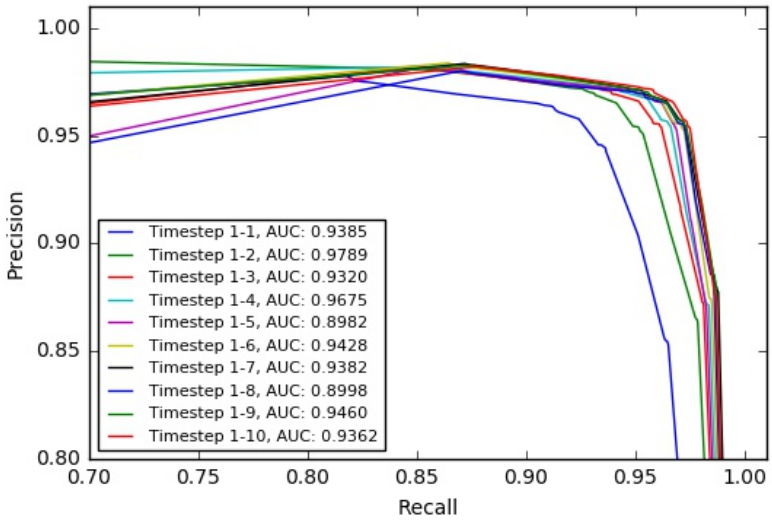


Figure 7: **Tree Ensemble Precision Recall:** The Precision Recall curves of the tree ensemble model at each time-step. We also see the AUC (area under the curve) for each PR curve.

5.2 Fully Connected Neural Networks

5.2.1 Description

Neural networks are used to learn an approximation for the unknown, underlying function that maps a set of input vectors to a set of corresponding output vectors. For classification problems such as the Early Warning Response System, the output vector is a one-hot encoding of the class (an all-zero vector with length equal to the number of classes with a 1 only at the index that corresponds to the input vector's class). A neural network approximates the arbitrary underlying function through a series of linear combinations and nonlinear transforms. Values propagate through a series of layers, $l_0, \ldots, l_n$, with l_0 the input layer and l_n the output layer (we therefore have $n-1$ intermediate, or 'hidden' layers between input and output). The layers have size $k_0, \ldots, k_n$ respectively. There is an associated $w_{i,j',j}$ weight value between the j'th node in l_{i-1} and the jth node in l_i. These weights are what are 'learned' for neural networks. The propagated value for the jth node in l_i is given as a nonlinear transform (such as rectified linear unit, ReLU, or hyperbolic tangent, tanh) applied to the linear combination $\sum_{j'=0}^{k_{i-1}} w_{i,j',j} v_{i-1,j'}$.We call these fully connected layers because each node in l_i is a linear transform of all k_{i-1} nodes in l_{i-1}. To prevent issues of overfitting, we use dropout at each fully connected layer, where we randomly drop a small subset of nodes. The weights are learned usually by some form of gradient descent and backpropagation on a loss function. The loss function used for classification problems is usually some form of categorical cross entropy. Essentially, we consider how well the output vector calculated by the neural network for a certain input vector compares to the true output vector, measured via the loss generated between this true and predicted vector. We then shift the weights to get the predicted output vector closer to the true output vector, by moving along gradient values, like a normal optimization problem.

5.2.2 Architecture

Given the scale of data, all neural network architectures tested had $O(10^4)$ weights as any more would lead to worries of overfitting on the dataset that we used. This meant that we could explore architectures with maximum two hidden layers of sizes $O(100)$ (the input layer also has $O(100)$ nodes, actual size dependent on number of time steps used). The actual architectures tested had hidden layers of sizes 128-64, 128-128, 256-128, and 256-256 (with the first number being the size of the hidden layer following the input layer and the second number being the size of the hidden layer preceding the output layer). Each hidden layer used a ReLU activation and 20% dropout was applied to the outputs of both hidden layers before passing the value to the next layer. The output layer (size 2) used a softmax activation to bound the values in the interval $[0, 1]$, paralleling confidence values in the class assignment. The architecture is shown visually in Figure 8. Tflearn, a lightweight wrapper to Tensorflow was used to set up and train all neural network architectures.

We used a weighted categorical cross-entropy loss function, weighting false negative errors (predicting an earthquake-caused trigger as noise-caused) much more highly than false positive errors (predicting a noise-caused trigger as earthquake-caused). This is because, while we are trying to minimize false positives, the first strict requirement is to maximize the detection of earthquake-caused triggers. Generally, if the predicted output vector is $(p_1, \dots p_n)$ and the true output vector is $(q_1, \dots q_n)$, with weights given on categories given by $(W_1, \dots W_n)$, the weighted categorical cross entropy is defined as

$$L = -\sum_{i=1}^{n} W_i q_i \ln p_i$$

We are guaranteed that this is positive since all $p_i \in (0, 1]$ due to the softmax activation on the output layer (we guarantee ≥ 0 by adding an ϵ to any p_i that equals zero). For our particular problem, we chose a weight vector of $(W_1, W_2) = (1, 100)$, essentially weighting classifying an earthquake signal incorrectly 100 times worse than classifying a non-earthquake signal incorrectly. A learning rate of 0.001 was used with an Adam optimizer. An 80-20 train-test split was used for training.

5.2.3 Results

The architecture with both hidden layers having size 256 performed the best, so we used this architecture in the pipeline.

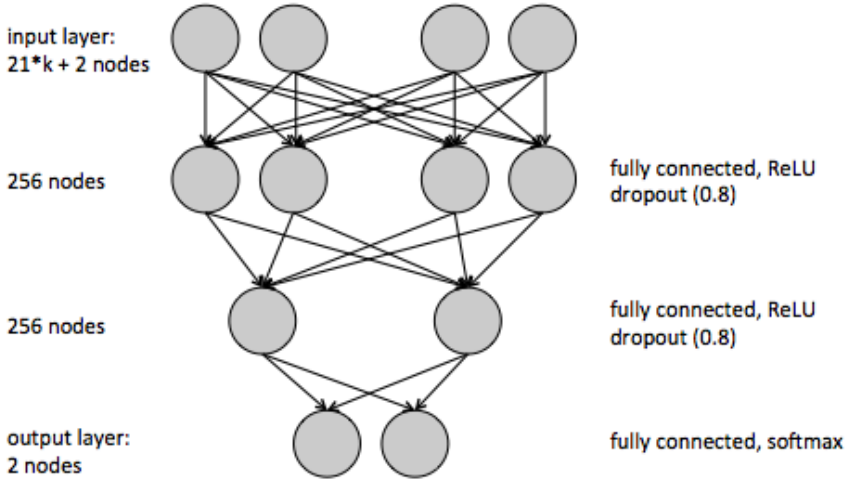


Figure 8: **Neural Network Architecture:** Architecture of fully connected neural network .

In (Figure 9) we see the precision-recall curves for the fully connected model on an out of sample dataset. we observe that the are under the curve ranges between .97 and .99 with generally better performance on later time steps.

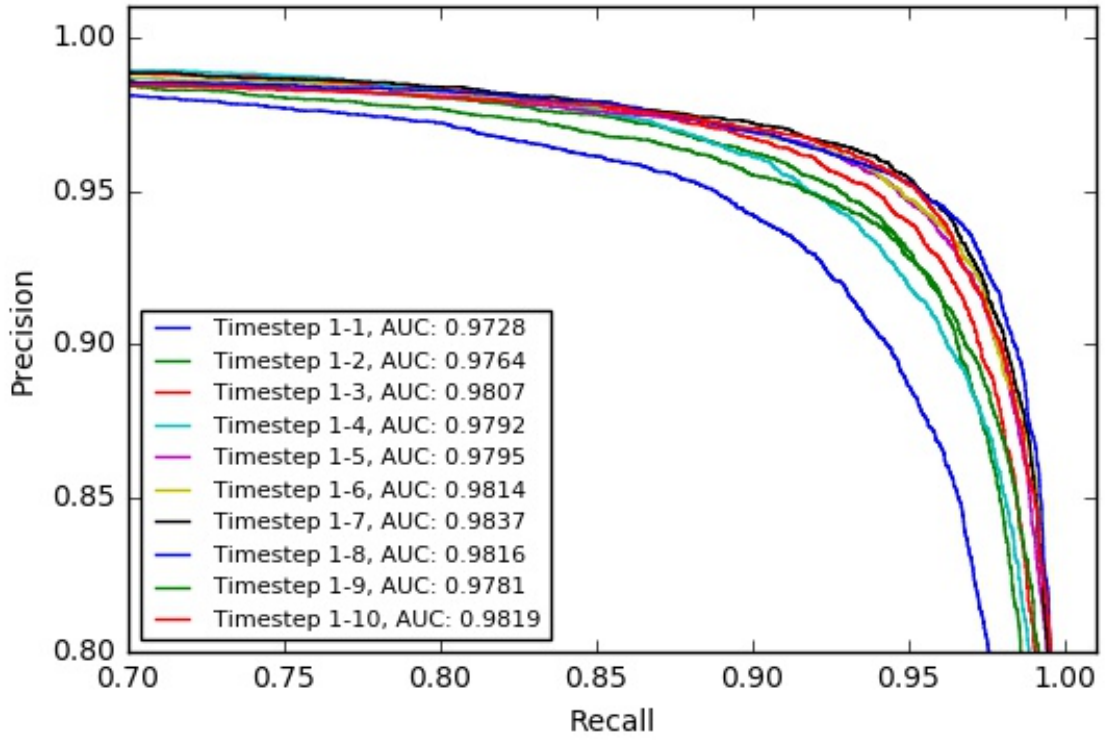


Figure 9: **Fully Connected Neural Network Precision-Recall:** The Precision Recall curves of the fully connected model at each time-step. The Area Under Curve (AUC) is a good metric to asses these curves, with 1.0 being the best possible value.

5.3 Recurrent Neural Networks

5.3.1 Description

The Recurrent Neural Networks are similar to the Fully Connected Neural Networks in that they learn hidden representations of the input data, ultimately outputting a probability of an earthquake. The primary difference with the RNN is that instead of looking at all 23 features at each time step at once it looks at them one time step at the time, with the previous values having some weighting on the representation of the current values.

Recurrent Neural Networks (RNNs) work by having a node's value depend on it's previous value. Thus, they are ideal for working with sequential or time dependent information. In this case, the RNN is implemented in TFlearn, which is a modular deep learning library built on top of TensorFlow. The model predicts probability of earthquake, and minimizes categorical cross entropy loss using an Adam optimizer (adaptive moment optimizer).

Specifically, in this case the recurrent component of the network is a GRU cell. For input vector x_t, output vector h_t and $\circ$ representing the Hadamard product, the GRU cell is defined as

$$s_t = z_t \circ s_{t-1} + (1 - z_t) \circ \sigma_h(W_s x_t + U_s(r_t \circ s_{t-1}) + b_s)$$

where

$$z_t = \sigma_g(W_z x_t + U_z s_{t-1} + b_z)$$

is called the update gate vector,

$$r_t = \sigma_g(W_r x_t + U_r s_{t-1} + b_r)$$

is called the reset gate vector, σ_g is the sigmoid function and σ_h is the tanh function. W, U, and b are parameters that are learned. See (Figure 10).

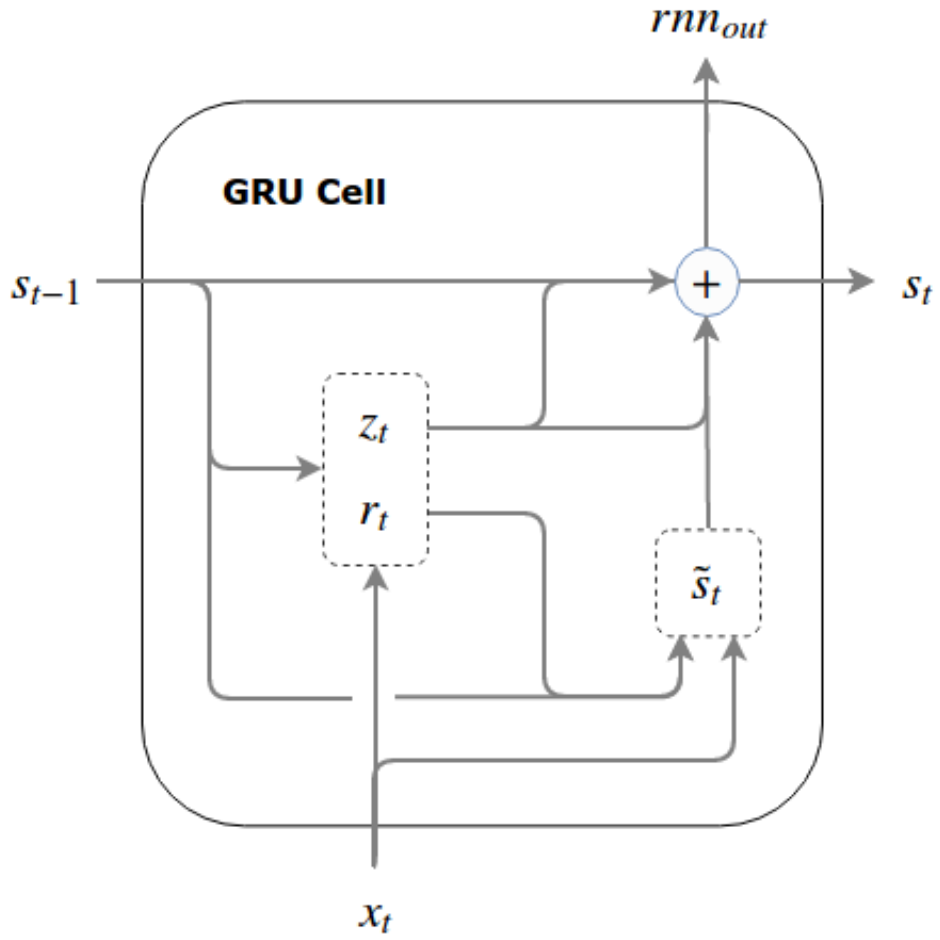


Figure 10: **Gated Recurrent Unit Architecture:** The structure of a single GRU cell in the recurrent layer. Using the gate vectors, the GRU cell is able to maintain "memory" of previous time-steps while still being able to optimized using gradient descent.

After the GRU cell is applied to the features over each time step, the resulting output is fed through two fully connected layers before generating a final prediction.

5.3.2 Architecture

The first component of the RNN is the GRU cell. Specifically, it takes all 23 features at each time step sequentially, and generates a single vector of length 256. This is followed by a fully connected layer with 512 nodes and ReLU activation, which is followed by the output layer containing probability of an earthquake using a softmax activation. See (Figure 11).

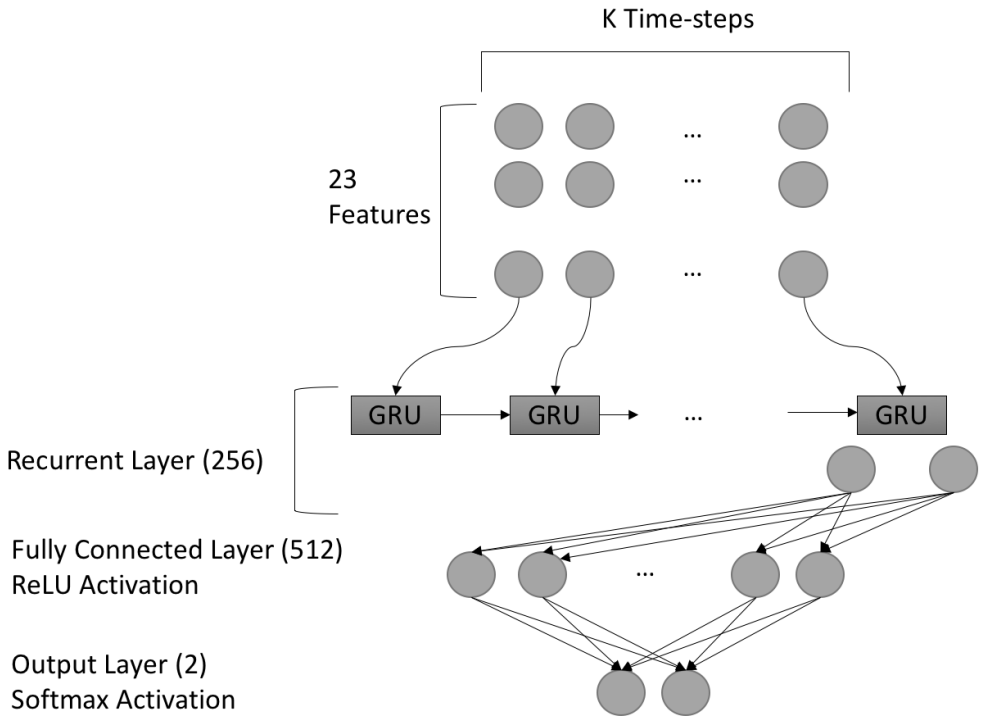


Figure 11: **Recurrent Neural Network Architecture:** Architecture of recurrent neural network. The set of features at each time step is fed into the recurrent layer, and keeping a memory of the previous layers, the recurrent layer outputs a size 256 vector. After a size 512 fully connected layer is applied, the probabilities of an earthquake vs noise are outputted.

In development, GRU cells with output nodes of 128 and 256 were both teseted, as well as fully connected layers of 512 nodes or two fully connected layers of 256 nodes. Ultimately, the GRU cell with 256 nodes and a single fully connected layer of 512 nodes was found to work best.

The loss function for the RNN is the standard categorical cross entropy loss, defined as

$$L = -\sum_{i=1}^{n} q_i \ln p_i$$

where the predicted output vector is $(p_1, \dots p_n)$ and the true output vector is $(q_1, \dots q_n)$.

5.3.3 Results

We assess the performance of just the Recurrent Neural Network. We look at the precision-recall curves at each time- step. As we expect, using more timestep, the performance increases, but even with very few timesteps the precision and recall are high.

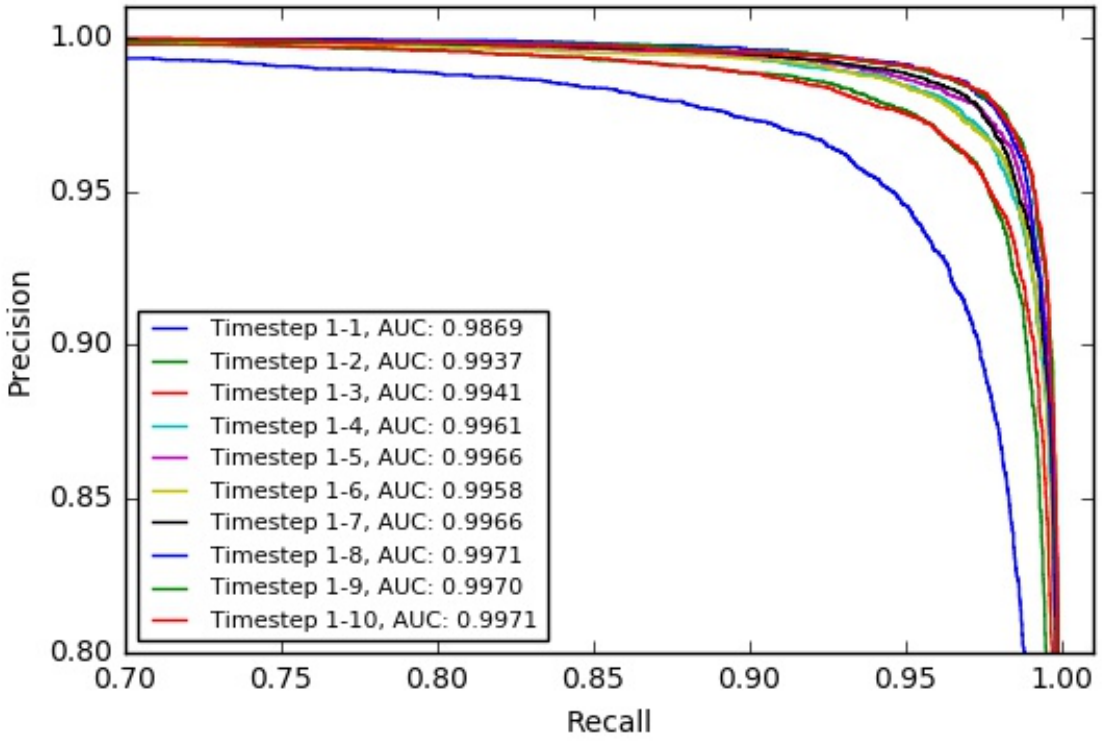


Figure 12: **Recurrent Neural Network Precision-Recall:** The Precision Recall curves of the RNN model at each time-step.

6 Overall Pipeline

To incorporate models into the early response system, we developed a pipeline that handles live streams of data from a station and predicts whether or not each received signal is an earthquake using a combination of the results from the Tree Ensemble, Fully Connected, and RNN Models described in the Individual Architectures Section. This is done by taking the mean of the confidence that each of the three models outputs that a signal is an earthquake. If the mean confidence is more than a certain threshold, then an earthquake is predicted and the pipeline pushes alarms to the central associator.

6.1 Integrated Training and Testing

One important component of the pipeline is the ability to train all of the models on the same data and test the full pipeline. The pipeline includes a benchmarking section which allows exactly this, training each model at each time step on a dataset. It also allows for a realistic test of the full pipeline behavior on a completely out of sample dataset, and generates plots and statistics to assess the performance of the real-time system.

6.2 Integration with Realtime System

Once all models have been trained, the pipeline is ready to run real-time. The pipeline contains a REST API built using the python Flask package. This creates a REST endpoint which when passed a single data point, returns the predicted probability of an earthquake. What this endpoint is actually doing is first passing the

data point through the pre-filter, and if the pre-filter does not throw it out as a simple case of noise, it is passed through all three models, and the mean and median confidence of the three models is returned. See (Figure 13).

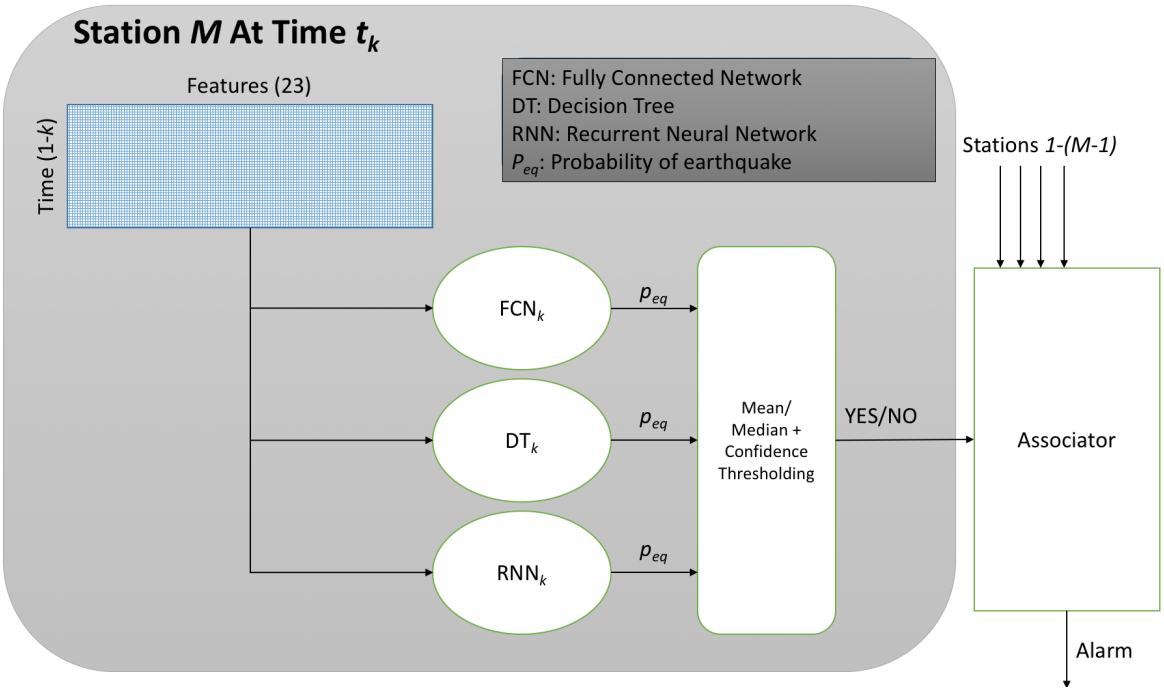


Figure 13: **Pipeline Integration:** How our pipeline will integrate into the real time system. This process would be run at each station, using the constantly recomputed features as more time passes.

The current system detects upticks in the real-time waveform data, and if an uptick is detected, features are generated on 0.5 second increments, and based on those features an alarm may be raised. Our pipeline seamlessly integrates into this work flow, where now once the features have been calculated, they simply need to be passed into our API, and with one API call the model confidences are available to raise alarms. With our pipeline running on say a powerful AWS instance, predictions can be generated quickly and easily.

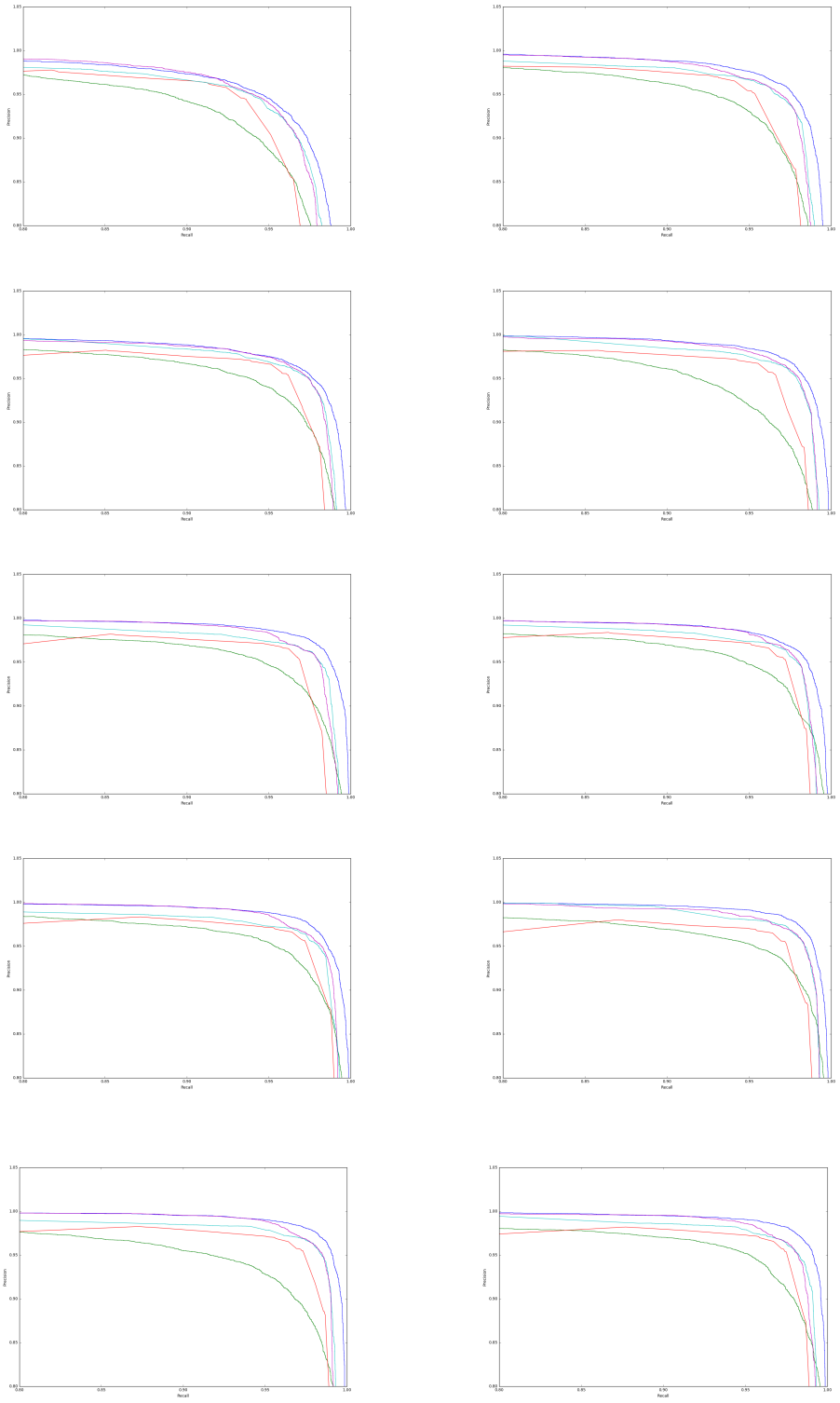
6.3 Ensemble Results

For each time step, the three individual model precision-recall curves as well as the two ensemble model precision-recall curves (taking the mean of the three individual model confidences and taking the median of the three individual model confidences) are shown in Figure 14. To determine the actual confidence value to use as the threshold when making a final classification, we found the confidence for each PR curve that maximized the precision $\times$ recall. This is a decent metric in finding the inflection point in the precision-recall curve, the closest point to the ideal (1,1) point. The true positive, false positive, and false negative rates were calculated at each time step's chosen confidence value for the two ensemble methods. The values are shown in Table 2 for the median ensemble and Table 3 for the mean ensemble. In terms of the PR curves, we generally see the RNN performing the best and the two ensemble methods slightly worse, with the fully connected neural networks and decision tree frameworks worse. This could be due to a number of reasons, from the RNN having larger weight.

We also looked into the true positive, false positive, and false negative rates, and accuracy of the models. We see in Table 2 and Table 3 that for the full system, the false positive rates for both types of ensembling quickly goes to 0.5 % after the first few seconds. Furthermore, the false negative rate and recall (true positive

rate) generally decrease and increase respectively as more data is available as expected. The false negative rate is a little high (around 3 % after 2 s). However we hope that the earthquakes being missed are just low magnitude earthquakes that are not that important to detect. However, this is definitely something we need to look into in more detail in the future.

Furthermore, we plot the accuracy, false positive rate, and false negative rate for each of the models (individual and ensembles) in Figure 15. For the ensemble models, the information in the plots includes the results from the prefiltering as well (which eliminates easy noise examples). Thus, the performance of the ensemble models is more representative of the overall performance. We see that the models all reach classification accuracies of above 94 % pretty consistently, with the mean and median ensemble models achieving classification accuracies of 99 % after about a second. The false negative rate of the models steadily increase as time passes, with the ensemble models achieving false negative rates of close to 2 %. Finally, the false positive rates of the ensemble methods look promising, with a false positive rate of around 0.5 % after a little more than a second.



(a) Time 0-0.5, 0-1.5, 0-2.5, 0-3.5, 0-4.5 s

(b) Time 0-1, 0-2, 0-3, 0-4, 0-5 s

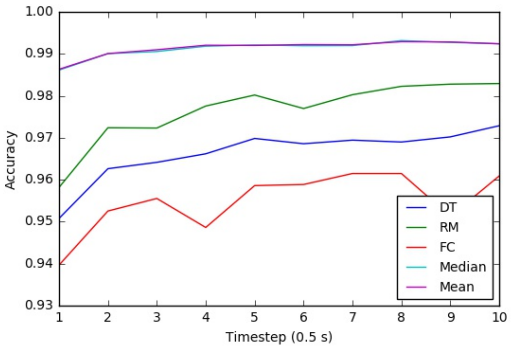
Figure 14: **Precision Recall Over Time:** The precision recall curves were generated with increasing time step size. Individual model and full ensemble performances are plotted on each of the above curves per given time range. Each time step is measured in 0.5 seconds. In the above curves, Blue is the Recurrent Neural Network, Purple is the Mean Ensemble, Light Blue is the Median Ensemble, Green is the Tree Ensemble, and Red is the Fully Connected Network.

Table 2: **Median Ensemble Results:** When using the median ensembling method, we find the confidence threshold that maximizes the precision * recall for each time-step. Using this threshold, we compute the False Positive Rate (Proportion of noise misclassified as earthquakes), False Negative Rate (Proportion of earthquakes misclassified as noise), and True Positive Rate (Proportion of earthquakes correctly classified as earthquakes)

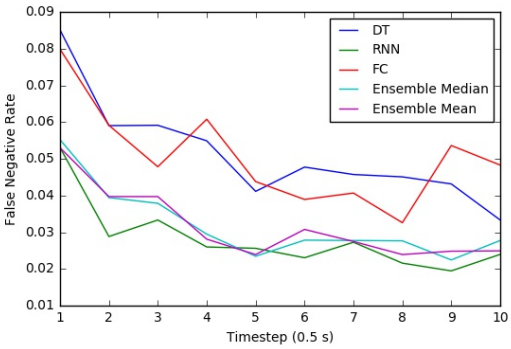
Timestep (s)	True Positive Rate	False Positive Rate	False Negative Rate
0.5	0.945	0.008	0.055
1	0.961	0.006	0.039
1.5	0.962	0.005	0.038
2	0.971	0.005	0.029
2.5	0.977	0.005	0.024
3	0.972	0.006	0.027
3.5	0.972	0.005	0.028
4	0.972	0.005	0.028
4.5	0.978	0.005	0.022
5	0.972	0.005	0.028

Table 3: **Mean Ensemble Results:** When using the mean ensembling method, we find the confidence threshold that maximizes the precision * recall for each time-step. Using this threshold, we compute the False Positive Rate, False Negative Rate, and True Positive Rate

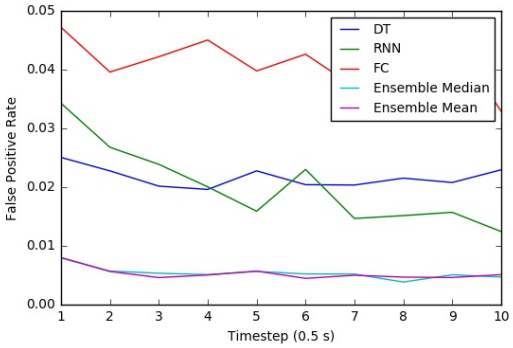
Timestep (s)	True Positive Rate	False Positive Rate	False Negative Rate
0.5	0.947	0.008	0.053
1	0.960	0.005	0.039
1.5	0.960	0.005	0.040
2	0.972	0.005	0.028
2.5	0.976	0.004	0.024
3	0.970	0.005	0.030
3.5	0.972	0.005	0.027
4	0.976	0.005	0.024
4.5	0.975	0.005	0.025
5	0.975	0.005	0.025



(a) Accuracy



(b) False Negative Rate



(c) False Positive Rate

Figure 15: **Ensemble Performance:** Out of sample performance for each of the individual models and the ensembles are plotted as more time steps of features are available. Median and Mean are two different ensembling techniques performed on the individual model outputs. Specifically, the False Positive rate (total proportion of noise incorrectly labeled as earthquake), the False Negative rate (total proportion of earthquakes incorrectly labeled as noise), and the total accuracy are included.

7 Future Work

The main goal that we have is to integrate our classification system into the current Early Warning Response System. This will require us to first insert our endpoint into the realtime feed for us to test our classification system (in terms of both accuracy and latency), but not actually being used in any actual decision making. After we have verified the realtime capabilities of this system, we would then work with the USGS to actually have our system being used to make realtime calls on whether incoming seismological signals are due to local earthquakes.

Besides this practical implementation goal, there are two main groups of future work that we plan on undertaking from a research point of view. One group regards improvements on detecting solely local earthquake signals, which we call the Earthquake versus Noise problem, and the other group of possible work regards possible other questions that can be investigated using this same, data-rich dataset.

7.1 Earthquake vs Noise

In this paper, we only train models on precomputed features of the seismic signals considered of geological importance rather than considering the raw waveforms. Thus, an obvious next step would be to try to learn an embedding of the raw seismological waveforms to use as a feature set. It is definitely possible that there is some other embedding of the waveform that is more conducive to learning the desired classification.

We would also like to try different feature selection methods and investigate their impacts on the final ensemble classification results. We could try removing different sets of possibly redundant features (perhaps subsets of the list of features found to be less useful through the Lasso regression method), or investigate the impact removing features that are computationally intensive to compute). Since we are concerned with speed, it is clearly better if we can minimize the amount of calculation needed to obtain the entire feature set necessary to make a classification.

Also, we would like to explore different ensembling techniques than taking the mean or median of the individual confidences. A simple top layer model that takes as input the results of the individual models could easily outperform the current system.

Finally, we also would like to undertake further exploration and investigation into understanding our models. While decision trees are easy to interpret, it is harder to recognize how neural networks are utilizing the input features. It is definitely harder to trust a system that is unexplainable, so we will try to use flow-based methods to explain what our models are doing.

7.2 Other Problems

Given the high promise that applying machine learning techniques seems to have in the Earthquake versus Noise problem, we are intrigued as to what other seismological questions could be addressed using these techniques. One of these problems is extending our Earthquake versus Noise classifier to a multiclass classifier, where we try to identify the teleseismic and regional earthquakes are their own labels. This is essentially making the noise signals more fine grained. It is quite possible that the seismological waveforms are information rich enough that we could make these finer distinctions. Currently, seismologists have a belief that, for example, teleseismic and regional earthquakes will have similar post-P-wave onset signals as local earthquakes, but it is possible that we could find some finer differentiations that are unseen by human observation or simpler statistical techniques.

Another problem is that of predicting the magnitude of an earthquake from the very beginning of the seismological signal post-P-wave onset. It is currently believed that a higher magnitude earthquake has a similar seismological signal as a lower magnitude earthquake, except for a longer period of time. If we could predict the magnitude (whether binned in a classification setup or pure value in a regression setup) from just a very brief section of seismological signal, then we might discover underlying features that identify a high magnitude earthquakes. This could lead to improvements in our understanding of earthquake mechanics and dynamics.

8 Conclusions

The goal in this project was to use machine learning techniques to improve the California Early Warning System. Thus, we aimed to create a set of relatively simple models which could obtain a better classification accuracy, and particularly lower false positive rate, than the current system. The main problem with the Early Warning System in its current state is the high false positive rate (1 percent), which results in a lot

of irritation as alarms are raised when no earthquake is occurring. Thus, the main goal was to catch as many earthquakes as possible (have a low false negative rate) while ensuring we reduce the amount of false triggers. We managed to achieve a false positive rate of 0.5 percent after about 1.5 s, improving on the current system by a factor of 2. However, we see that the false negative rate is no lower than 2 percent when using data for the first 5 s, indicating that the system does miss some earthquakes. However, we believe these earthquakes may just be low magnitude earthquakes that are relatively insignificant. With more model tuning and other approaches, as described in the Future Work section, we believe that we can achieve a significant improvement upon our current results. However, even in its current state, the system developed constitutes a high-performance, complete pipeline for real-time earthquake classification in which model changes can easily be made.

9 References

Meier, M.A., Heaton, T. and Clinton, J., 2015. The Gutenberg algorithm: Evolutionary Bayesian magnitude estimates for earthquake early warning with a filter bank. Bulletin of the Seismological Society of America, 105(5), pp.2774-2786.

Given, D.D., Cochran, E.S., Heaton, T., Hauksson, E., Hellweg, P., Vidale, J., and Bodin, P. (2014). Technical Implementation Plan for the ShakeAlert Production System: An Earthquake Early Warning System for the West Coast of the U.S., USGS Open File Report 2014-1097.

R2RT, https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

10 Acknowledgements

We would like to thank Men-Andrin Meier, Ph.D., for providing access to extensive data on the statistical characteristics of earthquakes and his invaluable guidance on subsequent classification. We would also like to thank Professor Steven Low, Professor Egill Hauksson, and Professor Yisong Yue for their guidance and support throughout the course and project.

11 Appendix

Descriptions of the features used for real-time earthquake classification are provided in Table 4 below.

Table 4: **Feature Description:** Descriptions of features computed on seismic waveforms after P-wave onset. All features are computed on all 10 time steps for each signal except for presig and rvar, which are only computed once per signal. All features are computed on the signals after high pass filtering except for those denoted by R, which are computed on the raw signals.

Feature	Description
pa, pv, pd	Peak absolute amplitude since P-wave onset for acceleration, velocity, displacement respectively.
fbamps (1-9)	Peak absolute filter bank amplitudes on velocity in 9, octave-wide, filter pass bands between 0.09375 Hz - 48 Hz. Computed on high-pass filtered velocity.
zhr	Peak absolute amplitude on vertical component of signal divided by peak amplitude of vector sum of horizontal components.
zcr, zcrR	Number of zero crossings divided by the signal duration.
skew, skewR	Measures the skewness (lopsidedness, or lack of symmetry) of the signal.
kurt, kurtR	Measures the kurtosis (measures how heavy tailed the data is relative to the normal distribution) of the signal.
cav, cavR	Integrated absolute velocity of the signal.
qtr	Median absolute amplitude in the last quarter of the waveform snippet divided by the mean absolute amplitudes in the first quarter.
maxstepR	Maximum jump between any two neighboring time series samples.
presig	95th percentile of amplitude distribution before P-onset.
tauc	Square root of ratio of integrated squared displacement and integrated squared velocity.
rvar	Ratio of sample variances in the time intervals [0:0.2]s after signal onset and [0.2:0.4]s after signal onset.
f38	Measures the maximum absolute deviation from the mean, relative to the variance.
k2	Sum of squared skewness and kurtosis.