Earthquake Prediction

Henrik Mjaaland
henrikmjaaland@hotmail.com
Institute of Electrical Engineering and Computer Science, University of Stavanger

Andrijana Podhraski
apodhraski@yahoo.com
Institute of Electrical Engineering and Computer Science, University of Stavanger

Inam Alissi
inalissi@gmail.com
Institute of Electrical Engineering and Computer Science, University of Stavanger

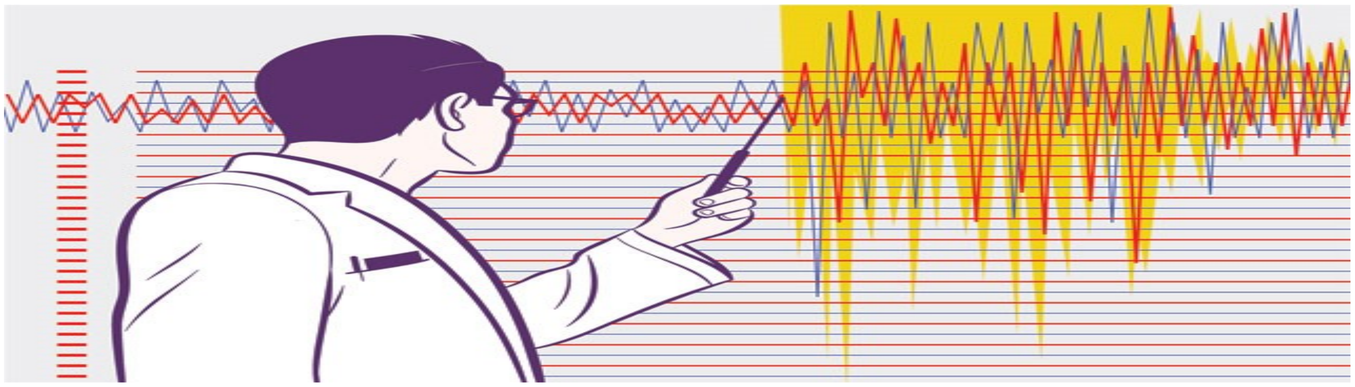


Figure 1: Earthquake Prediction. Source: [http://retrofittingca.com/wp-content/uploads/2017/02/PROOFED-RetrofittingCA-WhyEarthquakesAreSoHardtoPredictJanuary312017-PIC.jpg]

ABSTRACT

The scope of this paper is to submit an abstraction that predicts earthquakes with decent accuracy for the LANL Earthquake Prediction competition which is hosted by Kaggle. The accuracy of our implementation is computed using MSE by Kaggle.

This paper also explains the theory behind artificial recurrent neural networks (LSTM to be more precise), random forest and CatBoost regression, which is what our abstraction is based on. The code is written in Python using Jupyter Notebook.

Finally, this paper also compares our implementation with other algorithms for earthquake prediction using Scikit-learn, which is a library in Python.

1 INTRODUCTION

Earthquake prediction is becoming more and more relevant due to global warming. For example, pressure from melting ice can cause tectonic plates to shift, causing major earthquakes. Major earthquakes can be devastating and cause tsunamis, which is why it's so important to predict them.

A major earthquake has never been predicted before, and many scientists question whether it is even possible to predict earthquakes[1-2]. Nevertheless, we have developed an abstraction for predicting earthquakes, because even though one might not be able to predict earthquakes, the biggest earthquakes have patterns and one can still forecast them.

Also, there was an earthquake that was predicted in China several decades ago by looking at precursors (even though a later study said this was not a valid short-term prediction)[2]. In our abstraction we are using patterns of laboratory seismic waves to predict time of earthquakes.

2 RELATED WORK

This section discusses previous work which have influenced the work presented in this paper. Earthquake prediction problem is initially considered as a time series prediction. Later, earthquake prediction is carried out on the basis of computed features in place of time series of earthquakes, thereby converting a time series prediction problem into a classification problem [8].

We used Hann's function named after Julius von Hann [13], which is a window function, as one of the statistical summaries in the preprocessing of the dataset. We used the Hann function because our data is seismic data, which is periodic (repeating in time). This means that there will be a discontinuity at the edges of the data. The Hann function makes sure that the edges are zero, thus avoiding discontinuity.

Random forest classifier was successfully applied in earthquake prediction in laboratory environment [9]. Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks according to [6]. That gives us an incentive to reuse random forest in this project as a comparable method to LSTM which today stands as one of the most popular machine learning algorithms.

CatBoost is open-sourced machine learning algorithm developed and contributed by Yandex. CatBoost does not require extensive data training like other machine learning models, and can work on a variety of data formats, which motivated us to use it in this work.

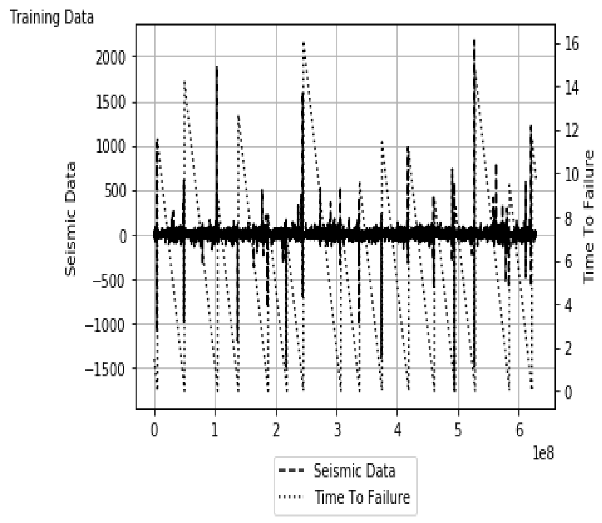


Figure 2: Training Data Plot

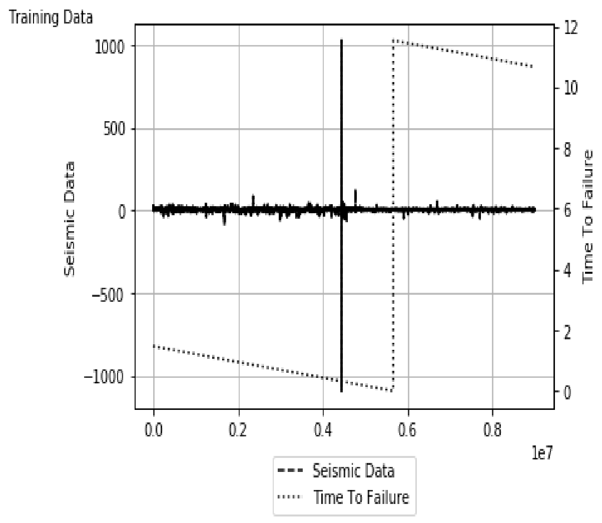


Figure 3: Earthquake zoomed-in plot

3 DATASET AND FEATURES

3.1 Dataset

The dataset used was provided by Los Alamos National Laboratory for a 2019 data science competition (Kaggle) and comes from an experimental set-up used to study earthquake physics. The goal of this competition is to use acoustic signals to predict the timing of laboratory earthquakes.

The dataset consists of approximately 629 millions of individual measurement of acoustic input signal, in total 8,9 GB and it is a single, continuous segment of experimental data. The test data consists of a folder containing 2.624 small segments (all together 823 MB), each consisting of 150.000 individual acoustic measurements.

The data within each test file is continuous, but the test files do not represent a continuous segment of the experiment; thus, the predictions cannot be assumed to follow the same regular pattern seen in the training file.

The dataset was too big to upload to Github, so we put the dataset in C:/data locally.

3.2 Data Processing

For LSTM, RF and BCG modeling approach we applied same preprocessing steps. The two columns in the train dataset have the following meaning:

- acoustic data: is the acoustic signal measured in the laboratory experiment;

- time to failure: this gives the time until a failure will occur.

We visualized the data by making a plot using Matplotlib and sampled train dataset with step 1000. The plot compares the feature attribute, seismic waves with the target attribute, time-to-failure.

There are 16 earthquakes simulated in the lab experiment (see figure 2). The acoustic data shows complex behaviour (amplitude and frequency). Just before each failure there is a strong increase in the amplitude followed by signals with significantly shorter amplitude (see figure 3).

Train dataset is splited in 4195 segments of 150.000 successive acoustic measurements in order to be comparable to test segments. The original data is dirty [11] in the sense that it is incomplete and noisy. The data is incomplete because it contains only aggregate data (acoustic signal is the only feature other than the target feature) and the data is noisy because it contains errors or outliers. Since the data set containes only aggregate data, we compute numerous statistical descriptors/summaries and scale them using MinMaxScaler from Scikit-learn which solves the latter problem concerning outliers/errors. Target train value corresponding to each segment is an original target value of the last row of the segment. For each test segment in the test folder our model will predict a single time-to-failure corresponding to the time between the last row of the segment and the next laboratory earthquake.

3.3 Features

We use standard statistical features like mean, standard deviation, minimum and maximum calculated on the whole segment, on the first or on the last portion of the segment. We also use different quantiles, trends and absolute trends (based on linear regression), number of values larger than threshold (in our settings it is 500) and some signal processing features as average amplitude envelope (hilbert-mean) and Hann window. Finally, we use features extracted by a rolling window approach (for three window sizes) hat have been used in initial work of Bertrand Rouet-Leduc and the LANL-group [9].

4 METHODS

4.1 Neural Networks

As the name suggests, neural networks are algorithms that are inspired by the human brain. Neural networks are a type of deep learning architecture. Deep learning is a machine learning method that learns features and tasks directly from data such as text, sound or images and classifies target features.

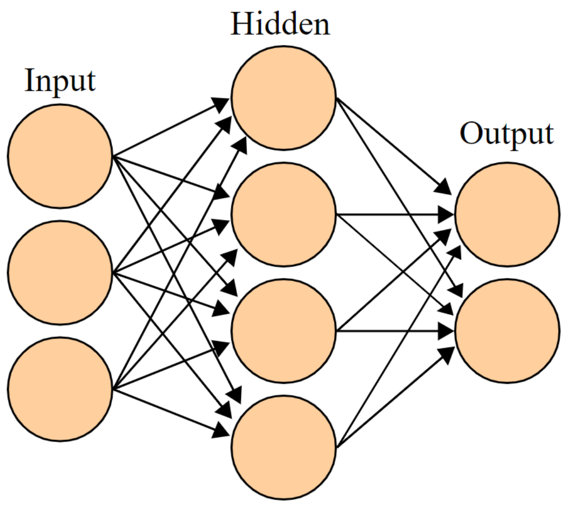


Figure 4: Neural Network Layers. Source: [https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Neural_Network_Basics]

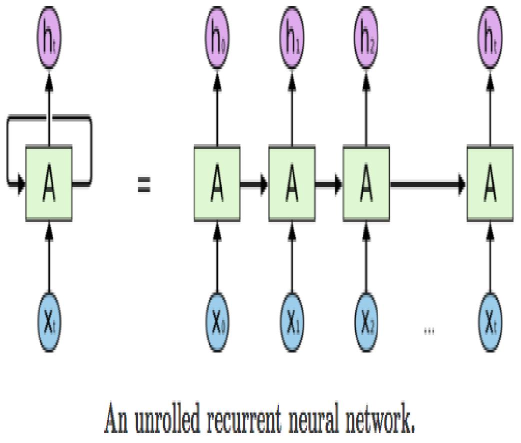


Figure 5: Recurrent Neural Network. Source: [https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912]

A neural network is like any other network in the way that it consists of a web of nodes or neurons that are connected by edges. Neural networks have many applications, but in this abstraction they are used for regression. In the kaggle competition, the target values to be predicted (time to failure), are numeric continuous values, which implies that this is a regression problem.

Neural networks consists of three types of layers (see figure 4); the input layer, hidden layers, and the final layer. The hidden layers are all the layers between the input layer and the final layer.

Each of the nodes in the hidden and output layers, have their own classifier. First of all, the nodes in the first hidden layer compute scores using a set of inputs from the input layer. The nodes in the first hidden layer that then activates will pass on their classification scores to the next hidden layer. Nodes are activated based on activation functions. The ReLU function is the most used one (we used it in our abstraction). Again, nodes that active pass on their scores to the next layer. Scores are passed on from layer to layer until the output layer is reached, where classification is performed based on the scores in the final hidden layer. The above process is called forward propagation and is repeated for all the sets of inputs.

The activation of nodes are not random. A set of inputs will always activate the same nodes. The reason why some nodes activate while others does not for the same set of inputs, is that activation of nodes are also dependent on biases or weights that are unique for each node. This means that classification accuracy depends on the biases of the nodes.

Accuracy can be increased by training the classifier. This is done by modifying the biases for each round of training. The biases are modified to minimize the difference between the classified target value(s) and the real target value(s), and the number of training rounds (epochs) should be as high as possible as to minimize this difference as much as possible.

4.1.1 Recurrent Neural Networks. Recurrent neural networks are good at modeling sequencial data, and the input data in the Kaggle earthquake prediction competition is seismic (sequential) data, which is why we chose to use recurrent neural networks in our abstraction. RNNs are good at modeling sequence data because they introduce artifical memory by using internal state. RNNs uses memory for prediction by creating loops between hidden layers as you can see in figure 5.

Sometimes when training RNN networks, the loss gradient becomes really small in the first hidden layers. This is a problem because the loss gradient is used to update weights, and the smaller the loss gradient is, the smaller the updates will be, which again leads to small loss reduction. This problem is known as 'the vanishing gradient problem'.

The reason this problem usually occurs is that the gradient is the product of multiple derivatives, and if most of these are less than 1, the gradient will be an even smaller number. The weights in a neural network are updated to the difference between weights and the gradient multiplied with the learning rate which is between 0 or 1 (resulting in again, an even smaller number). Obivously, the difference between the weights and numbers close to 0, will be approximately the same as the weights themselves, leading to little change after updates. The reason this problem usually occurs in the first layers, is that there are the most terms in the first layers, and the more terms below 1 that are multiplied together, the quicker the gradient will decrease.

LSTM (Long Short-Term Memory) networks are a type of recurrent neural networks that remembers specific sequences for long periods of time, thus solving the vanishing gradient problem.

LSTM consists of memory block cells for different time steps (see figure 6). These cells are responsible for remembering, and they have two states; hidden state (information in memory) and cell

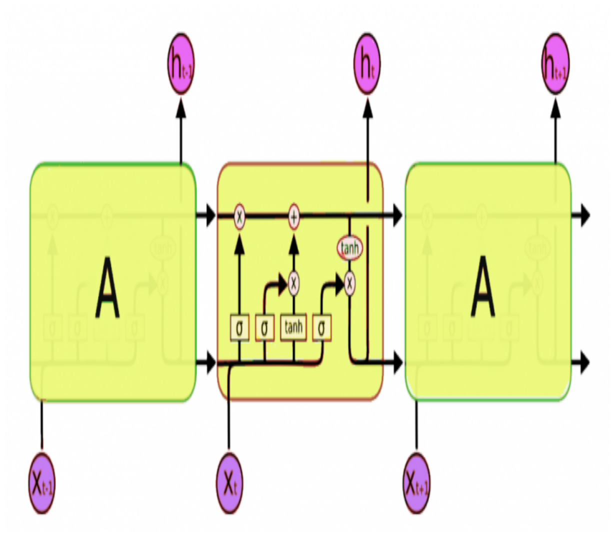


Figure 6: LSTM Architecture. Source: [https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/]

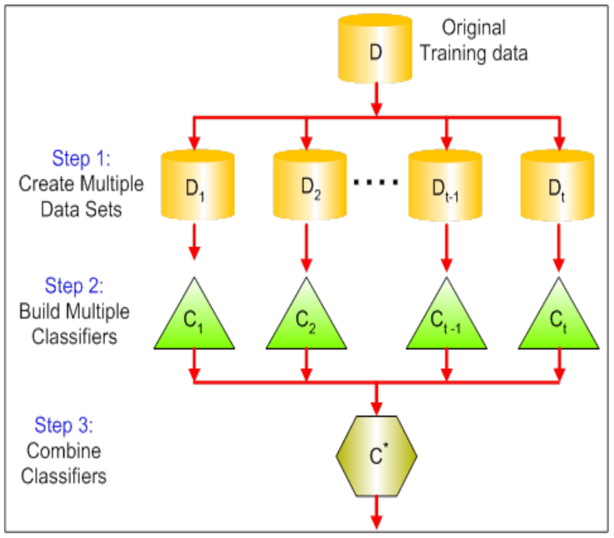


Figure 7: Ensemble learning method. Source: [https://www.analyticsvidhya.com/blog/2015/09/questions-ensemble-modeling/]

state (cell output). The cells also consists of three main components that control the cell states; forget gate, input gate, and output gate.

The forget gate filters information by deciding which information to remember and which information to discard.

LSTM solves the vanishing gradient problem by creating a connection between the forget gate activations and the calculation of the gradient. This connection creates a path for information flow through the forget gate for information the LSTM should not forget [3].

4.2 Random Forest

Random forest is one of ensemble methods. Ensemble methods are classification techniques that improve classification accuracy by aggregating the prediction of multiple base classifiers that are constructed from training data. While the ensemble classifier can be constructed in different ways, the logical procedure is the same: it starts with original data set, manipulates the data set, creates multiple base classifiers and finally combine them (see figure 7).

Aggregation is done by taking a vote on the predictions made by each base classifier (classification problem) or by taking average value of all predictions (regression problem) [7]. Ensemble methods work good with basic classifiers that are sensitive to minor perturbations in the training data set, so-called unstable classifiers, one of which is a decision (regression) tree.

Two conditions are necessary for an ensemble method to perform better than a base (single) classifier: independence of the base classifiers and the base classifiers should be better than random guessing. Although it is difficult to ensure independence in practice, the ensemble classifiers are often used and accuracy improvement is observed even if the base classifiers are slightly correlated [7].

Random forest manipulates its input features and uses decision (or regression, depending on problem) trees as its base classifiers. Decision (regression) trees are invariant under strictly monotone transformations of features, which means that scaling or other transformations can be used during preprocessing. Feature selection is performed as a part of the building tree procedure, which makes them resistant to the inclusion of many irrelevant features. These properties make them very popular learning method for data mining. But trees are inaccurate (seldom provide predictive accuracy) [14] and "deep" trees tend to overfitting, i.e tend to have low bias, but very high variance.

Random Forest prevents overfitting most of the time, by creating random subsets of the features and building smaller trees using these subsets. Subset can be chosen based on specific domain knowledge (which is not case in this project). This approach (random feature choice) gives good results even when the features are highly redundant [7]. We could expect that feature creation done in preprocessing part of the project could lead to certain level of redundancy and correlation among features and the random forest could be a good algorithm candidate in such circumstances. Afterwards, random forest combines (averages) the subtrees (see figure 8), with the goal of reducing the variance. Advantage of non-overfitting comes at a price: it leads to a small increase in the bias, some loss of interpretability and a large number of trees can make the algorithm to slow and ineffective for real-time predictions. A more accurate prediction requires more trees, which results in a slower model [6].

There are three approaches to randomization in random forest. The first one is randomly select F features to split at each node. Thus, splitting decision is determined from selected features and the tree is grown without pruning. Since F is only a portion of features, this approach can reduce runtime. This approach is known as Forest-RI (random input selection). The second one is used weather the number of features is small and is difficult to select subset of independent features. Then the feature space is enlarged by linear combinations of features (when linear coefficients are from a uniform distribution and in the range [-1,1]). At each node F such new

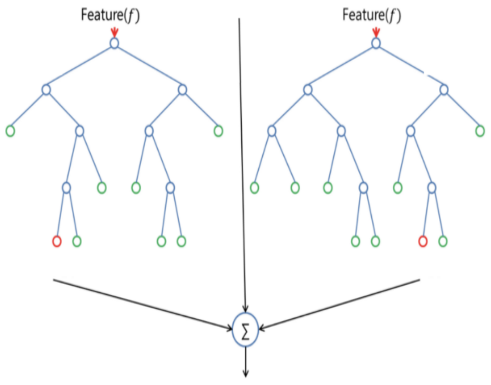
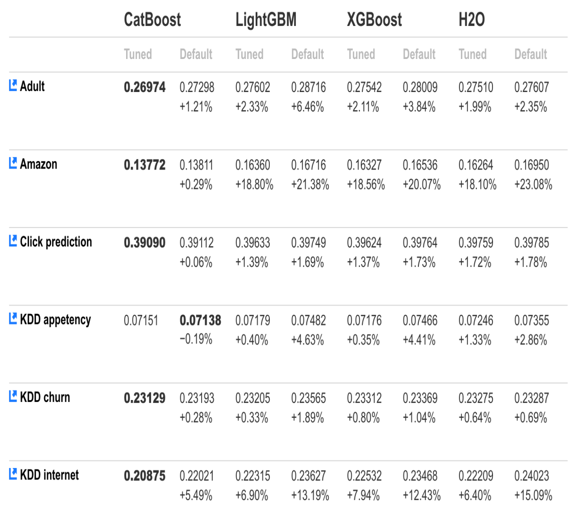


Figure 8: Ensemble learning method. Source: [https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd/]



	CatBoost		LightGBM		XGBoost		H2O	
	Tuned	Default	Tuned	Default	Tuned	Default	Tuned	Default
Adult	**0.26974**	0.27298 +1.21%	0.27602 +2.33%	0.28716 +6.46%	0.27542 +2.11%	0.28009 +3.84%	0.27510 +1.99%	0.27607 +2.35%
Amazon	**0.13772**	0.13811 +0.29%	0.16360 +18.80%	0.16716 +21.38%	0.16327 +18.56%	0.16536 +20.07%	0.16264 +18.10%	0.16950 +23.08%
Click prediction	**0.39090**	0.39112 +0.06%	0.39633 +1.39%	0.39749 +1.69%	0.39624 +1.37%	0.39764 +1.73%	0.39759 +1.72%	0.39785 +1.78%
KDD appetency	0.07151	**0.07138** −0.19%	0.07179 +0.40%	0.07482 +4.63%	0.07176 +0.35%	0.07466 +4.41%	0.07246 +1.33%	0.07355 +2.86%
KDD churn	**0.23129**	0.23193 +0.28%	0.23205 +0.33%	0.23565 +1.89%	0.23312 +0.80%	0.23369 +1.04%	0.23275 +0.64%	0.23287 +0.69%
KDD internet	**0.20875**	0.22021 +5.49%	0.22315 +6.90%	0.23627 +13.19%	0.22532 +7.94%	0.23468 +12.43%	0.22209 +6.40%	0.24023 +15.09%

Figure 9: Gradient Boosting Comparison. Source: [https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/]

features are generated and the best are used to split the node. This approach is known as Forest-RC (random combination). The third approach is randomly select one of the F best splits at each node. F should be large enough to avoid generation of correlated trees. This algorithm does not save runtime as the first two [14].

4.3 CatBoost

Boosting is a method of converting weak learners into strong learners. In boosting, each new tree is a fit on a modified version of the original data set. The gradient boosting algorithm trains a decision tree on each observation with possibly different assigned weight. After evaluating the first tree, algorithm increases the weights of those observations that are difficult to classify and lower the weights for those that are easy to classify. The second tree is grown on weighted data in hope that will improve upon the predictions of the first tree. This process is repeated for a specified number of iterations. Subsequent trees help to classify observations that are not well classified by the previous trees. Predictions of the final ensemble model is the weighted sum of the predictions made by the previous tree models [18].

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak learner, weak classifier, or weak predictor. No matter what the distribution over the training data is it will always do better than chance (an error rate will be less than 1/2) according [17]. It involves three elements: a loss function to be optimized, a weak learner to make predictions and an additive model to add weak learners to minimize the loss function. Several versions of the algorithm includes GBM, XGBoost, LightGBM and CatBoost (see figure 9).

The comparison (figure 9) shows the log-loss value for test data and it is lowest in the case of CatBoost in most cases. It clearly signifies that CatBoost mostly performs better for both tuned and default models, which motivated us to use it in this project.

"CatBoost is an algorithm for gradient boosting on decision trees. Developed by Yandex researchers and engineers, it is the successor of the MatrixNet algorithm that is widely used within the company for ranking tasks, forecasting and making recommendations. It is universal and can be applied across a wide range of areas and to a variety of problems."[16]

5 RESULTS AND DISCUSSION

5.1 Evaluation Metrics

As mentioned in the abstract, the accuracy of our models are computed using MSE by Kaggle.

5.2 Hyperparameters

5.2.1 LSTM. We used the following formula to find the optimal number of neurons for the LSTM model:

$$N = N?/(a * (N? + N?))$$

, where N? is the number of input neurons, N? the number of output neurons, N? the number of samples in the training data, and a represents a scaling factor that is usually between 2 and 10 [12].

We used two dense hidden layers because it is usually enough to solve complex problems (simple problems can generally be solved using only 1 layer).

The number of epochs was determined by using the training curve in figure 10.

5.2.2 Random Forest. The scikit-learn implementation of random forests (RandomForestRegressor class) is used in this project.

Each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model. The scikit-learn implementation combines classifiers by averaging their probabilistic prediction.[15]

Hyperparameters that are used to increase the predictive power of the model are "n esimators", "max features" and "min sample leaf".

The "n estimators" is the number of trees the algorithm builds. In general, a higher number of trees increases the performance, but it also slows down the computation.

The "max features" is the the maximum number of features Random Forest considers to split a node.

The "min sample leaf" is the minimum number of leafs that are required to split an internal node.

5.3 Experiments

5.3.1 LSTM. The LSTM abstraction was implemented using Python with Jupyter Notebook. The neural network was constructed using LSTM from the Keras API. First off the data is loaded, and plotted to get a feel of the patterns in the data. Next, we generated a Keras Sequential model (see Listing 1). Then an LSTM layer with 64 neurons, a time step of 1 (because there is 1 feature in the dataset), and an input dimension of 1 is added to the model. Then, two Dense layers were also added to the model (the first one with 32 neurons and the 'ReLu' activation function, and the second one with 1 neuron). We used 'ReLu' because it speeds up the training process because the gradient computations performed by 'ReLu'is very simple ('ReLu' sets elements to either 0 or 1 based on the sign of the given elements). After creating the model, the compile() method is called to specify the loss function, which in this case is 'mae' (Mean Absolute Error). We chose to use mean absolute error as the loss function because it's robust to deviations. As previously stated, LSTM alleviates the vanishing gradient problem, and we also used an optimizer called ADAM (Adaptive Moment Estimation) which is also specified by the compile() method and alleviates the vanishing gradient problem further. ADAM is one of the best optimizers when it comes to training error. A contestor is SGD. ADAM is best for the first epochs, while SGD is best for the last ones, but the difference is not significant. The best would be to combine ADAM and SGD by using ADAM for the first epochs and then SGD.

Listing 1: Create Model

```
self.model = Sequential()
self.model.add(LSTM(64,input_shap
e=(self.num_features, 1)))
self.model.add(Dense(32, activati
on='relu'))
self.model.add(Dense(1))
self.model.compile(optimizer=adam
(lr=0.005), loss="mae")
```

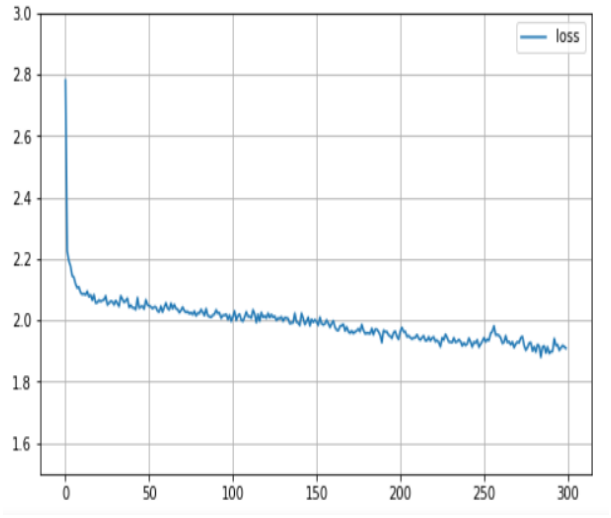


Figure 10: Training Curves

Listing 2: Train Model

```
self.model.fit(feature_train, target_train, epochs=epochs,
    batch_size=batch_size)
```

Before finally calling the model's predict() method (see Listing 3) to estimate probabilities of different values for each instance in the test set, the test data is pre-processed by adding some statistical summaries to the test data and then applying min-max feature scaling to it (see Section 3.2).

Listing 3: Predict Model

```
forecasts = self.model.predict(feature_test)
```

In section 3.2 we mention that one of the statistical descriptors computed is the Hann window. Another statistical feature that is added in the preprocessing is computed using the Pandas method, Rolling. This feature consists of means computed over and over again for different subsamples or windows of data (see Listing 4). Just like 'msa', feature scaling makes the algorithm more robust to deviants by normalizing the data.

Listing 4: Pandas Rolling

```
int_mean =
    interval_data.rolling(windows).mean().dropna().values
```

5.3.2 Random Forest. The random forest abstraction was implemented using Python 3.6.5 with Jupyter Notebook. Like in the LSTM abstraction, first the data is loaded, plotted to get a feel of the patterns in the data and the same preprocessing routine is used. Next, finding the best combination of hyperparameters lies at the heart of random forest. We use Grid Search (see listing 6) for that purpose (with "parameters" as starting setup).

Listing 5: Grid Search

```
parameters = {'max_depth': [10, 20, 30],
              'n_estimators': [100, 500, 1000],
              'max_features' : [15, 20, 25, 30],
              'min_samples_leaf': [1, 2, 4],
              'min_samples_split': [2, 5, 10],
              'bootstrap': [True, False],
             }
self.gsc = GridSearchCV(estimator =
     RandomForestRegressor(), param_grid = self.parameters,
     cv = 5, scoring = 'neg_mean_squared_error', verbose =
     0, n_jobs = -1)
```

Grid Search trains 648 (3*3*4*3*3*2) different models, evaluates all of them and returns the configuration of hyperparameters that scores highest (see listing 7), which is time consuming (almost 11 hours in our environment).

Listing 6: Best Estimator

```
def best_rf(self, data: pd.core.frame.DataFrame):
    feature_train, target_train =
         self.generate_feature_and_target(data)
    best_rf = self.gsc.fit(feature_train, target_train)
    best_estimator = best_rf.best_estimator_

    return best_rf, best_estimator

best estimator RandomForestRegressor(bootstrap=True,
     criterion='mse', max_depth=10, max_features=20,
     max_leaf_nodes=None, min_impurity_decrease=0.0,
     min_impurity_split=None, min_samples_leaf=1,
     min_samples_split=5, min_weight_fraction_leaf=0.0,
     n_estimators=1000, n_jobs=None, oob_score=False,
     random_state=None, verbose=0, warm_start=False)
```

Best estimator is then trained (listing 8).

Listing 7: Train Best Estimator

```
best_estimator.fit(feature_train, target_train)
```

Finally, forecasting is done (listing 9) on the test data that is already pre-processed by adding statistical summaries and scaled by applying min-max feature scaling (see Section 3.2).

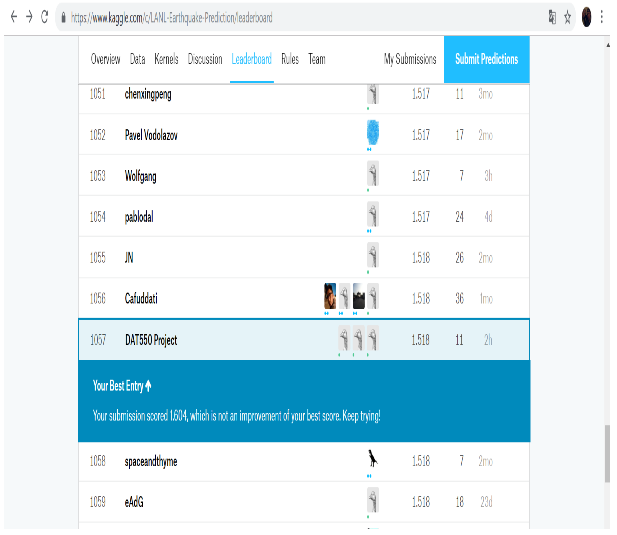
Listing 8: Forecast

```
forecast(best_estimator, rf.scaler, feature_test)
```

For experimental cause three starting setups for Grid Search are used (two of them in listing 9 and 10).

Listing 9: Grid Search and Best Estimator 2

```
parameters = {'max_depth': [5, 10, 20, 30],
              'n_estimators': [10, 50, 100, 1000],
              'max_features' : [20, 40, 70],
              'min_samples_leaf': [1, 2, 4],
              'min_samples_split': [2, 5, 10],
              'bootstrap': [True, False],
             }
```


https://www.kaggle.com/c/LANL-Earthquake-Prediction/leaderboard

1051	chenxingpeng	1.517	11	3mo
1052	Pavel Vodolazov	1.517	17	2mo
1053	Wolfgang	1.517	7	3h
1054	pablodal	1.517	24	4d
1055	JN	1.518	26	2mo
1056	Cafuddati	1.518	36	1mo
1057	DAT550 Project	1.518	11	2h

Your Best Entry ↑

Your submission scored 1.604, which is not an improvement of your best score. Keep trying!

1058	spaceandthyme	1.518	7	2mo
1059	eAdG	1.518	18	23d

Figure 11: LANL Earthquake Prediction Leaderboard

```
best estimator RandomForestRegressor(bootstrap=True,
     criterion='mse', max_depth=5, max_features=15,
     max_leaf_nodes=None, min_impurity_decrease=0.0,
     min_impurity_split=None, min_samples_leaf=2,
     min_samples_split=5, min_weight_fraction_leaf=0.0,
     n_estimators=100, n_jobs=None, oob_score=False,
     random_state=13, verbose=0, warm_start=False)
```

Listing 10: Grid Search and Best Estimator 3

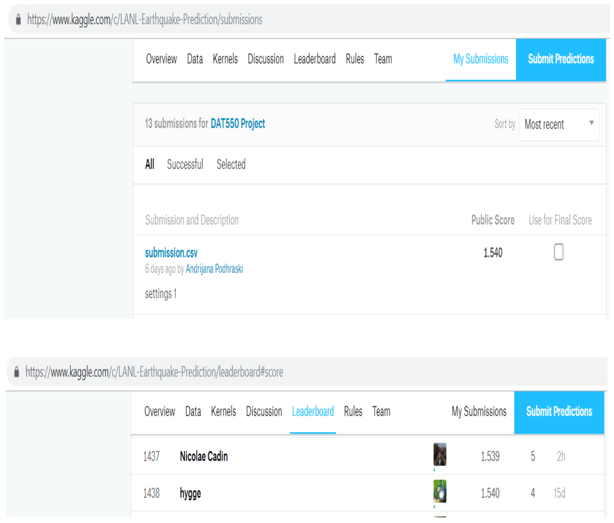
```
parameters = {'max_depth': [3, 5, 10, 15],
              'n_estimators': [50, 100, 500, 1000],
              'max_features' : [10, 15, 20],
              'min_samples_leaf': [1, 2, 4],
              'min_samples_split': [2, 5, 10],
              'bootstrap': [True, False],
             }

best setimator RandomForestRegressor(bootstrap=False,
     criterion='mse', max_depth=3, max_features=15,
     max_leaf_nodes=None, min_impurity_decrease=0.0,
     min_impurity_split=None, min_samples_leaf=4,
     min_samples_split=2, min_weight_fraction_leaf=0.0,
     n_estimators=500, n_jobs=None, oob_score=False,
     random_state=13, verbose=0, warm_start=False)
```

5.4 Results

The submissions for the LANL earthquake prediction competition on Kaggle were computed using Mean Absolute Error. As of 11th April '19, the best score our LSTM model got is 1.518, which put us on 1057th place out of 2462 competitors on the Kaggle leaderboard (see figure 11).

As of 18th April '19, the best score our random forest model got is 1.540 (almost as good as our LSTM), which is 1438th place out of 2791 competitors on the Kaggle leaderboard (see figure 12).


https://www.kaggle.com/c/LANL-Earthquake-Prediction/submissions

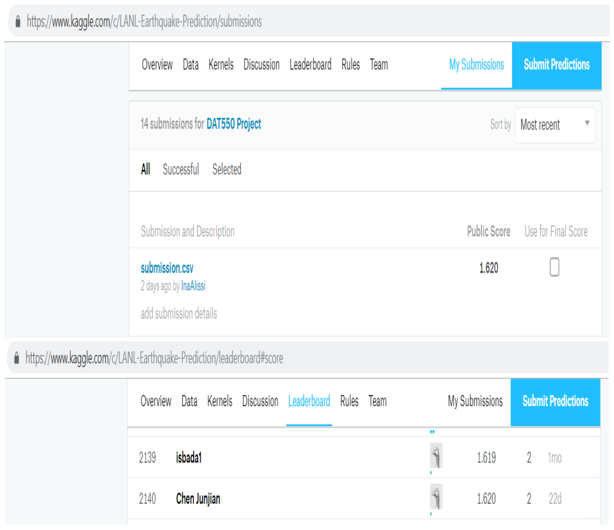
13 submissions for DAT550 Project

Submission and Description | Public Score | Use for Final Score
submission.csv — 6 days ago by Andrijana Podhraski — settings 1 | 1.540 |

https://www.kaggle.com/c/LANL-Earthquake-Prediction/leaderboard#score

1437 Nicolae Cadin 1.539 5 2h
1438 hygge 1.540 4 15d

Figure 12: LANL Earthquake Prediction Random Forest score


https://www.kaggle.com/c/LANL-Earthquake-Prediction/submissions

14 submissions for DAT550 Project

Submission and Description | Public Score | Use for Final Score
submission.csv — 2 days ago by InaAlissi — add submission details | 1.620 |

https://www.kaggle.com/c/LANL-Earthquake-Prediction/leaderboard#score

2139 isbada1 1.619 2 1mo
2140 Chen Junjian 1.620 2 22d

Figure 13: LANL Earthquake Prediction CatBoost score

As of 24th April '19, the best score our CatBoost model got is 1.620 (slightly below than LSTM and random forest), which is 2140th place out of 3056 competitors on the Kaggle leaderboard (see figure 13).

We compared the score of our model with three other models, namely, SVR, Linear Regression, and Gradient Boosting Regressor using the python library, Scikit-learn. The Scikit models are based on the basic feature benchmark provided by Kaggle [4]. The only model that got a score close to ours, was Gradient Boosting Regressor, which got a score of 1.836. However, deep learning such as LSTM are very slow at training. Training our model can take anywhere from 1 to 3 hours. One should choose a model based on required accuracy; if the accuracy requirements are low, one should choose a faster algorithm than LSTM, such as Linear Regression. Otherwise one should go with a more precise algorithm such as LSTM.

Python code for LSTM, Random Forest, CatBoost and the Scikit classifiers for comparison developed in this project can be found on the following links (respectively):

```
https://github.com/uis-dat550-spring19/Group-nr-1/tree/master/
    earthquake_predictions_LSTM
https://github.com/uis-dat550-spring19/Group-nr-1/tree/master/
    earthquake_predictions_RF
https://github.com/uis-dat550-spring19/Group-nr-1/tree/master/
    earthquake_predictions_CBG
https://github.com/uis-dat550-spring19/Group-nr-1/blob/master/
    Scikit-Classifiers.ipynb
```

6 CONCLUSION

The paper successfully implements an abstraction that predicts earthquakes with a decent performance. Of course, 'decent performance' is not a concise definition, but nevertheless, our LSTM scored above average, random forest scored slightly below in the LANL Earthquake Competition hosted by Kaggle, while CatBoost scored as third among our three methods.

7 CONTRIBUTIONS

This project uses three different machine learning approaches. We each tackled one of these approaches: LSTM and the Scikit Classifiers for comparison (Mjaaland, Henrik), Random Forest (Podhraski, Andrijana), CatBoost (Alissi, Inam).

REFERENCES

[1] Anonymous, (2019 26 03). Earthquake Prediction. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Earthquake_prediction
[2] Anonymous, (2019 26 03). Can you predict earthquakes?. Retrieved from USGS: https://www.usgs.gov/faqs/can-you-predict-earthquakes?qt-news_science_products=0#qt-news_science_products
[3] Nir Arbel, (2019 27 03). How LSTM networks solve the problem of vanishing gradients. Retrieved from medium: https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577
[4] Inversion (Kaggle username), (2019 11 04). Basic Feature Benchmark. Retrieved from Kaggle: https://www.kaggle.com/inversion/basic-feature-benchmark
[5] Anonymous, (2019 14 04). Random forest. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Random_forest
[6] Niklas Donges, (2019 14 04).The Random Forest Algorithm. Retrieved from Towards Data Science: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd
[7] P-N. Tan, M. Steinbach, V. Kumar. Introduction to Data Mining. Pearson Education Limited, 2014.
[8] Khawaja M. Asim, Adnan Idris, Talat Iqbal, Francisco Martínez-Álvarez, (2019 15 04). Earthquake prediction model using support vector regressor and hybrid neural networks https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6033417/
[9] Bertrand Rouet-Leduc, Claudia Hulbert, Nicholas Lubbers, Kipton Barros, Colin J. Humphreys, Paul A. Johnson, (2019 15 04).Machine Learning Predicts Laboratory Earthquakes. Retrieved from Advancing Earth and Space Science: https://agupubs.onlinelibrary.wiley.com/doi/full/10.1002/2017GL074677
[10] Anonymous, (2019 16 04). Julius von Hann. Retreived from Wikipedia: https://en.wikipedia.org/wiki/Julius_von_Hann
[11] Jaak Vilo, (2019 16 04). Descriptive analysis, preprocessing, visualisation. Retreived from Tartu Ülikool - courses: https://courses.cs.ut.ee/MTAT.03.183/2017_spring/uploads/Main/DM_01_Descriptive.pdf
[12] Karsten Eckhardt, (2019 16 04). Choosing the right hyperparameters. Retreived from Towards Datascience: https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046
[13] R. B. Blackman, J. Tukey. Particular Pairs of Windows. "The Measurement of Power Spectra, From the Point of View of Communications Engineering", New

York: Dover, 1959, pp. 98–99.

[14] Leo Breiman, (2019 17 04). Random Forest. Retrieved from Statistics Department University of California Berkeley: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

[15] 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, E. Duchesnay, (2019 17 04). Ensemble methods. Retrived from Scikit Learn: https://scikit-learn.org/stable/modules/ensemble.html#forest

[16] Anonymous, (2019 24 04). CatBoost. Retreived from: https://catboost.ai/

[17] Anonymous, (2019 24 04). Gradient boosting. Retreived from Wikipedia: https://en.wikipedia.org/wiki/Gradient_boosting

[18] Harshdeep Singh, (2019 24 04). Understanding Gradient Boosting Machines. Retreived from Towards Data science: https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab