Alex Khater
ML1 Final Project Individual Report

Introduction

This is a log of the work I have completed on my Machine Learning final project focused on tsunami prediction. For this project I located a dataset that contained basic information on nearly 800 seismic events since January 1st 2001. The intention of this project was to develop a binary classification model that could determine whether or not a tsunami would occur given information about an earthquake. Earthquakes that create tsunamis can cause untold damage and death several orders of magnitude greater than just the initial earthquake and so this project sought to develop a model that could potentially serve as an early warning system to minimize the impact these events have.

This report is not intended as a grand summary of the project so much as a log of my individual work. Please see the final report for more detail and research into the models used and the conclusions drawn.

Individual Work

In each phase of the project, I had a different set of tasks. In the early planning phases of the project, I was the one that located our dataset from Kaggle and developed the pitch that would turn into our full project. I also wrote the initial draft for our proposals. I also conducted the majority of the literature review and presented my findings on the RIKEN Pioneering Project study on using deep learning and advanced seismic data to predict tsunamis in Japan[1]

In the pre-modeling phase I cleaned the data by dropping the very small amount of null values that our dataset had, removing any observation that I classified as an outlier (having a value greater than mean + 3*std).

After the initial data cleaning phase, I performed some Statistical Tests. I used a correlation plot to see the correlation between all the variables and then I used a t-test for correlation significance with the target variable, ‘tsunami’s’ and I verified that there are some significant correlations to features we have in our dataset, which is a good sign as it shows that there is useful information for our models to use. Most of the correlations seemed quite low and 27 features is not enough to need immense dimensionality reduction so I decided not to use PCA.

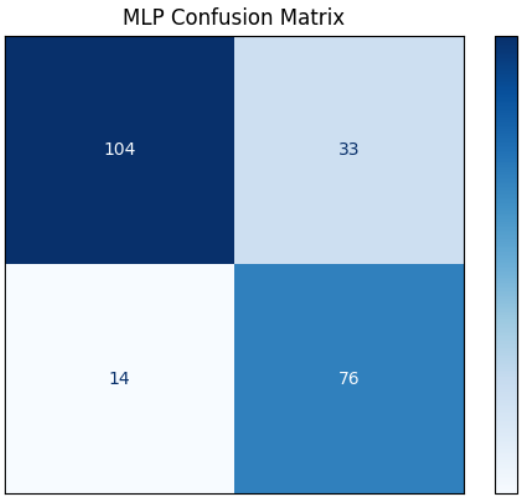
Firstly I removed the binary target tsunami variable from the dataset so as to not affect it with the scaling and encoding. I used the Shapiro Wilks Test to check for normality (since our sample size is under 1000) and then when there were significant p-values signaling non-normality in certain numerical variables. I used the Standard Scaler from sklearn to normalize the data. I also used pd.get_dummies to encode the variables. After this encoding and scaling was done, I re added to the target data and saved the prepped data as a separate csv so my group mates and I could easily use it for modeling.

[1] https://www.nature.com/articles/s41467-022-33253-5#Sec11

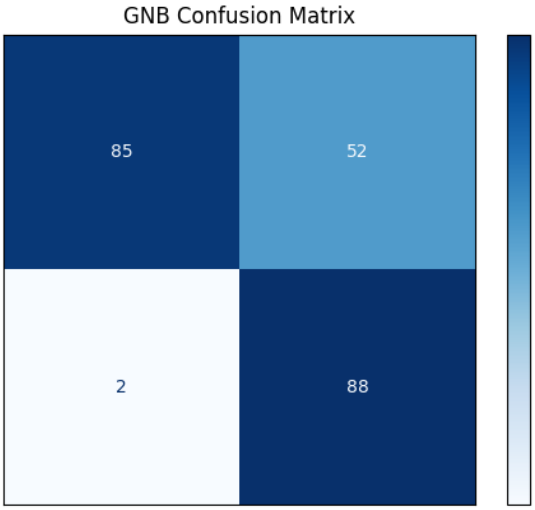
Modeling:

To start the modeling phase, I tested 3 Neural Networks with default parameters to classify whether or not an Earthquake would create a tsunami. These 3 networks were an MLP, GNB, and XGBOOST. The confusion matrices are posted below

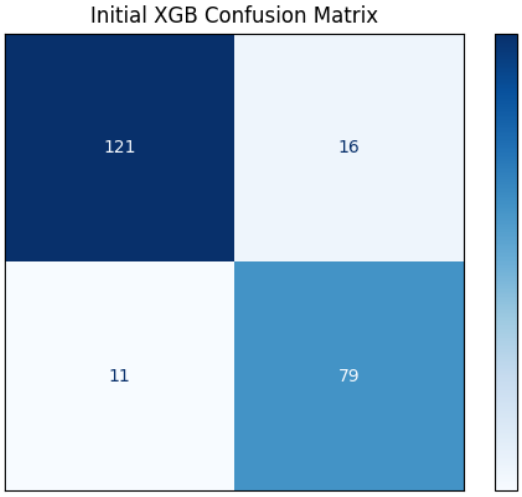
The accuracy score for the MLP (with no tuning): 0.793



The accuracy score for the GNB (with no tuning):0.762

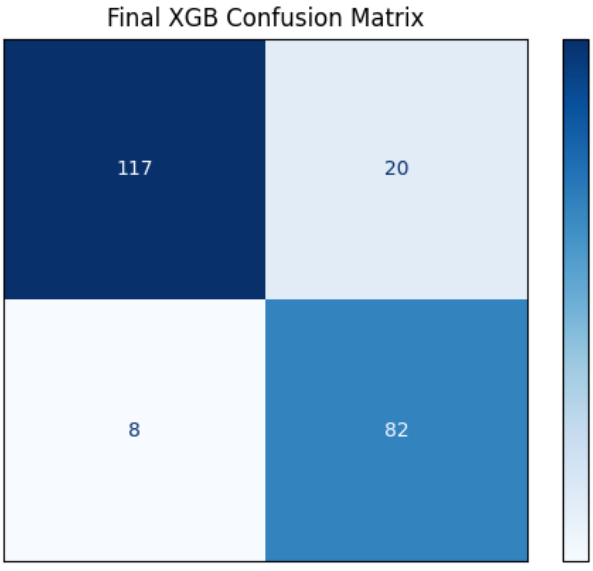


The accuracy score for the XGB (with no tuning): 0.881



We can see that pre-tuning the XGBoost model outperforms the other two easily, so I decided to put my effort towards tuning that algorithm to try and get better results. I implemented grid search cv for one iteration, looked at the best parameters output and readied a second gridsearch pass for the parameters that were selected on the edge of their ranges. After that I created a new XGBoost Model with the following confusion matrix.

Final XGBoost Accuracy: 0.877



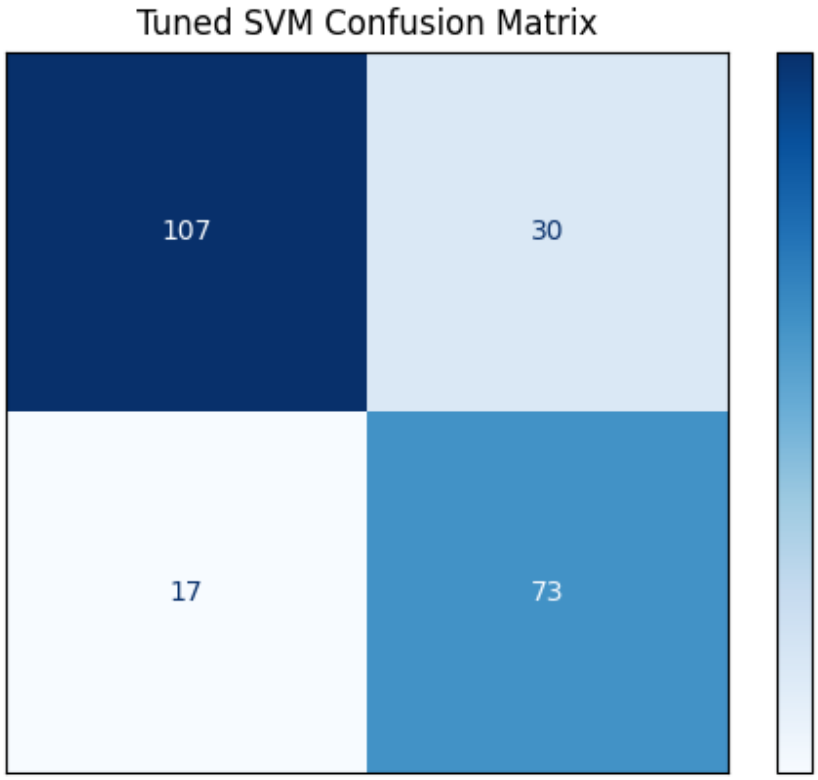
I was initially taken back when after the hyperparameter tuning, my accuracy score slightly decreased, implying that the unoptimized XGBoost was the best. Since our data was unbalanced, I decided to consult the RoC_AUC (Receiver Operating Characteristic Area Under Curve) metric instead as it is generally better for comparing classification models, especially

when working with unbalanced data. Running all the previously mentioned models here are the ROC_AUC scores:

Tuned XGB: 0.951419302514193
First XGB: 0.9463909164639092
MLP: 0.8616382806163829
GNB: 0.8031630170316302

We can see that per this metric, the tuned XGB is the best method and should be used. But these are all neural networks and I was curious to see if a non-neural network approach would yield better results so I decided to implement an SVM.

I used the same process I did with the xgboost models for the SVM. After Hyperparameter Tuning this was the result



With a ROC_AUC score of: 0.8716950527.

We can see after all this work, that the tuned xgboost model is far and away the best for this classification task, at least of the models that I tried. Next I am going to present my results to my group mates and see who has made the most effective models. We decided against using the GNB model, but we kept the others

After my coding was finished, I compiled my code with my group mates code and created a Code folder in the github with an accompanying 'read me'. First I merged my cleaning file, my statistics file, and Alejandra's EDA into a preprocessing file. After that I merged all 3 of our modeling files into one singular file so that all the modeling data was there. We all collaborated on putting together the final report and presentation.

RESULTS

Here is a table summarizing my previous results:

Method	Precision	Recall	ROC_AUC	F1 Score
MLP	0.7591	0.8814	0.8616	0.8156
XGBoost	0.8540	0.9360	0.9514	0.8932
SVM	.7810	0.8629	0.8716	0.819

We can see that the XGBoost is the strongest model of the ones that I created by far. These results are very good, especially given the relative simplicity of the data we were using. As I talked about in the literature review, more advanced data would certainly lead to better results, but I am quite proud of the XGBoost Model that I created.

Code Attribution

In my final code files, I had authored about 335 lines of code. Of that code 51 lines were pulled from the internet through various tutorials. I modified 26 of those lines thus, using the formula given in the project description.

51-26/(225+51) *100= 9.05% of my code was copied from the internet. Sources below.

Code References

- XGBoost Code: https://towardsdatascience.com/beginners-guide-to-xgboost-for-classification-problems-50f75aac5390
- SVM Code: https://www.hackerearth.com/blog/developers/simple-tutorial-svm-parameter-tuning-python-r/#:~:text=One%20can%20tune%20the%20SVM,learn%20is%20called%20gridSearchCV().&text=Parameters%20of%20this%20function%20are,estimator%20object%20which%20is%20svm.

- Confusion Matrix Code:https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels
- Removal of Outliers: https://stephenallwright.com/remove-outliers-pandas/
- Encoding Dataset Tips: https://towardsdatascience.com/categorical-encoding-techniques-93ebd18e1f24

Scholarly References:

Leventis, D. (2022, January 2). *XGBoost mathematics explained*. Medium. Retrieved April 30, 2023, from https://dimleve.medium.com/xgboost-mathematics-explained-58262530904a

scikit learn. (n.d.). *scikit.learn manual*. scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation. Retrieved April 29, 2023, from https://scikit-learn.org/stable/index.html

Pupale, R. (2019, February 11). *Support vector machines(svm) - an overview*. Medium. Retrieved April 30, 2023, from https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989