Predictive Japan Earthquake Model

G6 – Epicenter Visionaries

Data Science Capstone Project
Predictive Modeling Report

Date:

02/09/2024

Team Members:

Name: Priyanka Patil (pp673)

Name: Alex Hoang (ah3756)

Name: Vijval Vemula (vv354)

Name: Mohamed Shehaf Aakil Sharfudeen (ms5475)

1. Define the Predictive Modeling Problem

A. Input: What are the input data and define the input data clearly?

- **Hypothesis:** The magnitude of earthquakes in Japan is influenced by a combination of spatial and temporal factors, including the geographical coordinates (latitude, longitude), depth of the earthquake, time of occurrence, and the seismic energy released. A predictive model can be built to estimate the earthquake magnitude using these features, providing insights into the underlying patterns and contributing factors of seismic activity in the region.
- The input consists of various features describing the characteristics, location, and context of the earthquake events in Japan. Here is the detail of all the features:
    - **time:** The timestamp when the earthquake occurred.
    - **latitude:** The geographic coordinate that specifies the north-south position of the earthquake epicenter.
    - **longitude:** The geographic coordinate that specifies the east-west position of the earthquake epicenter.
    - **depth:** The depth of the earthquake epicenter beneath the Earth's surface.
    - **mag:** The magnitude of the earthquake.
    - **magtype:** The type of magnitude scale used to measure the earthquake (e.g., Richter scale, moment magnitude scale).
    - **nst:** The number of seismic stations reporting.
    - **gap:** The largest azimuthal gap between seismograph stations.
    - **dmin:** The closest station distance to the earthquake.
    - **rms:** The root mean square of the amplitude spectrum of the earthquake.
    - **net:** The seismic network that reported the earthquake.
    - **id:** An identifier for the earthquake event.
    - **updated:** The timestamp when the earthquake data was last updated.
    - **place:** The location description of the earthquake.
    - **type:** The type of seismic event (e.g., earthquake).
    - **horizontalerror:** The horizontal error of the earthquake location.
    - **deptherror:** The error in the depth measurement of the earthquake.
    - **magerror:** The error in the magnitude measurement of the earthquake.
    - **magnst:** The number of seismic stations used to calculate magnitude.
    - **status:** The status of the earthquake event (e.g., reviewed, automatic).
    - **locationsource:** The source of the earthquake location data.
    - **magsource:** The source of the earthquake magnitude data.
    - **significant_earthquake:** A flag indicating if the earthquake is significant.
    - **magtype_mb, magtype_ms, magtype_mw, magtype_mwb, magtype_mwc, magtype_mwr, magtype_mww:** Different types of magnitude measures.
    - **distance_to_tokyo, distance_to_osaka, distance_to_kyoto:** Distances from earthquake epicenter to specific cities.
    - **seismic_energy:** The amount of energy released during the earthquake.
- Out of these features, based on the hypothesis, we extracted some of the features that are used for modelling:
    - Geographical Coordinates:
        - Latitude
        - Longitude
    - Temporal Factors:

    - Time of Occurrence (You may need to preprocess this into a format that the model can use, such as timestamp or numerical representation of time.)
  - Depth of the Earthquake:
    - Depth
  - Seismic Energy Released:
    - This can be a measure of the energy released during the earthquake.

B. Data Representation: What is the data representation?

- The data is represented in a tabular format, where each row corresponds to a recorded earthquake event, and the columns represent features of interest.
- The values in each cell of the table contain the specific information related to the earthquake event at the intersection of the corresponding row and column. This tabular format allows for a structured representation of the data, making it suitable for analysis and application of predictive modeling techniques.

Data dictionary

- **Numerical columns:** 'latitude', 'longitude', 'depth', 'mag', 'nst', 'gap', 'dmin', 'rms', 'horizontalerror', 'depterror', 'magerror', 'magnst'
- **Non-numerical columns:** 'time', 'magtype', 'net', 'id', 'updated', 'place', 'type', 'status', 'locationsource', 'magsource'
- **Categorical columns and number of categories:**

	0
magtype	8
net	4
type	2
status	1
locationsource	1
magsource	2

- **Data types, count, and size:**

Column	Data Type	Count	Data Size
time	object	22123	486706

latitude	float64	22123	486706
longitude	float64	22123	486706
depth	float64	22123	486706
mag	float64	22123	486706
magtype	object	22123	486706
nst	float64	22123	486706
gap	float64	22123	486706
dmin	float64	22123	486706
rms	float64	22123	486706
net	object	22123	486706
id	object	22123	486706
updated	object	22123	486706
place	object	22123	486706
type	object	22123	486706
horizontalerror	float64	22123	486706
deptherror	float64	22123	486706
magerror	float64	22123	486706
magnst	int64	22123	486706
status	object	22123	486706
locationsource	object	22123	486706
magsource	object	22123	486706

- **Descriptive Statistics:**

	latitude	longitude	depth	mag	nst	gap	dmin	rms	horizontalerror	depterror	magerror	magnst
count	22123.000000	22123.000000	22123.000000	22123.000000	22123.000000	22123.000000	22123.000000	22123.000000	22123.000000	22123.000000	22123.000000	22123.000000
mean	35.550333	140.915084	58.762230	4.921088	76.824617	106.778920	2.563600	0.847942	6.443357	6.003373	0.151051	155.434525
std	4.922796	3.981671	90.882413	0.492188	86.643993	37.942867	1.957825	0.235881	3.554477	4.395367	0.098761	143.230679
min	24.862900	128.936000	0.000000	4.500000	4.000000	8.000000	0.038000	0.040000	0.000000	0.000000	0.010000	1.000000
25%	31.386000	140.127550	21.300000	4.600000	20.714165	88.000000	2.086011	0.700000	4.170000	1.947000	0.068000	32.000000
50%	36.318000	141.795400	33.740000	4.800000	41.000000	106.700000	2.236286	0.840000	6.590000	5.857000	0.133000	116.000000
75%	39.270400	143.068500	52.500000	5.100000	114.000000	127.800000	2.385836	1.000000	8.300000	8.400000	0.208000	244.000000
max	43.580000	152.222000	683.360000	9.100000	929.000000	303.000000	43.302000	1.880000	20.500000	38.200000	0.558000	706.000000

C. Output: What are you trying to predict? Define the output clearly.

- The goal is to predict the magnitude of the next earthquake. The target variable for our predictive modeling task is the magnitude of the earthquake.
- The model may reveal which features have a significant impact on the predicted earthquake magnitude. This can provide insights into the factors that contribute most to seismic activity.
- Predicting the magnitude of an earthquake can contribute to the development of early warning systems. While it might not be possible to predict earthquakes with perfect accuracy, identifying patterns that precede larger magnitudes can provide valuable seconds to minutes of warning, allowing for emergency response actions and potentially saving lives.
- Understanding the potential magnitude of earthquakes in a region is crucial for risk assessment and mitigation strategies. Engineers and city planners can use this information to design structures that can withstand seismic activity, and authorities can establish building codes and regulations to minimize the impact of earthquakes on communities.

2. Predictive Models

A. What are the methods? Give a general introduction of the methods with references

Gradient Boosting: Gradient boosting is a powerful machine learning technique used for building predictive models, particularly in regression and classification tasks. It belongs to the ensemble learning methods, where multiple models are combined to create a stronger model. Unlike bagging methods like Random Forest, which build independent models in parallel, gradient boosting builds models sequentially, with each new model focusing on correcting the errors made by the previous ones.

- **Basic Idea**: Gradient boosting builds an ensemble of weak learners (often decision trees) in a sequential manner, where each new weak learner is trained to correct the errors of the ensemble so far. It combines these weak learners to create a strong predictive model.
- **Sequential Learning**: Unlike bagging methods, which build models independently, gradient boosting builds models sequentially. Each new model is trained on the residuals (the differences between the actual and predicted values) of the previous models.
- **Loss Function Optimization**: Gradient boosting minimizes a loss function, such as mean squared error for regression or cross-entropy loss for classification, by iteratively adding new models that reduce the loss.
- **Gradient Descent**: The "gradient" in gradient boosting refers to the gradient of the loss function with respect to the predictions of the current ensemble. The

new weak learner is trained to minimize the loss by following the negative gradient direction.

- **Regularization**: Gradient boosting often includes regularization techniques to prevent overfitting, such as tree constraints (e.g., maximum depth, minimum samples per leaf), shrinkage (learning rate), and subsampling of the data.
- **Hyperparameter Tuning:** Gradient boosting involves tuning hyperparameters, such as the learning rate, number of trees, and tree-specific parameters, to optimize the model's performance.
- **Popular Implementations**: There are several popular implementations of gradient boosting, including XGBoost, LightGBM, and CatBoost, which are widely used in practice due to their efficiency and effectiveness.

References:

- Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. Annals of statistics, 1189-1232.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.
- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794).
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Yu, T. (2017). LightGBM: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems (pp. 3146-3154).
- Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. In Advances in Neural Information Processing Systems (pp. 6638-6648).

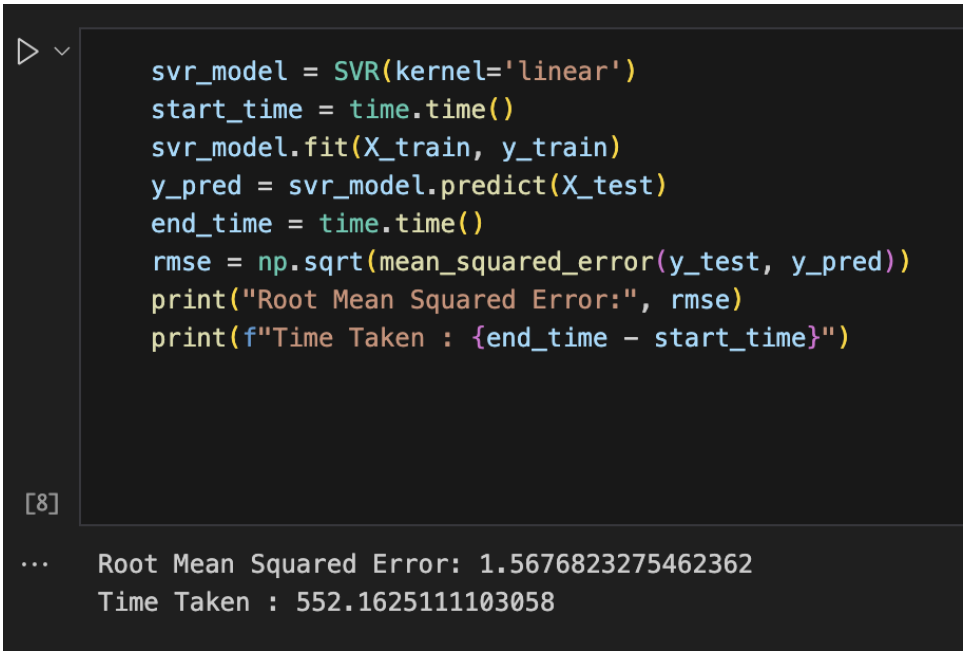
Neural Networks: Neural networks are computational models inspired by the structure and functioning of the human brain, capable of learning complex patterns and relationships from data. They consist of interconnected nodes (neurons) organized into layers, with each layer processing information and passing it to the next layer. During training, neural networks adjust their weights and biases based on the error between predicted and true outputs, using optimization algorithms like gradient descent. Activation functions introduce non-linearity to the network, allowing it to learn complex relationships. Various types of neural networks exist, including feedforward neural networks (FNN), convolutional neural networks (CNN), and recurrent neural networks (RNN), each suited to different types of data and tasks such as image recognition, natural language processing, and sequential data analysis. Deep learning, a subfield of neural networks, has revolutionized many fields due to its ability to automatically learn hierarchical representations of data.

Support Vector Machines (SVMs): SVMs are powerful supervised machine learning models used for classification and regression tasks. They are particularly effective in high-dimensional spaces and when the number of dimensions exceeds the number of samples. SVMs are widely used in fields such as bioinformatics, text mining, image recognition, and more.

At its core, an SVM performs classification by finding the hyperplane that best separates different classes in feature space. This hyperplane is chosen in such a way that it maximizes the margin between the classes, which is the distance between the hyperplane and the nearest data point from each class, also known as support vectors.

B. Describe the methods with a pseudo code using the definitions in Section 1.

- **Support Vector Machines (SVM):**
    a. Initialize the Support Vector Machines model.
    b. Define the model with kernel like linear and rbf.
    c. Fit the model on the training data using these kernels differently.
    d. Predict the target variable on the test data.
    e. Evaluate the model performance using root mean squared error (RMSE) on the test set.



```
svr_model = SVR(kernel='linear')
start_time = time.time()
svr_model.fit(X_train, y_train)
y_pred = svr_model.predict(X_test)
end_time = time.time()
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", rmse)
print(f"Time Taken : {end_time - start_time}")
```

```
[8]
...  Root Mean Squared Error: 1.5676823275462362
     Time Taken : 552.1625111103058
```

C.

- **Gradient Boosting Regression (GBR):**
    - Initialize the GradientBoostingRegressor model.
    - Define a parameter grid for hyperparameter tuning including:
        - Number of estimators (trees) in the ensemble: [50, 100, 150]
        - Learning rate: [0.01, 0.1, 0.2]
        - Maximum depth of the trees: [3, 5, 7]
    - Perform grid search cross-validation to find the best combination of hyperparameters.
    - Fit the model on the training data using the best hyperparameters.
    - Predict the target variable on the test data.
    - Evaluate the model performance using root mean squared error (RMSE) on the test set.

```
gb_model = GradientBoostingRegressor()
param_grid_gb = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}
start_time = time.time()
grid_search_gb = GridSearchCV(gb_model, param_grid_gb, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_gb.fit(X_train, y_train)
end_time = time.time()
best_gb_model = grid_search_gb.best_estimator_
gb_pred = best_gb_model.predict(X_test)
gb_mse = mean_squared_error(y_test, gb_pred)
print("Best Hyperparameters:")
print(grid_search_gb.best_params_)
print(f"Mean Squared Error on Test Set: {gb_mse}")
print(f"Time taken for Grid Search and Training: {end_time - start_time} seconds")
```

```
Best Hyperparameters:
{'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 150}
Mean Squared Error on Test Set: 0.010122116172053454
Time taken for Grid Search and Training: 71.21306729316711 seconds
```

- **Neural Networks (using tensorflow):**
    - Initialize a Sequential model in TensorFlow.
    - Add multiple dense layers with ReLU activation:
        - Input layer with 200 neurons.
        - Hidden layers with 200, 150, 100, 50 and 25 neurons respectively.
        - Output layer with 1 neuron and linear activation.
    - Compile the model using mean squared error as loss function and Adam optimizer with learning rate 0.001.
    - Train the model on the training data for 300 epochs with a batch size of 32.
    - Predict the target variable on the test data.
    - Evaluate the model performance using root mean squared error (RMSE) on the test set.

```
nn_model = Sequential()
nn_model.add(Dense(200, input_dim=len(features), activation='relu'))
nn_model.add(Dense(200, activation='relu'))
nn_model.add(Dense(150, activation='relu'))
nn_model.add(Dense(150, activation='relu'))
nn_model.add(Dense(100, activation='relu'))
nn_model.add(Dense(100, activation='relu'))
nn_model.add(Dense(50, activation='relu'))
nn_model.add(Dense(50, activation='relu'))
nn_model.add(Dense(25, activation='relu'))
nn_model.add(Dense(25, activation='relu'))
nn_model.add(Dense(1, activation='linear'))
nn_model.compile(loss='mean_squared_error', optimizer=Adam(learning_rate=0.001))
start_time = time.time()
nn_model.fit(X_train, y_train, epochs=300, batch_size=32, verbose=2)
end_time = time.time()
nn_pred = nn_model.predict(X_test)
nn_mse = mean_squared_error(y_test, nn_pred)
print(f"Neural Network (using TensorFlow) MSE: {nn_mse}")
print(f"Time taken for training: {end_time - start_time} seconds")
```

D. Justify the choice of the method.

- **Gradient Boosting Regressor (GBR):**
    - Strengths:
        - GBR is an ensemble method that builds trees sequentially, where each tree corrects the errors of the previous one, leading to strong predictive performance.
        - It handles well both numerical and categorical data without requiring data preprocessing.
        - GBR naturally handles interactions and non-linear relationships between features.

        - It automatically handles feature selection and can work with missing data.
    - Justification:
        - GBR is a popular choice for regression tasks due to its high predictive accuracy and robustness against overfitting.
        - Grid search is used to tune hyperparameters, ensuring optimal performance for the given data.
- **Neural Network (using TensorFlow):**
    - Strengths:
        - Neural networks are highly flexible models capable of learning complex patterns in data.
        - They can capture intricate relationships between features and the target variable through multiple hidden layers.
        - TensorFlow provides a powerful framework for building and training neural networks efficiently.
    - Justification:
        - Neural networks are suitable for tasks where the relationships between features and the target variable are complex and non-linear.
        - In this case, the architecture of the neural network consists of multiple hidden layers, allowing it to learn intricate patterns in the seismic data.
        - The Adam optimizer with a learning rate of 0.001 is chosen for efficient convergence during training.
- **Support Vector Machines:**
    - Strengths:
        - Can handle both linear and nonlinear classification and regression tasks through the use of different kernel functions, making them suitable for a wide range of real-world problems.
        - They are robust against overfitting, especially when using appropriate regularization techniques, ensuring stable performance across different datasets.
        - SVMs use only a subset of training points (support vectors) to define the decision boundary, making them memory efficient, particularly for large datasets.
        - SVMs have a solid theoretical foundation in convex optimization, ensuring convergence to the global minimum and providing a clear understanding of their behavior, which justifies their use in various machine learning applications.
    - Justification:
        - SVMs exhibit strong predictive accuracy, making them a popular choice for classification and regression tasks where high performance is crucial.
        - SVMs naturally handle high-dimensional data and complex relationships between features, allowing them to effectively capture patterns and make accurate predictions in real-world scenarios.
        - SVMs have been successfully applied to various domains, including image classification, text categorization, and bioinformatics, demonstrating their versatility and effectiveness in solving diverse machine learning problems.

- **Model Evaluation:**
    - The choice of models is further justified by evaluating them using metrics such as root mean squared error (RMSE) on the test set.

- Both models are trained and evaluated on the same data set, allowing for a fair comparison of their performance.
- The models' performance is assessed based on how well they minimize the prediction error and generalize to unseen data.

Overall, the choice of Gradient Boosting Regressor and Neural Network is justified by their ability to capture complex patterns in the data and their strong predictive performance, as evidenced by the evaluation metrics. Moreover, the use of grid search for hyperparameter tuning ensures that the models are optimized for the given dataset. We also wanted to experiment with the basics of neural networks, and that is why we deliberately included this as one of the models that we wanted to include as predictive models for this.

3. Evaluations

A. What metrics do you use for evaluation?
   - The choice of models is further justified by evaluating them using metrics such as root mean squared error (RMSE) on the test set.
   - Both models are trained and evaluated on the same data set, allowing for a fair comparison of their performance.
   - The models' performance is assessed based on how well they minimize the prediction error and generalize to unseen data.
   - Neural Nets:

```
Epoch 300/300
554/554 - 1s - loss: 0.0377 - 1s/epoch - 2ms/step
139/139 [==============================] - 0s 1ms/step
Neural Network (using TensorFlow) RMSE: 0.14008668543210404
Time taken for training: 303.83577585220337 seconds
```

   - Gradient Boosting Regressor:

```
Best Hyperparameters:
{'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 150}
Root Mean Squared Error (RMSE) on Test Set: 0.10038132843868584
Time taken for Grid Search and Training: 91.0340564250946 seconds
```

   - Support Vector Machines (SVM)

```
...  Root Mean Squared Error: 1.5676823275462362
     Time Taken : 552.1625111103058
```

   - From the above results, we can see that Neural Nets has taken more time than Gradient Boosting Regressor and SVM. The time taken is like around 3 times faster than Neural Nets and 5 times faster than SVM. Although the RMSE of Neural Nets and Gradient Boosting are pretty much the same, the SVM's RMSE is higher.

B. What is your ground truth?
   - The ground truth in our analysis refers to the observed or actual values of the target variable in our dataset. In the context of our predictive modeling task, the ground truth represents the true outcomes or measurements that we aim to predict using our model. It serves as the basis for evaluating the performance of our model and assessing how well it aligns with the actual data. Essentially, the ground truth provides a reference point against which we compare the predictions generated by our model to determine its

accuracy and effectiveness. To be more precise, the test values of Magnitude are our ground truth values, which are the values of magnitudes of earthquakes that actually happened on that particular area.

C. Discuss the performance and the limitations of the method.

- **Gradient Boosting Regressor:**
  - Sensitivity to Noisy Data:
    - Like other tree-based algorithms, Gradient Boosting Regressor can be sensitive to noisy data and outliers. It may try to fit the noise in the data, leading to overfitting.
  - Computational Complexity:
    - Gradient Boosting Regressor can be computationally expensive and time-consuming to train, especially when dealing with large datasets or complex models with many trees.
  - Hyperparameter Sensitivity:
    - Gradient Boosting Regressor has several hyperparameters that need to be tuned, such as the learning rate, number of trees, tree depth, and regularization parameters. Finding the optimal combination of hyperparameters can require extensive experimentation and computational resources.
  - Potential for Overfitting:
    - Gradient Boosting Regressor is susceptible to overfitting, especially when the model is too complex or when the learning rate is too high. Overfitting can occur if the model captures noise or irrelevant patterns in the training data, leading to poor generalization performance on unseen data.
  - Limited Interpretability:
    - While Gradient Boosting Regressor can provide accurate predictions, it is often considered a black-box model, making it challenging to interpret how individual features contribute to the predictions. Understanding the internal workings of the model and extracting insights from it may be difficult.
  - Scalability:
    - Gradient Boosting Regressor may not scale well to very large datasets or distributed computing environments. Training the model on large datasets may require significant computational resources and memory.
  - Gradient Descent Approach:
    - Gradient Boosting Regressor relies on the gradient descent optimization algorithm, which may get stuck in local minima or plateaus, especially if the objective function is non-convex. This can affect the model's ability to find the optimal global solution.
- **Neural Nets:**
  - Complexity and Interpretability:
    - Neural Networks, especially deep neural networks, are highly complex models with multiple layers of interconnected neurons. While they can capture complex patterns in the data, understanding how the model makes predictions can be challenging due to their black-box nature.
  - Overfitting:

- Neural Networks are prone to overfitting, especially when the model architecture is complex or when the training dataset is small. Techniques such as regularization, dropout, and early stopping are commonly used to mitigate overfitting in neural networks.

- Training Time and Computational Resources:
  - Training deep neural networks can be computationally intensive and time-consuming, especially for large datasets or complex architectures. Training neural networks may require access to high-performance computing resources such as GPUs or TPUs.
- Data Requirements:
  - Neural Networks typically require a large amount of training data to learn complex patterns effectively. Training a neural network with insufficient data may result in poor generalization performance and overfitting.
- Hyperparameter Tuning:
  - Neural Networks have various hyperparameters, including the number of layers, the number of neurons per layer, activation functions, and learning rates. Tuning these hyperparameters to optimize performance requires experimentation and computational resources.

SVM:

- Insufficient Data:
  - SVMs perform better when trained on a large amount of data. If the dataset is too small or lacks diversity, the model may fail to capture the underlying patterns in the data.
- Imbalanced Data:
  - When classes in the dataset are imbalanced (i.e., one class significantly outnumbers the others), SVM may prioritize the majority class and perform poorly on the minority class(es).
- Non-linear Relationships:
  - SVM is inherently a linear classifier. If the relationship between features and the target variable is non-linear, SVM may fail to capture this complex relationship without appropriate kernel functions or feature engineering.
- Incorrect Choice of Kernel:
  - The choice of kernel function in SVM significantly impacts its performance. If the wrong kernel is chosen or the hyperparameters of the kernel are not tuned properly, the model may fail to generalize well to unseen data.
- Overfitting or Underfitting:
  - SVM models are susceptible to overfitting if the model complexity is too high or underfitting if the model complexity is too low. Regularization parameters (C in linear SVM and C, gamma in kernel SVM) need to be tuned properly to avoid these issues.
- Noise in Data:
  - If the dataset contains noise or irrelevant features, SVM may learn from these noisy patterns and fail to generalize well to unseen data.
- Feature Scaling:

    - SVM performance can be sensitive to the scale of features. If features are not properly scaled, with some having much larger ranges than others, it can affect the decision boundary and lead to suboptimal performance.
- Outliers:
    - Outliers can significantly impact the decision boundary learned by SVM. If outliers are not properly handled or removed, they can skew the decision boundary and lead to poor predictions.
- Model Complexity:
    - SVM may struggle with highly complex datasets where the decision boundary is not easily separable by a hyperplane. In such cases, more sophisticated models or ensemble methods may be more suitable.

- Data Leakage:
    - If there is any form of data leakage present in the dataset, where information from the test set leaks into the training process, the model may perform well on the test set but fail to generalize to new, unseen data.

Appendix

[Addition materials that are not included in the above sections.]

Table of Contributions

The table below identifies contributors to various sections of this document.

	Section	Writing	Editing
1	**Predictive Modeling Problem Definition**	**Alex**	**Alex, Vijval**
2	**Predictive Models**	**Aakhil**	**Priyanka, Vijval**
3	**Evaluations**	**Priyanka**	**Aakhil, Alex**
4	**Appendix**	**Vijval**	**Priyanka, Aakhil**

Grading

The grade is given based on quality, clarity, presentation, completeness, and writing of each section in the report. This is the grade of the group. Individual grades will be assigned at the end of the term when peer reviews are collected.