Indian Institute of Technology, Dharwad

॥ सा विद्या या विमुक्तये ॥
ಭಾ.ತಂ.ಸಂ. ಧಾರವಾಡ
भा. प्रौ. सं. धारवाड
I.I.T. DHARWAD

CS209 : Artificial Intelligence
And
CS214 : Artificial Intelligence Laboratory
Project Report : **Team 23: Oil Spill Detection Using Satellite Image Features**

Course Instructor:
Dr. Dileep A.D.
Mentor Name:
Neha R P

Submitted by:
1. Darisi Surya Teja (MC23BT012)
2. Abhinav Shankar (IS22BM001)
3. Juhi Kumari (MC23BT002)
4. Mayank Meena (MC23BT016)

Contents

List of Figures

List of Tables

1 Introduction

The objective of this Project is to develop a machine learning model that can classify whether a given image patch contains an oil spill or not.

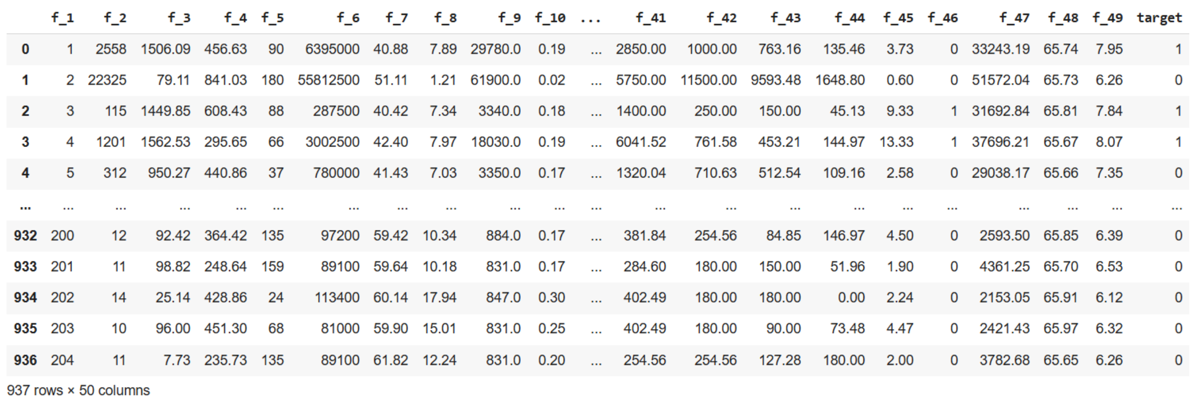
1.1 How does oil impact marine life?

If an oil spill happens in an area with wildlife, the damage can be significant. Oil spills are harmful to marine birds and mammals as well as fish and shellfish. Oil destroys the *insulating ability of fur on mammals and impacts the water repelling qualities of a bird's feathers*[2], without the insulation or water repelling qualities mammals and birds can die from hypothermia. Dolphins and whales can inhale oil, which has an impact on their immune system and can impact reproduction. While fish and shellfish aren't immediately impacted, because oil floats on water, as the oil mixes and sinks, fish can experience impacted growth, enlarged livers, fin erosion and a reduction in reproductive capabilities. In fish and shellfish, the impact can also be lethal, when it is not lethal, they are often no longer safe for human consumption. [1]

Hence, the ability to detect oil spills is crucial for environmental monitoring, marine ecosystem protection, and early response to accidental or illegal oil dumping.

1.2 Description of the Dataset

The dataset was developed using satellite images of the ocean, some of which contain oil spills while others do not. The images were divided into smaller sections (patches) and processed using computer vision algorithms to extract feature vectors that describe the contents of each patch. Each feature vector contains numerical values describing the visual characteristics of an image patch.



	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_10	...	f_41	f_42	f_43	f_44	f_45	f_46	f_47	f_48	f_49	target
0	1	2558	1506.09	456.63	90	6395000	40.88	7.89	29780.0	0.19	...	2850.00	1000.00	763.16	135.46	3.73	0	33243.19	65.74	7.95	1
1	2	22325	79.11	841.03	180	55812500	51.11	1.21	61900.0	0.02	...	5750.00	11500.00	9593.48	1648.80	0.60	0	51572.04	65.73	6.26	0
2	3	115	1449.85	608.43	88	287500	40.42	7.34	3340.0	0.18	...	1400.00	250.00	150.00	45.13	9.33	1	31692.84	65.81	7.84	1
3	4	1201	1562.53	295.65	66	3002500	42.40	7.97	18030.0	0.19	...	6041.52	761.58	453.21	144.97	13.33	1	37696.21	65.67	8.07	1
4	5	312	950.27	440.86	37	780000	41.43	7.03	3350.0	0.17	...	1320.04	710.63	512.54	109.16	2.58	0	29038.17	65.66	7.35	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
932	200	12	92.42	364.42	135	97200	59.42	10.34	884.0	0.17	...	381.84	254.56	84.85	146.97	4.50	0	2593.50	65.85	6.39	0
933	201	11	98.82	248.64	159	89100	59.64	10.18	831.0	0.17	...	284.60	180.00	150.00	51.96	1.90	0	4361.25	65.70	6.53	0
934	202	14	25.14	428.86	24	113400	60.14	17.94	847.0	0.30	...	402.49	180.00	180.00	0.00	2.24	0	2153.05	65.91	6.12	0
935	203	10	96.00	451.30	68	81000	59.90	15.01	831.0	0.25	...	402.49	180.00	90.00	73.48	4.47	0	2421.43	65.97	6.32	0
936	204	11	7.73	235.73	135	89100	61.82	12.24	831.0	0.20	...	254.56	254.56	127.28	180.00	2.00	0	3782.68	65.65	6.26	0

937 rows × 50 columns

Figure 1.1: The Dataset

2 Methodology

The 'target' attribute, which needs to be predicted, consists of two classes, namely:
'0' : Non-Spill – Negative class (majority), representing clean ocean patches.
'1' : Oil Spill – Positive class (minority), representing patches containing oil spills.

2.1 Handling Class Imbalance

The given dataset is greatly biased towards Non-Spill i.e., target variable '0'. To handle this bias effectively, three different techniques have been used:

- **SMOTE (Synthetic Minority Oversampling Technique)** works by generating synthetic examples for the minority class by interpolating between existing instances, which helps improve the performance of models on imbalanced data.
- **Undersampling** is a technique to address class imbalance in data sets by reducing the number of instances in the majority class, so that it equals the number of instances in minority class.
- **Oversampling** increases the number of instances in the minority class by duplicating existing samples, so that it equals the number of instances in majority class.

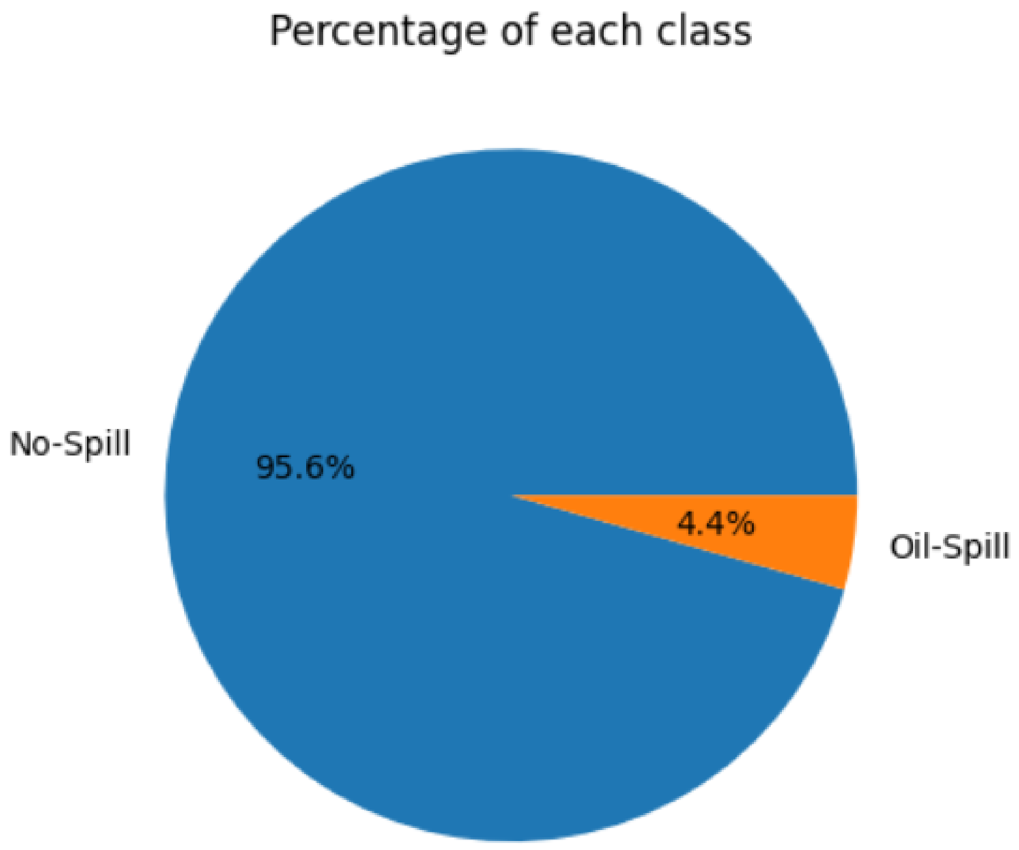


Figure 2.1: Division of Classes in the Original Dataset

2.2 Evaluation Metrics

The performance of each model was evaluated using the following metrics:

- **Accuracy:** Accuracy measures the proportion of correctly classified samples out of the total samples:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Samples}}$$

It gives a general measure of how well the model performs overall.

- **Precision (Positive Predictive Value):** Precision evaluates the proportion of correctly predicted positive cases out of all predicted positive cases:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

  This metric is critical when the cost of false positives is high.

- **Recall (Sensitivity):** Recall measures the ability of the model to identify actual positive cases:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

  It is essential in scenarios where missing positive cases is costly.

- **F1-Score:** The F1-score provides a harmonic mean of precision and recall:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

  It balances the trade-off between precision and recall, especially when classes are imbalanced.

- **Equal Error Rate (EER):** EER is a critical metric in binary classification tasks that is useful when you want a balanced decision point and can't afford to favor one error over another. It is the point at which the False Acceptance Rate (FAR) equals the False Rejection Rate (FRR). A lower EER indicates better model performance in balancing false positives and false negatives:

$$\text{EER} : \text{FAR} = \text{FRR}$$

  EER is relevant when both false alarms and missed detections are costly. In oil spill detection, where both can have financial or environmental impacts, EER gives you a fair and interpretable balance point.

- **Confusion Matrix:** A confusion matrix visually represents the counts of true positives, true negatives, false positives, and false negatives. It helps identify the strengths and weaknesses of the model by showing misclassifications explicitly.

2.3 Code Snippet

The following code snippet has been used to fit the train data and predict the results for the validation data to hyper-tune the parameters.

```python
def model_evaluate(model, model_name):
    model.fit(X_train_smote, y_train_smote)
    y_pred = model.predict(X_val_scaled)
    if hasattr(model, 'predict_proba'):
      y_prob = model.predict_proba(X_val_scaled)[:,1]
    else:
      decision_scores = model.decision_function(X_val_scaled)
      y_prob = (decision_scores - decision_scores.min()) / (
         decision_scores.max() - decision_scores.min())

    acc = accuracy_score(y_val, y_pred)
    precision = precision_score(y_val, y_pred)
    recall = recall_score(y_val, y_pred)
    f1 = f1_score(y_val, y_pred)
    roc_auc = roc_auc_score(y_val, y_prob)
    fpr, tpr, thresholds = roc_curve(y_val, y_prob)
    auc_score = auc(fpr, tpr)
    eer = compute_eer(y_val, y_prob)

    print(f"\n{model_name}ValidationResults")
    print(f"Accuracy:{100*acc:.4f}")
    print(f"Precision:{precision:.4f}")
    print(f"Recall:{recall:.4f}")
    print(f"F1-score:{f1:.4f}")
    print(f"ROC-AUC:{roc_auc:.4f}")
    print(classification_report(y_val, y_pred))

    cm = confusion_matrix(y_val, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'ConfusionMatrix-{model_name}')
    plt.show()

    models.append(model_name)
    accuracy_smote.append(round(acc*100, 2))
    precision_smote.append(precision)
    recall_smote.append(recall)
    f1_smote.append(f1)
    fpr_smote.append(fpr)
    tpr_smote.append(tpr)
    auc_smote.append(auc_score)
    eer_smote.append(eer)
```

2.4 List of Classification Models Used

2.4.1 K - Nearest Neighbors

The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. It is one of the popular and simplest classification and regression classifiers used in machine learning today.[3]

The classification results show that SMOTE with $k = 1$ achieved the highest accuracy of 94.33% and the lowest Equal Error Rate (EER) of 0.03. In comparison, Undersampling with ($k = 5$) resulted in an accuracy of 85.82% and a higher EER of 0.26, while

Oversampling with $k = 1$ yielded the same accuracy of 85.82% but a much lower EER of 0.0074. These results suggest that SMOTE provides a more balanced and effective improvement in both accuracy and error rate.

Metric	SMOTE			Undersampling			Oversampling		
	Class 0	Class 1	Avg	Class 0	Class 1	Avg	Class 0	Class 1	Avg
Precision	0.98	0.38	0.37	0.98	0.15	0.15	0.98	0.15	0.15
Recall	0.96	0.50	0.50	0.87	0.50	0.50	0.87	0.50	0.50
F1-Score	0.97	0.43	0.42	0.92	0.23	0.23	0.92	0.23	0.23

Table 2.1: Class-Wise Metrics for KNN

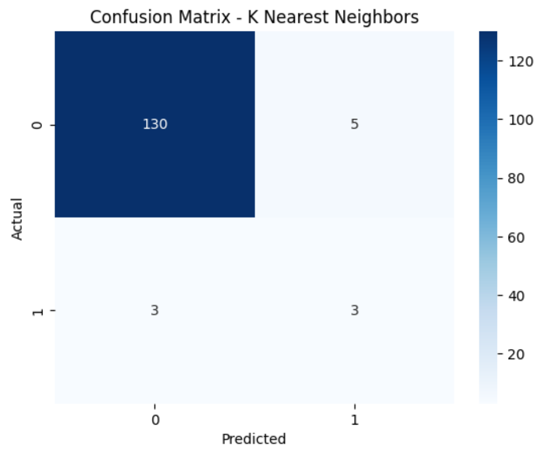


Figure 2.2: KNN (SMOTE)

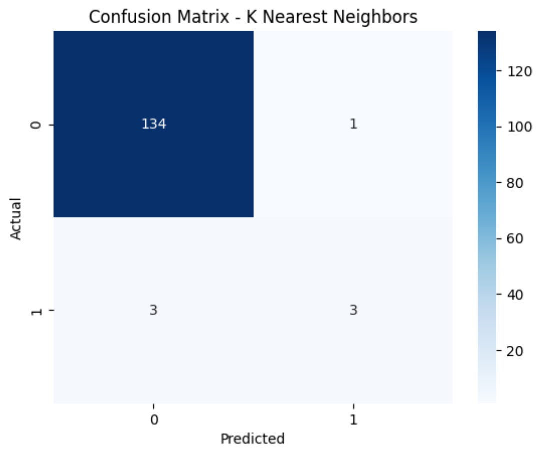


Figure 2.3: KNN (Oversampling)

2.4.2 Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The Gaussian variant models continuous features assuming a normal distribution.[4]

Metric	SMOTE			Undersampling			Oversampling		
	Class 0	Class 1	Avg	Class 0	Class 1	Avg	Class 0	Class 1	Avg
Precision	0.97	0.05	0.049	0.98	0.15	0.150	0.98	0.15	0.150
Recall	0.28	0.83	0.833	0.87	0.50	0.500	0.87	0.50	0.500
F1-Score	0.44	0.09	0.093	0.92	0.23	0.231	0.92	0.23	0.231

Table 2.2: Class-Wise Metrics for Naive Bayes

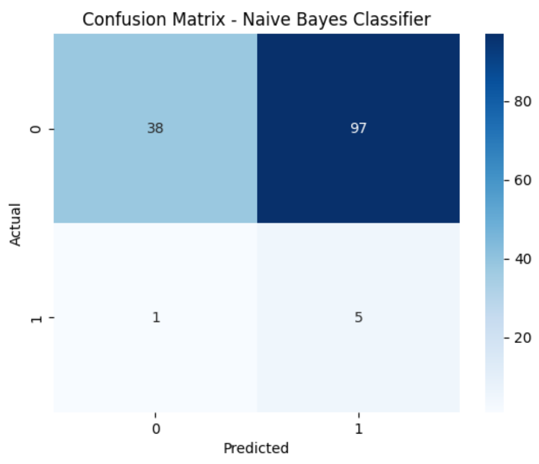


Figure 2.4: Naive Bayes (SMOTE)

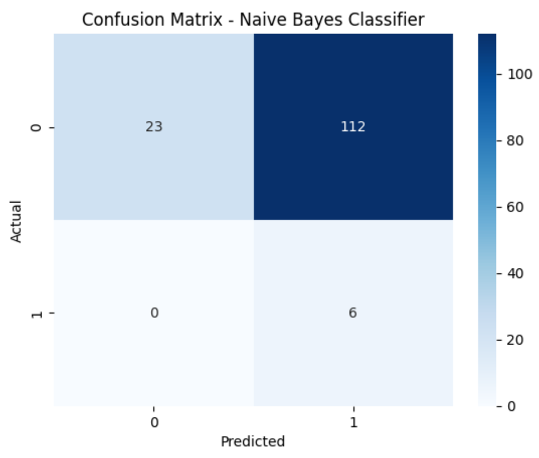


Figure 2.5: Naive Bayes (Oversampling)

2.4.3 Linear SVM

Linear Support Vector Machine (SVM) is a supervised classifier that finds the optimal hyperplane separating the data by maximizing the margin. It is effective in high-dimensional spaces and works well when the data is linearly separable. The best hyperparameter for the Linear SVM with SMOTE and OVERSAMPLING was found to be $C = 10$.

Metric	SMOTE			Undersampling			Oversampling		
	Class 0	Class 1	Avg	Class 0	Class 1	Avg	Class 0	Class 1	Avg
Precision	0.99	0.45	0.455	0.98	0.44	0.444	0.98	0.44	0.444
Recall	0.96	0.83	0.833	0.96	0.67	0.667	0.96	0.67	0.667
F1-Score	0.97	0.59	0.588	0.97	0.53	0.533	0.97	0.53	0.533

Table 2.3: Class-Wise Metrics for Linear SVM

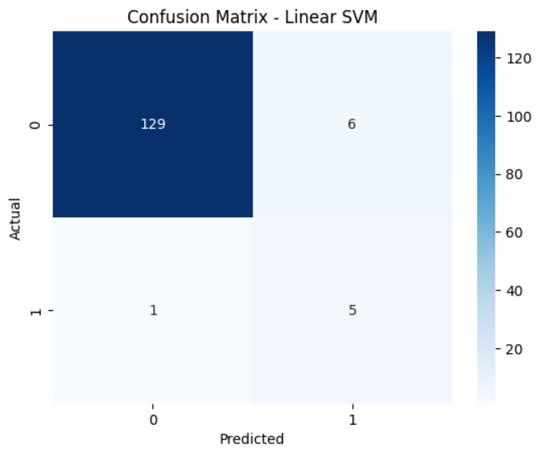


Figure 2.6: Linear SVM (SMOTE)

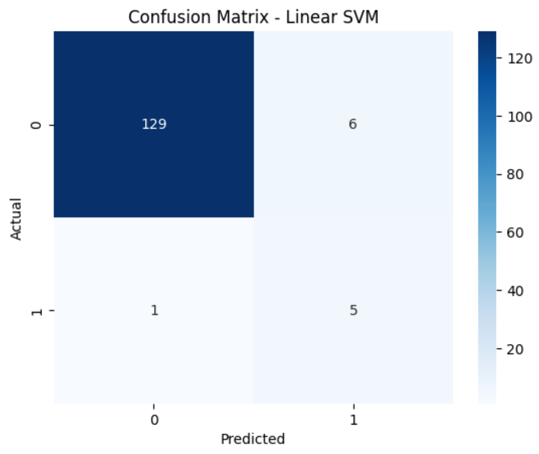


Figure 2.7: Linear SVM (Oversampling)

2.4.4 RBF SVM

The RBF (Radial Basis Function) kernel extends SVM to handle non-linearly separable data by transforming it into higher-dimensional space. It is powerful for modeling complex boundaries, but sensitive to hyperparameters like gamma and regularization. The best hyperparameter for the Linear SVM with SMOTE and OVERSAMPLING was found to be $C = 10$.

Metric	**SMOTE**			**Undersampling**			**Oversampling**		
	Class 0	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**
Precision	0.98	0.38	0.375	0.99	0.18	0.179	0.99	0.18	0.179
Recall	0.96	0.50	0.500	0.83	0.83	0.833	0.83	0.83	0.833
F1-Score	0.97	0.43	0.429	0.90	0.29	0.294	0.90	0.29	0.294

Table 2.4: Class-Wise Metrics for RBF SVM

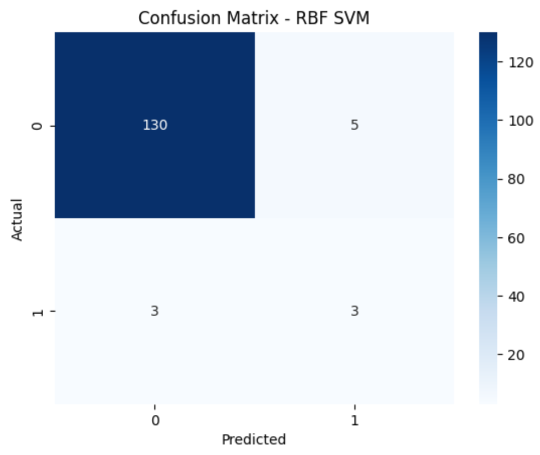


Figure 2.8: RBF (SMOTE)

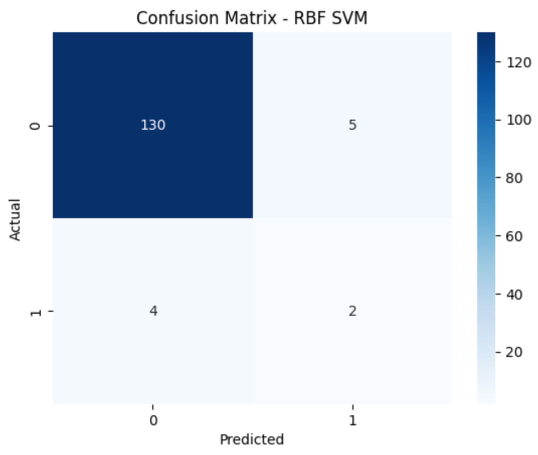


Figure 2.9: RBF (Oversampling)

2.4.5 Polynomial SVM

Polynomial SVM uses a polynomial kernel to create curved decision boundaries, suitable when interactions between features are non-linear. It introduces flexibility to the standard linear SVM but may overfit if the degree is too high.

The best degree turned out to be **3** when class imbalance is handled through SMOTE and **2** when handled with Oversampling. The regularization constant, C was found to be **0.01** in both the cases.

Metric	**SMOTE**			**Undersampling**			**Oversampling**		
	Class 0	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**
Precision	0.96	0.50	0.500	0.97	0.11	0.105	0.97	0.20	0.200
Recall	0.99	0.17	0.167	0.87	0.33	0.333	0.94	0.33	0.333
F1-Score	0.98	0.25	0.250	0.92	0.16	0.160	0.95	0.25	0.250

Table 2.5: Class-Wise Metrics for Polynomial SVM

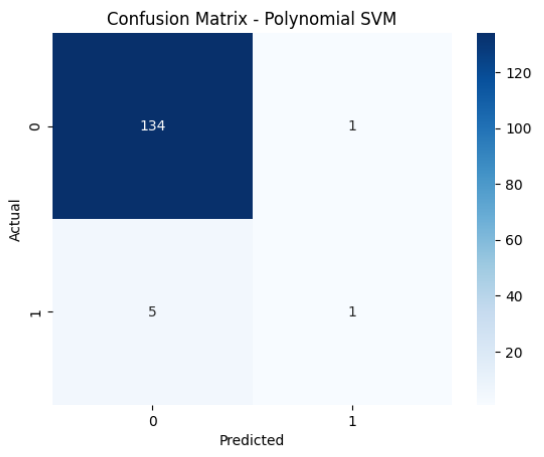


Figure 2.10: Polynomial SVM (SMOTE)

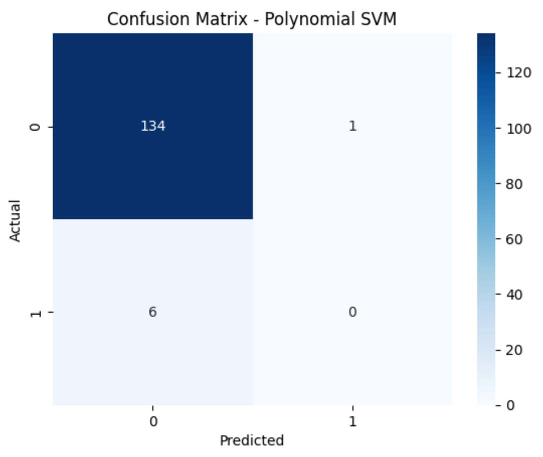


Figure 2.11: Polynomial SVM (Oversampling)

2.4.6 Random Forest

Random Forest is an ensemble method that constructs a collection of decision trees and aggregates their predictions. It is robust to noise, handles feature interactions, and is less prone to overfitting due to its averaging nature. It generally performs well with minimal parameter tuning.

Metric	**SMOTE**			**Undersampling**			**Oversampling**		
	Class 0	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**
Precision	0.97	0.33	0.333	0.97	0.20	0.200	0.97	0.20	0.200
Recall	0.97	0.33	0.333	0.94	0.33	0.333	0.94	0.33	0.333
F1-Score	0.97	0.33	0.333	0.95	0.25	0.250	0.95	0.25	0.250

Table 2.6: Class-Wise Metrics for Random Forest

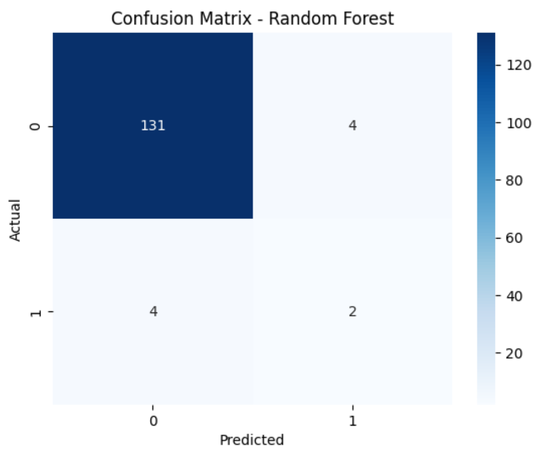


Figure 2.12: Random Forest (SMOTE)

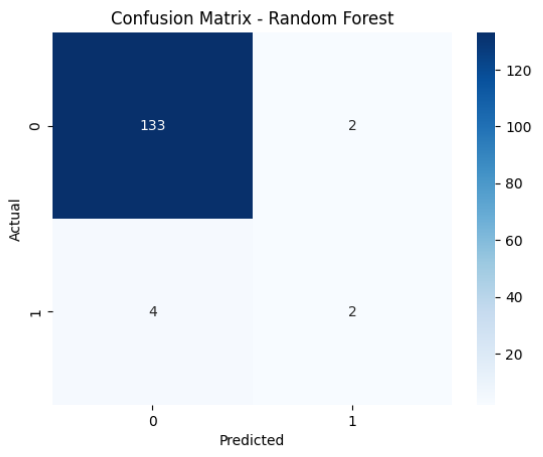


Figure 2.13: Random Forest (Oversampling)

2.4.7 Logistic Regression

Logistic Regression is a linear classification algorithm that models the probability of a binary outcome using the sigmoid (logistic) function. It assumes a linear relationship between input features and the log-odds of the target. It is simple, efficient, and interpretable, but may underperform when classes are not linearly separable.

Metric	SMOTE			Undersampling			Oversampling		
	Class 0	Class 1	Avg	Class 0	Class 1	Avg	Class 0	Class 1	Avg
Precision	0.99	0.31	0.313	0.97	0.20	0.200	0.97	0.20	0.200
Recall	0.92	0.83	0.833	0.94	0.33	0.333	0.94	0.33	0.333
F1-Score	0.95	0.45	0.455	0.95	0.25	0.250	0.95	0.25	0.250

Table 2.7: Class-Wise Metrics for Logistic Regression

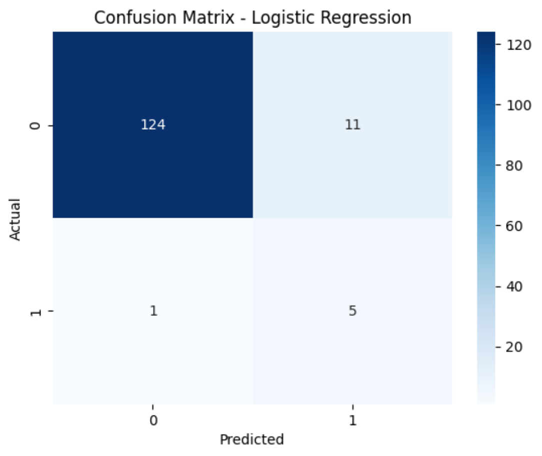


Figure 2.14: Logistic Regression (SMOTE)

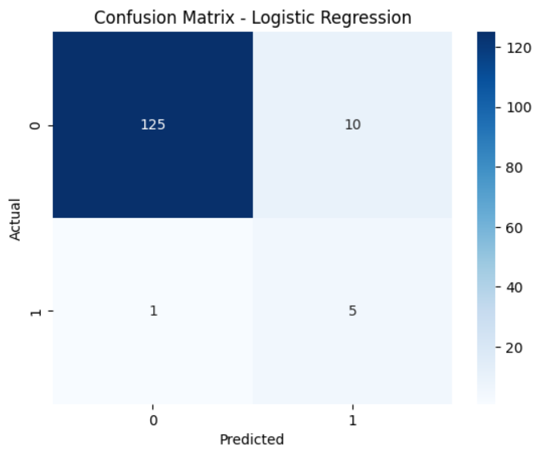


Figure 2.15: Logistic Regression (Oversampling)

2.4.8 Perceptron

The Perceptron is a binary linear classifier that updates weights based on misclassified examples using a thresholded activation function. It is one of the earliest neural models but performs well only when the data is linearly separable. For complex data, its performance is often outclassed by modern algorithms.

Metric	SMOTE			Undersampling			Oversampling		
	Class 0	Class 1	Avg	Class 0	Class 1	Avg	Class 0	Class 1	Avg
Precision	0.99	0.50	0.500	0.97	0.20	0.200	0.97	0.20	0.200
Recall	0.96	0.83	0.833	0.94	0.33	0.333	0.94	0.33	0.333
F1-Score	0.98	0.62	0.625	0.95	0.25	0.250	0.95	0.25	0.250

Table 2.8: Class-Wise Metrics for Perceptron Model

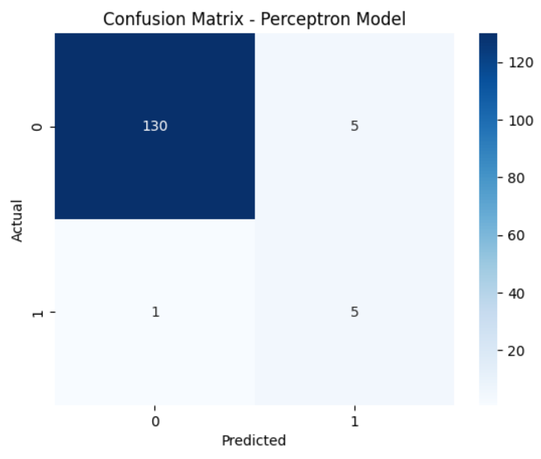


Figure 2.16: Perceptron (SMOTE)

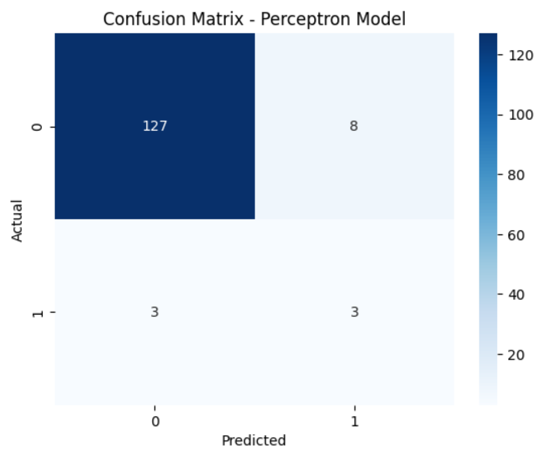


Figure 2.17: Perceptron (Oversampling)

2.4.9 Neural Network (Logistic)

The best hidden layer configuration for the Neural Network with logistic activation and **SMOTE was (100, 100)** The best hidden layer configuration for the Neural Network with logistic activation and **oversampling was (50, 50)**

Metric	**SMOTE**			**Undersampling**			**Oversampling**		
	Class 0	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**
Precision	0.97	0.18	0.181	0.98	0.12	0.121	0.97	0.20	0.200
Recall	0.93	0.33	0.333	0.79	0.67	0.667	0.94	0.33	0.333
F1-Score	0.95	0.24	0.235	0.87	0.21	0.205	0.95	0.25	0.250

Table 2.9: Class-Wise Metrics for Neural Network Model

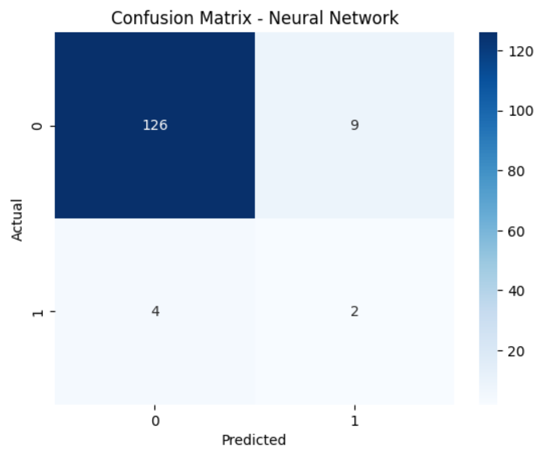


Figure 2.18: Neural Network (Logistic) (SMOTE)

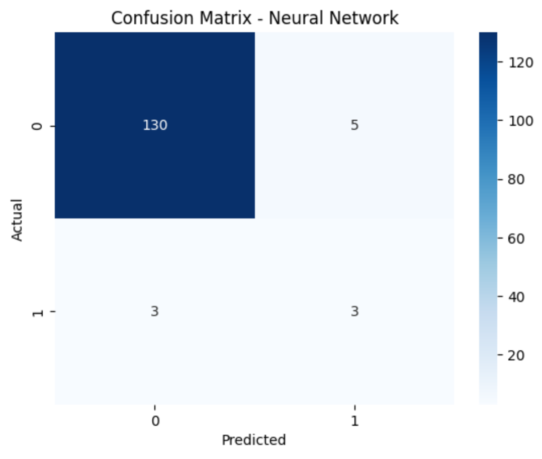


Figure 2.19: Neural Network (Logistic) (Oversampling)

2.4.10 Neural Network (Tanh)

- The best hidden layer configuration for the Neural Network with Tanh activation and **SMOTE was (100, 100)**.

- The best hidden layer configuration for the Neural Network with Tanh activation and **oversampling was (50, 50)** .

Metric	SMOTE			Undersampling			Oversampling		
	Class 0	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**
Precision	0.98	0.30	0.300	0.98	0.11	0.105	0.98	0.27	0.272
Recall	0.95	0.50	0.500	0.75	0.67	0.667	0.94	0.50	0.500
F1-Score	0.96	0.38	0.375	0.85	0.18	0.182	0.96	0.35	0.353

Table 2.10: Class-Wise Metrics for Neural Network (Tanh) Model

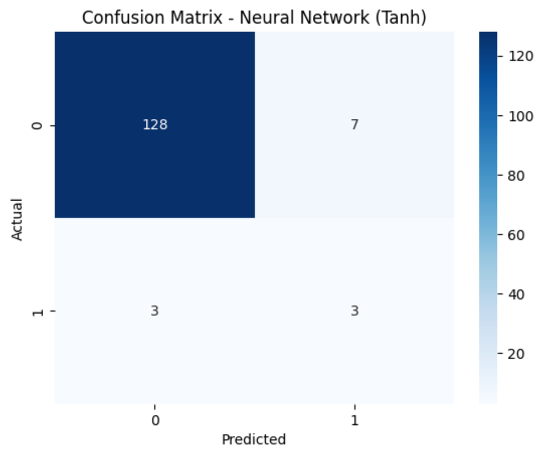


Figure 2.20: Neural Network (Tanh) (SMOTE)

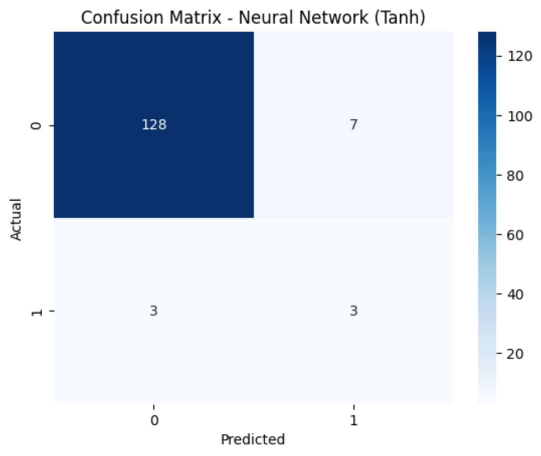


Figure 2.21: Neural Network (Tanh) (Oversampling)

2.4.11 Neural Network (ReLu)

Tuned Hyper-Parameters: The best hidden layer configuration turned out to be (50,50) irrespective of the technique used to balance the classes, when the activation function for the Neural Networks is ReLu.

Metric	SMOTE			Undersampling			Oversampling		
	Class 0	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**	**Class 0**	**Class 1**	**Avg**
Precision	0.98	0.38	0.375	0.98	0.12	0.121	0.97	0.20	0.200
Recall	0.96	0.50	0.500	0.79	0.67	0.667	0.94	0.33	0.333
F1-Score	0.97	0.43	0.429	0.87	0.21	0.205	0.95	0.25	0.250

Table 2.11: Class-Wise Metrics for Neural Network (ReLU) Model

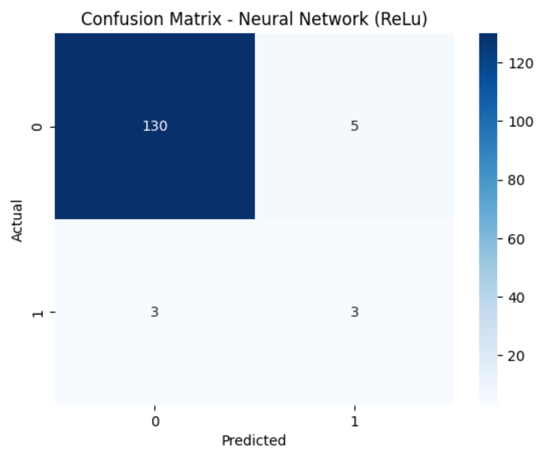


Figure 2.22: Neural Network (ReLu) (SMOTE)

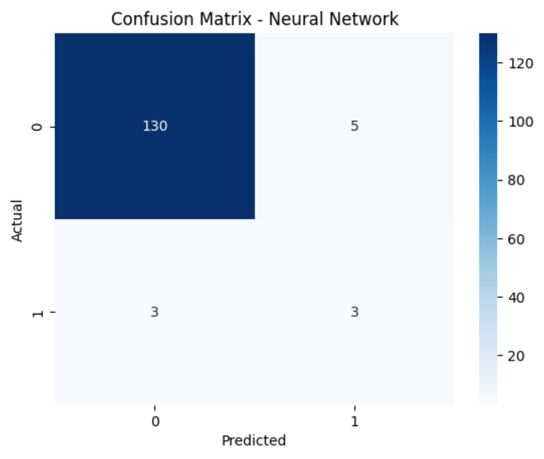


Figure 2.23: Neural Network (ReLu) (Oversampling)

2.5 Feature Importance Analysis

Figure 2.24 illustrates the feature importances derived from the trained Random Forest classifier. The model identifies `f47` as the most influential feature by a substantial margin, followed by `f1`, `f34`, and `f35`. This insight provides interpretability into the decision-making process of the model, allowing us to prioritize key features in further analysis or feature selection. The long-tail distribution of importance suggests that while a few features dominate the model's predictions, many others contribute marginally.

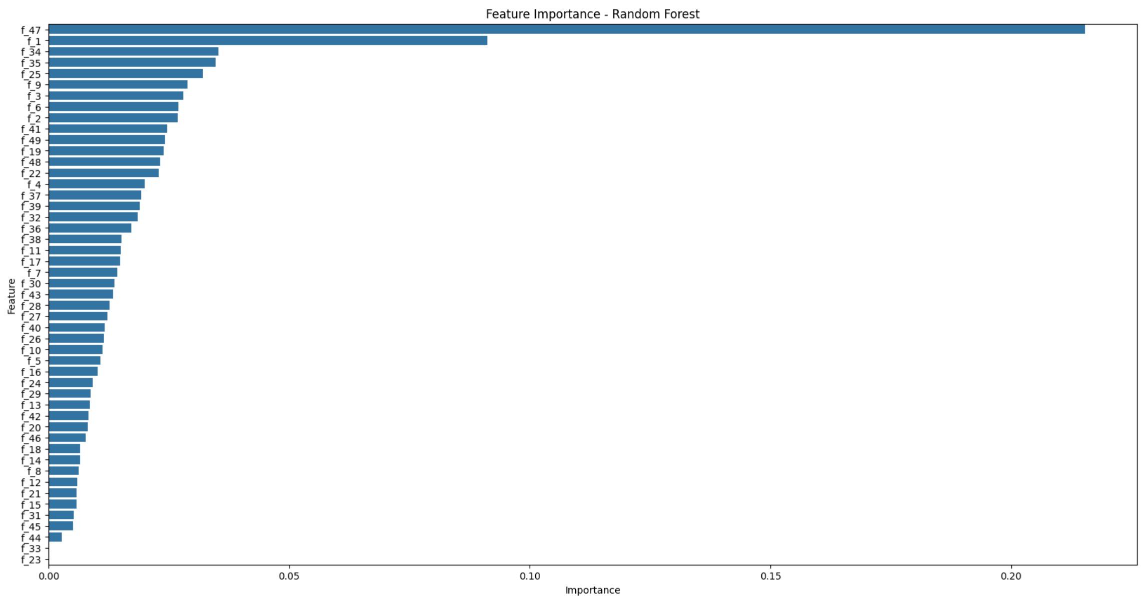


Figure 2.24: Feature Importance from Random Forest Classifier. Feature `f47` has the highest importance, significantly more than the rest.

2.6 Model Comparison using ROC Curves

The ROC curves for various classifiers are shown in Figure 2.25. The AUC values, which quantify model performance, show that the Perceptron model (AUC = 0.90), Random

Forest (AUC = 0.89), and RBF SVM (AUC = 0.88) achieve the highest scores. These models demonstrate superior capability in distinguishing between hazardous and non-hazardous asteroids. Meanwhile, the K Nearest Neighbors model (AUC = 0.73) shows relatively lower performance, highlighting the importance of algorithm selection in classification tasks.

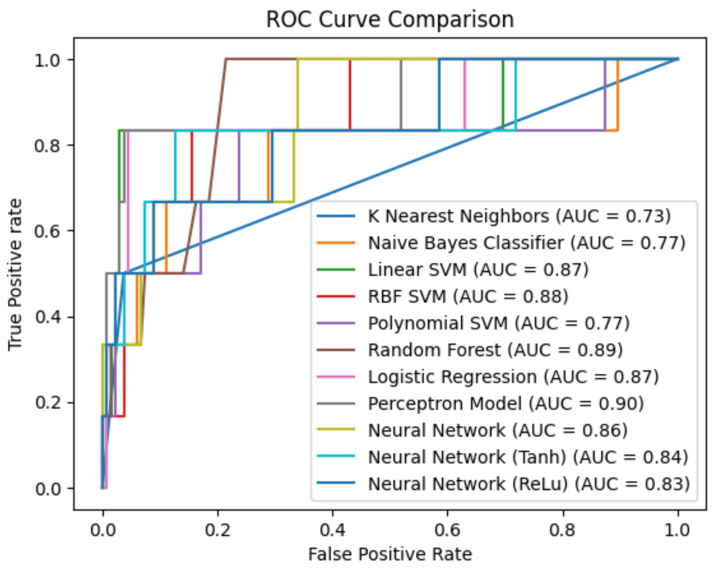
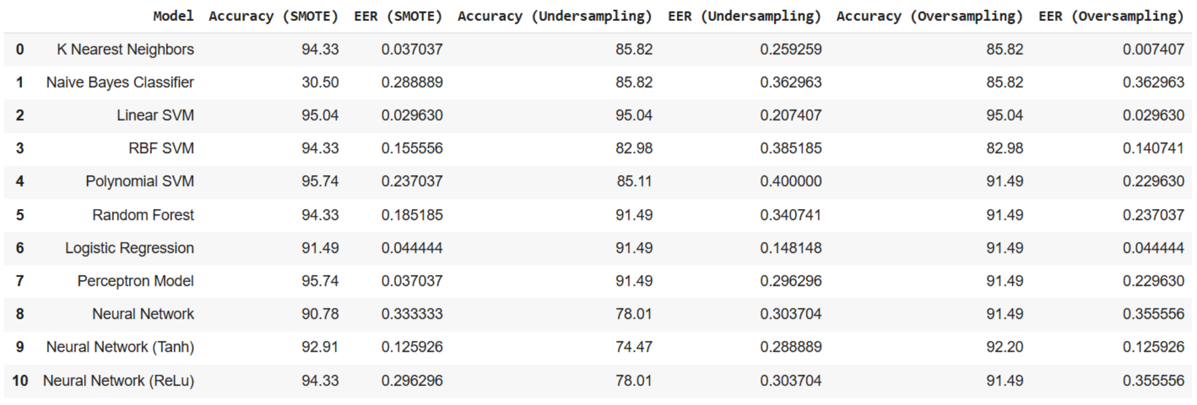


Figure 2.25: ROC curve comparison of all classifiers with AUC scores. A higher AUC indicates better model performance.

3 Inferences from Validation Data



	Model	Accuracy (SMOTE)	EER (SMOTE)	Accuracy (Undersampling)	EER (Undersampling)	Accuracy (Oversampling)	EER (Oversampling)
0	K Nearest Neighbors	94.33	0.037037	85.82	0.259259	85.82	0.007407
1	Naive Bayes Classifier	30.50	0.288889	85.82	0.362963	85.82	0.362963
2	Linear SVM	95.04	0.029630	95.04	0.207407	95.04	0.029630
3	RBF SVM	94.33	0.155556	82.98	0.385185	82.98	0.140741
4	Polynomial SVM	95.74	0.237037	85.11	0.400000	91.49	0.229630
5	Random Forest	94.33	0.185185	91.49	0.340741	91.49	0.237037
6	Logistic Regression	91.49	0.044444	91.49	0.148148	91.49	0.044444
7	Perceptron Model	95.74	0.037037	91.49	0.296296	91.49	0.229630
8	Neural Network	90.78	0.333333	78.01	0.303704	91.49	0.355556
9	Neural Network (Tanh)	92.91	0.125926	74.47	0.288889	92.20	0.125926
10	Neural Network (ReLu)	94.33	0.296296	78.01	0.303704	91.49	0.355556

Figure 3.1: Accuracy-EER Comparison across all the models

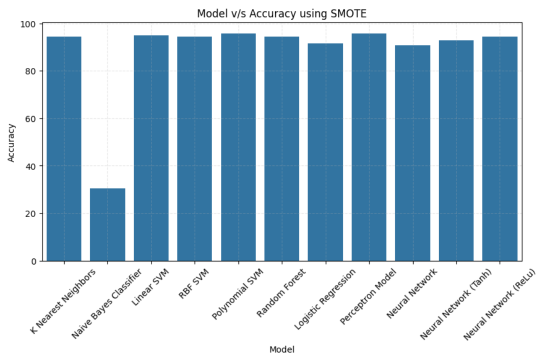


Figure 3.2: Comparison of accuracy of all the Models (SMOTE)

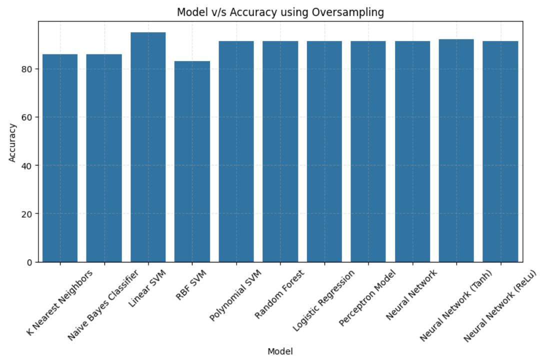


Figure 3.3: Comparison of accuracy of all the Models (Oversampling)

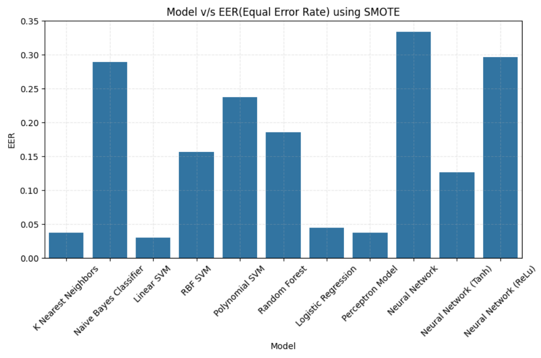


Figure 3.4: Comparison of EER of all the Models (SMOTE)

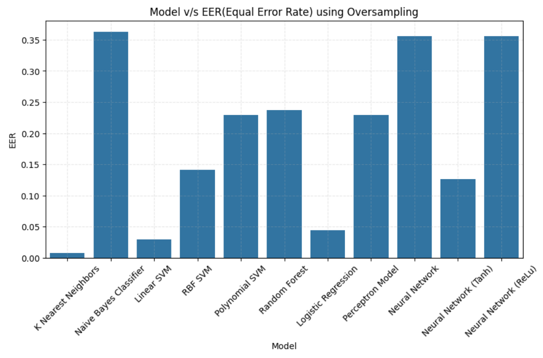


Figure 3.5: Comparison of EER of all the Models (Oversampling)

Based on the comprehensive evaluation across sampling techniques and model types, the **Perceptron model with SMOTE** stands out as the top-performing classifier, achieving the highest accuracy of **95.74%** with a low Equal Error Rate (EER) of **0.037**. This is closely followed by the **Linear SVM**, which maintains consistent performance across all sampling strategies, showing robustness with an accuracy of **95.04%** and a minimum EER of **0.029**. Additionally, while complex models like Neural Networks benefit from oversampling methods, their higher EERs suggest potential overfitting or instability. In contrast, models like **Logistic Regression** and **K-Nearest Neighbors** also offer solid performance with low EERs, making them suitable for deployment in resource-constrained environments. Therefore, considering both predictive power and generalization ability, the **Perceptron with SMOTE** and **Linear SVM** are the most recommended models for this classification task.

As the over sampler creates copies of the minority class, as a result over sampling technique in a way increases the probability for over-fitting.[6]

The under sampler removes huge amount of rows from the majority class and hence it poses serious threat for under-fitting.[6]

4 Results on Test Data

The test data on the most effective model in our study, **Perceptron**, achieved an overall accuracy of **92.91%**, indicating a high rate of correct classifications, where the classes are balanced using SMOTE. Additionally, the model attained an equal error rate (EER) of **0.1185**, reflecting a well-balanced trade-off between false positives and false negatives. These results highlight the suitability of the perceptron for the reliable and accurate detection of oil spills.

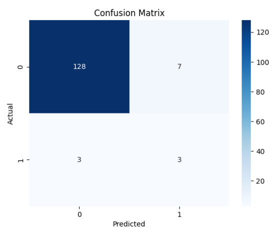


Figure 4.1: Confusion Matrix for Test Data using Perceptron

Metric	**Class 0**	**Class 1**	**Avg**
Precision	0.98	0.30	0.300
Recall	0.95	0.50	0.50
F1-Score	0.96	0.38	0.375

Figure 4.2: Class-Wise Metrics for Perceptron Model (SMOTE)

5 Conclusion

In summary, oil spills in the ocean can be extremely harmful to both onshore and offshore environments. It might take decades to restore to a normal environment, but it will never be the same again. The response aims to reduce the effect of these issues by preventing the spill from reaching land, minimizing the harm to marine life, and expediting the breakdown of the exposed oil. Wildlife recovery, cleaning, and rehabilitation are also often important parts of oil spill response. However wildlife is difficult to find and catch, oil spills can happen over wide areas, and some animals (like whales) are too big to recover. Unfortunately, it's unrealistic to rescue all wildlife impacted during oil spills. The hazardous impact of oil spills requires early detection, response, and mitigation strategy implementation as soon as possible. This report provides an overview of the classification models that are trained to detect oil spills from satellite images so that measures can be taken to reduce the damage they cause to the ocean environment.[7][8]

As the Perceptron Model exhibits consistently strong performance both in terms of high accuracy and low Equal Error Rate across the training, validation, and test data, it can be recommended as an effective and reliable approach for addressing oil spill detection classification tasks.

References

[1] https://www.unep.org/news-and-stories/story/how-manage-damage-oil-spills

[2] https://oceanservice.noaa.gov/facts/oilimpacts.html

[3] https://www.ibm.com/think/topics/moreaboutknn

[4] https://scikit-learn.org/stable/modules/naivebayes

[5] Google Colab Link for Python Implementation

[6] https://medium.com/@debspeaks/imbalanced-data-classification-oversampling-and-undersampling-297ba21fbd7c

[7] https://www.sciencedirect.com/science/article/pii/S2352485524005097

[8] https://www.noaa.gov/education/resource-collections/ocean-coasts/oil-spills