Resume Points of Our Seniors:  
1) Handled imbalanced dataset with SMOTE algorithm and reduce dimensions from 22 to 8 using PCA

2) Analysed performanced of logistic regression, decision tree, KNN, bernauli and gausian naive bayes

3) Achieved best accuracy of 96.6, ROC-AUC score of 0.98 for and F1 score of 0.97 for KNN:

Our Résume Points:

**Comparative Analysis of ML Techniques for Parkinson’s Disease Prediction**

◦ Handled **Imbalanced** Data with **SMOTE** algorithm and reduced dimension from 753 to 116 features using **PCA**.

◦ Analyzed performance of **XGboost**, **SVM**, **Random Forest**, **KNN**, **Logistic,** and **Naive Bayes** Algorithms.

◦ Achieved Highest **ROC-AUC** scores **0.96** (Random Forest) and 0.94 (PCA, XGBoost) for Balanced Datase

**Latex Code, Direct copy and paste for resume Points:**

**Abstract:**

Parkinson's disease is a complex and debilitating neurological disorder affecting millions of people worldwide. Early detection of the disease plays a crucial role in providing effective treatment and better management of symptoms. In this article, we explore the potential of various machine learning algorithms, including XGBoost, KNN, SVMs, and Random Forest, to detect the early onset of Parkinson's disease. We utilize a valuable dataset available from the UCL Parkinson Data-set to train and evaluate these algorithms.

Motivation:

The impact of Parkinson's disease on a person's quality of life can be profound, making early detection a matter of utmost importance. Timely diagnosis allows for prompt medical intervention, enabling healthcare professionals to implement appropriate treatment strategies and improve the overall prognosis for patients. With the rising prevalence of Parkinson's disease on a global scale, there is an increasing need for accurate and efficient methods to detect its early onset.

Machine learning algorithms hold great promise in addressing this challenge, as they can analyze complex patterns in the data and potentially identify subtle indicators of the disease that might not be apparent through traditional diagnostic methods. Leveraging the power of machine learning can revolutionize the early detection of Parkinson's disease and lead to improved patient outcomes.

**Introduction:**

Parkinson's disease is characterized by a range of motor symptoms, such as tremors, stiffness, and difficulties in body movements. Detecting the disease at an early stage is critical, as it allows for timely medical intervention and significantly enhances the chances of effectively managing the disease progression. However, many existing diagnostic methods can only identify Parkinson's disease at an advanced stage, by which time there is already a substantial loss of dopamine in the basal ganglia, a region responsible for controlling body movements.

link to know about Parkinson Disease:

https://www.analyticsvidhya.com/blog/2021/07/parkinson-disease-onset-detection-using-machine-learning/

**DATASET:**

Dataset 1:

download link of dataset:

https://archive.ics.uci.edu/dataset/174/parkinsons

This dataset is composed of a range of biomedical voice measurements from 31 people, 23 with Parkinson's disease (PD). Each column in the table is a particular voice measure, and each row corresponds one of 195 voice recording from these individuals ("name" column). The main aim of the data is to discriminate healthy people from those with PD, according to "status" column which is set to 0 for healthy and 1 for PD.

The data is in ASCII CSV format. The rows of the CSV file contain an instance corresponding to one voice recording. There are around six recordings per patient, the name of the patient is identified in the first column.

Additional Information

Matrix column entries (attributes):

name - ASCII subject name and recording number

MDVP:Fo(Hz) - Average vocal fundamental frequency

MDVP:Fhi(Hz) - Maximum vocal fundamental frequency

MDVP:Flo(Hz) - Minimum vocal fundamental frequency

MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP - Several measures of variation in fundamental frequency

MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA - Several measures of variation in amplitude

NHR,HNR - Two measures of ratio of noise to tonal components in the voice

status - Health status of the subject (one) - Parkinson's, (zero) - healthy

RPDE,D2 - Two nonlinear dynamical complexity measures

DFA - Signal fractal scaling exponent

spread1,spread2,PPE - Three nonlinear measures of fundamental frequency variation

Dataset 2:

3 repetitions for each patient.

PD : 188 patients

Healthy : 64 patients

Total : 252 patients

no. of rows : 756

download link of dataset: https://www.kaggle.com/datasets/dipayanbiswas/parkinsons-disease-speech-signal-features?resource=download

Data Set Information:

The data used in this study were gathered from 188 patients with PD (107 men and 81 women) with ages ranging from 33 to 87 (65.1Â±10.9) at the Department of Neurology in CerrahpaÅŸa Faculty of Medicine, Istanbul University. The control group consists of 64 healthy individuals (23 men and 41 women) with ages varying between 41 and 82 (61.1Â±8.9). During the data collection process, the microphone is set to 44.1 KHz and following the physicianâ€™s examination, the sustained phonation of the vowel /a/ was collected from each subject with 3 repetitions.

Attribute Information:

Various speech signal processing algorithms including Time Frequency Features, Mel Frequency Cepstral Coefficients (MFCCs), Wavelet Transform based Features, Vocal Fold Features and TWQT features have been applied to the speech recordings of Parkinson's Disease (PD) patients to extract clinically useful information for PD assessment.

Preprocessing:

**Random sampling** : data is split using simple random sampling, which means that the samples are randomly shuffled and divided into training and testing sets. This randomness may lead to different distributions of classes in the training and testing sets, especially if the dataset is imbalanced. In some cases, the test set might have an unrepresentative proportion of one class, leading to potentially higher accuracy.

**Stratified sampling:**

split using stratified sampling, which ensures that the proportion of classes in the training and testing sets remains similar to the original dataset. This is particularly important when dealing with imbalanced datasets, as it helps the model to be trained on a more balanced representation of classes. However, in cases where the data is not highly imbalanced, stratified sampling might lead to slightly smaller variations in class distribution between the training and testing sets compared to random sampling. This could result in a lower accuracy in some situations.

stratified sampling are expected due to the nature of the stratified split. The stratified sampling aims to maintain the class distribution in both the training and testing sets, which can lead to unequal class sizes in the test set, especially when the original dataset has an imbalanced class distribution.

In the context of Parkinson's disease prediction, when dealing with imbalanced data, stratified sampling is generally preferred over random sampling. Here's why:

Stratified Sampling:

Balances Class Distribution: Stratified sampling ensures that the class distribution in both the training and testing sets is representative of the original dataset. It maintains the proportions of different classes, making it suitable for imbalanced datasets like Parkinson's disease prediction, where one class (e.g., Parkinson's positive) might be significantly smaller than the other (e.g., healthy).

Better Generalization: By preserving the class distribution, stratified sampling helps the model to be exposed to a more balanced representation of classes during training. This allows the model to learn patterns from both the majority and minority classes, leading to better generalization and improved performance on unseen data.

More Reliable Evaluation: When evaluating the model's performance on the test set, stratified sampling ensures that the results are representative of the real-world scenario, where class imbalance might be present. This provides a more reliable estimate of the model's performance.

Random Sampling:

Suitable for Balanced Data: Random sampling is generally suitable when the dataset is balanced, meaning there is an equal representation of classes. In such cases, random sampling ensures that each data point has an equal chance of being included in the training or testing set.

Speed and Simplicity: Random sampling can be computationally faster and simpler to implement, especially for large datasets. When class distribution is not imbalanced, random sampling can still provide reasonable results.

In summary, for Parkinson's disease prediction with an imbalanced dataset, stratified sampling is the better choice. It helps to ensure that the model is trained and evaluated in a way that reflects the real-world class distribution, which can lead to improved performance and better generalization on unseen data.

Q. What if we have apply SMOTE than which one is best to use ?

If you have applied SMOTE to handle the class imbalance, then the choice between stratified sampling and random sampling becomes less critical. SMOTE has already addressed the class imbalance issue by oversampling the minority class, creating synthetic samples to balance the dataset.

In this scenario, both stratified sampling and random sampling can be suitable choices, but stratified sampling remains slightly more preferable. Here's why:

With SMOTE:

Stratified Sampling: Stratified sampling can still be beneficial, even after applying SMOTE. It ensures that the training and testing sets have similar class distributions, which can help to evaluate the model's performance in a way that reflects real-world conditions.

Random Sampling: If you have already applied SMOTE, the class imbalance has been addressed, making random sampling a reasonable option as well. Since the class distribution is now balanced, random sampling can still provide reliable evaluation results.

In practice, if you have already applied SMOTE, stratified sampling might offer a slight advantage in terms of a more balanced and representative evaluation of the model. However, both stratified sampling and random sampling can be suitable options with SMOTE in place, and the difference in performance between the two sampling methods is likely to be less pronounced compared to the case without SMOTE. Ultimately, the choice between stratified and random sampling might not have a significant impact on model performance if SMOTE has already balanced the classes.

**Preprocessing 👍**

Data Loading: Same as before, the first step is to load the dataset using pandas read\_csv() function.

Feature Selection: As mentioned earlier, feature selection is the process of selecting a subset of relevant features from the original set of features. In the code, we have assumed that feature selection has been performed before using XGBoost, which is a powerful ensemble learning algorithm capable of feature importance ranking. XGBoost can assign importance scores to each feature, and we can then select the most relevant ones based on these scores.

Feature Scaling: The features are scaled using MinMaxScaler from scikit-learn to ensure they are in a similar range. XGBoost is not as sensitive to feature scaling as some other algorithms, but it can still benefit from scaling when features have very different ranges.

Data Balancing with SMOTE: Similar to before, SMOTE is used to balance the data by generating synthetic samples for the minority class. While XGBoost is known to handle imbalanced datasets well, using SMOTE can further improve its ability to predict the minority class and avoid overfitting.

Stratified Sampling: We still perform stratified sampling when splitting the data into training and testing sets to maintain the class distribution in both sets.

Model Training and Evaluation: In the code, XGBoost (eXtreme Gradient Boosting) is used as one of the models. XGBoost is an ensemble learning method that uses decision trees as base learners and combines their predictions to produce a more robust and accurate model. It is a popular choice due to its high predictive performance and efficient computation.

Overall, the inclusion of XGBoost in the preprocessing steps adds another powerful machine learning algorithm to the mix. Its ability to rank feature importance and handle imbalanced datasets makes it a valuable addition to the model selection process.

Q. Why XGboost is giving good accuracy, F1 score, TPR and FPR in all those models ?

XGBoost is known for its superior performance in various machine learning tasks, and it often outperforms other algorithms in terms of accuracy, F1 score, and other evaluation metrics. There are several reasons why XGBoost may be giving higher metrics compared to other models:

Gradient Boosting Technique: XGBoost is based on the gradient boosting technique, which builds multiple weak learners (typically decision trees) sequentially. Each tree is trained to correct the errors of its predecessor. This iterative approach allows XGBoost to focus on misclassified samples and progressively improve the model's performance.

Regularization: XGBoost employs L1 and L2 regularization terms in its objective function to prevent overfitting. Regularization helps control the complexity of the model and reduces the likelihood of fitting noise in the data, leading to better generalization.

Feature Importance: XGBoost provides a feature importance ranking, which allows us to identify and use the most relevant features in the dataset. By focusing on the most informative features, the model can make better decisions and achieve higher accuracy.

Handling Imbalanced Data: In the code, we use SMOTE to balance the data before training the models. While XGBoost is already robust to imbalanced data, the additional balanced dataset from SMOTE further enhances its ability to capture patterns in the minority class, resulting in improved F1 score and TPR.

Fine-tuning and Hyperparameter Optimization: XGBoost provides a wide range of hyperparameters that can be tuned to optimize model performance. By carefully tuning these hyperparameters, we can achieve better accuracy and F1 score compared to default settings.

Parallelization and Optimization: XGBoost is designed for efficiency and speed. It supports parallel processing and various optimization techniques, making it faster and more scalable than other algorithms, especially when dealing with large datasets.

Robustness to Outliers and Missing Values: XGBoost is less sensitive to outliers and missing values compared to some other algorithms. It can handle these data issues without requiring extensive data preprocessing.

While XGBoost tends to perform well across various datasets, it's essential to remember that the choice of algorithm depends on the specific characteristics of the dataset and the problem at hand. Different algorithms may perform better in certain scenarios, and it's always a good practice to try multiple models and compare their results before finalizing the choice. Additionally, hyperparameter tuning and feature engineering can significantly impact the performance of any model, including XGBoost.

**Models:**

Model 1, Without SMOTE, Imbalance dataset, Min-max Scaling, Random Sampling

Model 2, SMOTE, Balanced Dataset, Min-MAX Scaling, Random Sampling

Model 3, SMOTE, Balanced Dataset, Min-Max Scaling, Stratified Sampling

**Algorithms:**  
  
Support Vector Machine (SVM):

SVM is a powerful and versatile supervised machine learning algorithm used for both classification and regression tasks.

In classification tasks like this one, SVM aims to find the optimal hyperplane that best separates the data into different classes. It tries to maximize the margin between classes, which leads to better generalization.

SVM can handle non-linear data by using kernel functions to transform the input space into a higher-dimensional space where data points become more separable.

In this code, we use the default radial basis function (RBF) kernel, which is often suitable for various problems.

k-Nearest Neighbors (KNN):

KNN is a simple and intuitive classification algorithm that works based on the idea of similarity.

In KNN, to classify a data point, we find the k-nearest neighbors (data points with the closest feature values) from the training data and take a majority vote among those neighbors' classes to assign the label to the new point.

KNN does not involve training a model. Instead, it stores the entire training dataset, making it a lazy learner.

It is essential to choose an appropriate value for 'k' to avoid overfitting or underfitting.

Random Forest:

Random Forest is an ensemble learning method based on decision trees.

It builds multiple decision trees during training and combines their predictions through voting (classification) or averaging (regression) to make the final prediction.

Each decision tree is trained on a random subset of the data, and only a subset of features is considered for each split. This randomness and combination of trees help to reduce overfitting and improve performance.

Random Forest is known for its robustness, scalability, and high accuracy.

Logistic Regression:

Despite its name, logistic regression is a binary classification algorithm, commonly used for problems with two classes (binary classification).

It estimates the probability that a data point belongs to a particular class using the logistic function (sigmoid function).

It models the relationship between the features and the target class, with the aim of predicting the probability of the data point belonging to the positive class (Parkinson's disease in this case).

Gaussian Naive Bayes:

Naive Bayes is a probabilistic classifier based on Bayes' theorem with the assumption of independence between features (hence "naive").

Gaussian Naive Bayes is the variant used when the features are continuous and assumed to have a Gaussian (normal) distribution.

It calculates the conditional probability of a data point belonging to a class based on the likelihood of each feature given the class and the prior probability of the class.

XGBoost (Extreme Gradient Boosting):

XGBoost is an advanced gradient boosting algorithm designed for speed and performance.

Gradient boosting is an ensemble technique that builds multiple weak learners (typically decision trees) sequentially, with each tree attempting to correct the errors of its predecessor.

XGBoost introduces regularization, tree pruning, and handling missing values, making it more accurate and less prone to overfitting.

It is widely used in various machine learning competitions and has become a popular choice in industry applications due to its robustness and high performance.

**Evaluation Metrices:**

Accuracy: Accuracy measures the proportion of correct predictions out of the total predictions. It is commonly used as a general performance metric when the class distribution is balanced.

ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) is an evaluation metric commonly used to assess the performance of classification models, particularly in binary classification problems. It measures the ability of a model to distinguish between positive and negative classes across different threshold values. Here's an overview of ROC-AUC and its interpretation:

ROC Curve: The ROC curve is a graphical representation of the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The ROC curve helps visualize the trade-off between sensitivity (TPR) and specificity (1 - FPR) for different thresholds.

AUC (Area Under the Curve): The AUC is the area under the ROC curve. It summarizes the overall performance of a model across all possible threshold values. AUC ranges from 0 to 1, where a higher value indicates better model performance.

Interpretation of ROC-AUC:

AUC = 0.5: The model's performance is no better than random guessing. It's like flipping a coin.

0.5 < AUC < 1: The model is able to distinguish between the positive and negative classes to some extent. The higher the AUC, the better the model's performance. An AUC of 1 indicates a perfect model.

AUC = 1: The model perfectly separates the positive and negative classes for all threshold values.

Advantages of ROC-AUC:

Threshold-Invariant: ROC-AUC is threshold-invariant, meaning it evaluates the model's performance across different threshold levels. It focuses on the model's ability to rank examples correctly, regardless of the specific threshold chosen.

Imbalanced Classes: ROC-AUC is useful for imbalanced datasets, where one class is much more prevalent than the other. It provides a comprehensive view of the model's performance beyond simple accuracy.

Model Comparison: ROC-AUC is especially useful for comparing different models. The model with the higher AUC is generally considered better at distinguishing between classes.

Limitations of ROC-AUC:

Doesn't Consider Costs: ROC-AUC doesn't consider the real-world costs associated with false positives and false negatives. Depending on the specific problem, you might need to optimize for a different metric.

Imbalanced Datasets: While ROC-AUC is useful for imbalanced datasets, it doesn't provide a complete picture of the model's performance. Precision-Recall curves and metrics might be more informative in such cases.

In summary, ROC-AUC is a valuable evaluation metric for binary classification problems, providing insight into the overall discriminative power of the model. However, it's important to consider other metrics and the specific context of your problem when evaluating and interpreting model performance.

F1 Score: F1 score is the harmonic mean of precision and recall. It is a suitable metric when dealing with imbalanced datasets as it considers both false positives and false negatives. F1 score is more informative than accuracy when classes are imbalanced.

True Positive Rate (TPR): TPR, also known as sensitivity or recall, measures the proportion of positive samples that are correctly identified by the model. It is important to consider TPR, especially when the cost of false negatives is high, such as in medical diagnoses.

False Positive Rate (FPR): FPR measures the proportion of negative samples that are incorrectly classified as positive by the model. It is essential in applications where the cost of false positives is significant, like fraud detection.

*Now, let's go step by step through the code to understand which evaluation metrics are used for each model and why:*

Data Preprocessing: We first load the dataset and extract the features and labels. The dataset contains speech features, and the labels represent whether a subject has Parkinson's disease or not.

Scaling Features: We use MinMaxScaler to scale the features to a similar range. Scaling is important to ensure that features with larger numerical values do not dominate the learning process.

SMOTE (Optional): If SMOTE is applied, we use it to balance the data by oversampling the minority class (Parkinson's disease). SMOTE generates synthetic samples to match the number of samples in the majority class (Healthy).

Train-Test Split: We split the dataset into training and testing sets. Stratified sampling is used to preserve the class distribution in both sets, especially when the data is imbalanced.

Model Training and Evaluation: We train and evaluate multiple models using the training and testing data.

a. For SVM, KNN, Random Forest, and XGBoost:

We train each model with the training data.

We make predictions on the test data using each model.

We calculate accuracy and F1 score for each model using the predicted labels and true labels.

We create a confusion matrix to calculate TPR and FPR for each model.

b. For Logistic Regression:

We set the maximum number of iterations to avoid convergence warnings.

We train the logistic regression model with the training data.

We make predictions on the test data using the logistic regression model.

We calculate accuracy and F1 score for the logistic regression model using the predicted labels and true labels.

We create a confusion matrix to calculate TPR and FPR for the logistic regression model.

c. For Naive Bayes:

We train the Gaussian Naive Bayes model with the training data.

We make predictions on the test data using the Naive Bayes model.

We calculate accuracy and F1 score for the Naive Bayes model using the predicted labels and true labels.

We create a confusion matrix to calculate TPR and FPR for the Naive Bayes model.

Results: We display the evaluation metrics (accuracy, F1 score, TPR, FPR) for each model to compare their performance on the test data.

**Results:**

**Model 1 | Accuracy | F1 Score | ROC-AUC | True Positive Rate | False Positive Rate |**

**---------------------|----------|----------|----------|--------------------|--------------------|**

**XGBoost | 88.16 | 0.92 | 0.92 | 0.96 | 0.37 |**

**SVM | 0.83 | 0.89 | 0.69 | 0.99 | 0.62 |**

**KNN | 0.86 | 0.91 | 0.79 | 0.94 | 0.36 |**

**Random Forest | 0.85 | 0.91 | 0.74 | 0.98 | 0.49 |**

**Logistic Regression | 0.87 | 0.92 | 0.77 | 0.98 | 0.43 |**

**Naive Bayes | 0.76 | 0.83 | 0.73 | 0.80 | 0.34 |**

**-Model2--------------------|----------|----------|----------|--------------------|--------------------|**

**XGBoost | 95.00 | 0.95 | 0.95 | 0.94 | 0.04 |**

**SVM | 0.82 | 0.81 | 0.82 | 0.78 | 0.14 |**

**KNN | 0.89 | 0.88 | 0.89 | 0.79 | 0.00 |**

**Random Forest | 0.96 | 0.96 | 0.96 | 0.93 | 0.02 |**

**Logistic Regression | 0.88 | 0.88 | 0.88 | 0.86 | 0.12 |**

**Naive Bayes | 0.79 | 0.79 | 0.79 | 0.69 | 0.18 |**

**---------------------|----------|----------|----------|--------------------|--------------------|**

**XGBoost | 95.00 | 0.95 | 0.95 | 0.95 | 0.05 |**

**SVM | 0.81 | 0.77 | 0.81 | 0.64 | 0.02 |**

**KNN | 0.81 | 0.81 | 0.81 | 0.78 | 0.14 |**

**Random Forest | 0.93 | 0.93 | 0.93 | 0.93 | 0.06 |**

**Logistic Regression | 0.87 | 0.87 | 0.87 | 0.83 | 0.12 |**

**Naive Bayes | 0.75 | 0.73 | 0.75 | 0.69 | 0.18 |**

**+Model4----------------------------------+---------+----------+--------------+----------+-------+--------+-------+---------+**

**| Model | Accuracy| F1 Score | ROC-AUC Score| TPR | FPR |Removed |Features| Total |**

**| | | | | (Sensitivity)| (1-Specificity)|Features | After |**

**| | | | | | | | After | Features|**

**+-Model3---------------------------------+---------+----------+--------------+----------+-------+--------+---------+---------+**

**| SVM | 0.95 | 0.95 | 0.95 | 0.92 | 0.02 | 637 | 116 | 753 |**

**| KNN | 0.89 | 0.88 | 0.90 | 0.79 | 0.00 | 637 | 116 | 753 |**

**| Random Forest | 0.92 | 0.92 | 0.92 | 0.92 | 0.07 | 637 | 116 | 753 |**

**| Logistic Regression | 0.86 | 0.86 | 0.87 | 0.80 | 0.07 | 637 | 116 | 753 |**

**| Naive Bayes | 0.79 | 0.78 | 0.79 | 0.72 | 0.14 | 637 | 116 | 753 |**

**| XGBoost | 0.94 | 0.94 | 0.94 | 0.91 | 0.02 | 637 | 116 | 753 |**

**+------------------------------------+---------+----------+--------------+----------+-------+--------+---------+---------+**

**Questions :**

Question 1:

a. Why is scaling the features using MinMaxScaler important in machine learning models?

Answer 1a:

Scaling the features using MinMaxScaler is important in machine learning models because it brings all features to a similar range, typically between 0 and 1. This is crucial for models like SVM, KNN, and Neural Networks that are sensitive to the scale of input features. Scaling ensures that no feature dominates the others, leading to more stable and efficient training.

Question 2:

b. Explain the purpose of using SMOTE for handling imbalanced datasets. How does it impact the performance of models?

Answer 2b:

The purpose of using SMOTE (Synthetic Minority Over-sampling Technique) for handling imbalanced datasets is to create synthetic samples of the minority class by interpolating between existing samples. This helps balance the class distribution, allowing the model to learn from both classes more effectively and improve its performance on the minority class. SMOTE impacts the performance of models by improving their true positive rate while reducing the false positive rate, leading to better overall accuracy and F1 score.

Question 3:

c. What is stratified sampling, and why is it preferred when splitting the dataset for training and testing?

Answer 3c:

Stratified sampling is a sampling technique where the data is divided into subsets in such a way that the class distribution in each subset closely matches the original dataset. It is preferred when splitting the dataset for training and testing because it ensures that both the training and testing sets have a representative distribution of classes, reducing the risk of biased evaluation. In the context of imbalanced datasets, stratified sampling ensures that both the minority and majority classes are well-represented in both the training and testing sets, enabling a more fair evaluation of model performance.

Question 4:

a. Describe the concept of the confusion matrix. How is it useful in evaluating the performance of a classifier?

Answer 4a:

The confusion matrix is a table that shows the true positives, true negatives, false positives, and false negatives of a classifier's predictions. It helps in evaluating the performance of a classifier by providing a more detailed view of its predictive abilities beyond simple accuracy. True positives (TP) represent the correctly predicted positive samples, true negatives (TN) represent the correctly predicted negative samples, false positives (FP) represent the incorrectly predicted positive samples, and false negatives (FN) represent the incorrectly predicted negative samples.

Question 5:

b. What is the significance of accuracy, F1 score, true positive rate (TPR), and false positive rate (FPR) in model evaluation? How do they differ, and when would you use each of them?

Answer 5b:

Accuracy represents the proportion of correct predictions out of the total predictions. F1 score is the harmonic mean of precision and recall and is useful when the classes are imbalanced. True positive rate (TPR) is the proportion of true positive predictions out of the total actual positive samples, and it measures the model's ability to correctly identify positive samples. False positive rate (FPR) is the proportion of false positive predictions out of the total actual negative samples, and it measures the model's ability to incorrectly classify negative samples as positive.

Accuracy is useful when the classes are balanced, and all classes have similar importance. F1 score is suitable when there is an imbalance between classes and when false positives and false negatives have different consequences. TPR and FPR are particularly relevant in cases where one class is more critical to be correctly classified, such as in medical diagnosis where correctly identifying positive cases (TPR) is crucial, and incorrectly classifying healthy individuals as positive (FPR) can have serious consequences.

Question 6:

c. Among the models SVM, KNN, Random Forest, Logistic Regression, and Naive Bayes, which one performed the best on the given dataset? Justify your answer based on the provided evaluation metrics.

Answer 6c:

Among the models SVM, KNN, Random Forest, Logistic Regression, and Naive Bayes, the best-performing model on the given dataset is XGBoost. It achieved the highest accuracy, F1 score, TPR, and the lowest FPR compared to other models, indicating its superiority in handling the dataset.

Question 7:

a. What is XGBoost, and how does it differ from traditional gradient boosting techniques?

Answer 7a:

XGBoost (Extreme Gradient Boosting) is an advanced implementation of gradient boosting algorithms. It is a popular machine learning algorithm known for its efficiency, speed, and high performance. XGBoost differs from traditional gradient boosting techniques in several ways:

Regularization: XGBoost includes regularization techniques such as L1 (Lasso) and L2 (Ridge) regularization, which help prevent overfitting and improve model generalization.

Split Finding: XGBoost uses a "greedy algorithm" to find the best split points in the trees, resulting in faster and more efficient training.

Tree Pruning: XGBoost employs a depth-first strategy for tree building, allowing it to prune trees during the building process, which further improves model efficiency and prevents overfitting.

Handling Missing Values: XGBoost has built-in capabilities to handle missing data during training, eliminating the need for imputation techniques.

Weighted Quantile Sketch: XGBoost uses weighted quantile sketch to efficiently find the optimal split points for continuous features, which speeds up the process of building decision trees.

Question 8:

b. Explain the advantages of using ensemble learning, and why did XGBoost outperform other models in the given dataset?

Answer 8b:

Ensemble learning combines the predictions of multiple individual models (base learners) to produce a final prediction. The advantages of ensemble learning include:

Improved Performance: Ensemble models often outperform individual models, as they can reduce bias and variance, leading to better generalization.

Robustness: Ensemble models are less prone to overfitting and are more robust to noisy data.

Model Diversity: By using different algorithms or subsets of data, ensemble methods create diverse models, which can capture different patterns in the data.

XGBoost outperformed other models in the given dataset due to several reasons:

Handling Imbalanced Data: XGBoost can handle imbalanced datasets effectively, as demonstrated by its high true positive rate and low false positive rate. This is crucial in the context of medical diagnosis, where correctly identifying positive cases (patients with Parkinson's disease) is critical.

Regularization: XGBoost's regularization techniques help prevent overfitting, ensuring better generalization to unseen data.

Optimization: XGBoost's optimization algorithm efficiently finds the best split points in the trees, leading to faster training and better performance.

Ensemble Nature: As an ensemble method itself, XGBoost leverages the strengths of multiple decision trees to make more accurate predictions.

Question 9:

a. Compare and contrast the results of the three models (XGBoost, SVM, and KNN) on the same dataset. Which model performed better, and why?

Answer 9a:

Comparing the results of XGBoost, SVM, and KNN on the same dataset:

XGBoost achieved the highest accuracy, F1 score, true positive rate, and the lowest false positive rate among the three models. It outperformed the others due to its ability to handle imbalanced data, regularization, and optimization techniques.

SVM achieved moderate accuracy and F1 score, but it had a high false positive rate compared to XGBoost. SVM might have struggled with the imbalanced dataset, resulting in a lower true positive rate.

KNN achieved reasonable accuracy and F1 score, but its true positive rate was lower than that of XGBoost. KNN might not have been able to capture complex patterns in the dataset as effectively as XGBoost's ensemble of decision trees.

Based on the provided evaluation metrics, XGBoost performed better on the given dataset due to its ensemble nature, regularization, and efficient optimization.

Question 10:

b. Discuss the impact of using SMOTE on model performance. How did it affect the true positive rate, false positive rate, and overall accuracy of the models?

Answer 10b:

Using SMOTE on an imbalanced dataset has a significant impact on model performance:

True Positive Rate (TPR): SMOTE increases the true positive rate by generating synthetic samples of the minority class. This means the model becomes better at correctly identifying positive cases, which is crucial in scenarios like medical diagnosis.

False Positive Rate (FPR): SMOTE can also reduce the false positive rate by improving the model's ability to correctly classify negative cases.

Overall Accuracy: In most cases, SMOTE leads to an improvement in overall accuracy as the model becomes more balanced and learns from both classes more effectively.

However, it is essential to note that the exact impact of SMOTE can vary depending on the dataset and the model being used. While it generally helps improve model performance, there may be cases where SMOTE could lead to overfitting or other undesirable effects, so it should be used judiciously.

Question 11:

a. For the model with the highest accuracy, identify the features that significantly contributed to the classification decision. How would you interpret the importance of these features?

Answer 11a:

To identify the features that significantly contributed to the classification decision for the model with the highest accuracy (XGBoost), we can use XGBoost's built-in feature importance attribute. This attribute provides a score for each feature based on how much it contributed to reducing the impurity in the trees during the training process.

A higher feature importance score indicates a more critical role in the classification decision. By analyzing these scores, we can identify the top features that influenced the model's predictions the most. For example, if features related to vocal characteristics had high importance scores, it would indicate that these vocal features were crucial in distinguishing between Parkinson's disease and healthy individuals.

Interpreting the importance of these features can help us gain insights into the characteristics that have the most significant impact on the target variable, potentially leading to a better understanding of the disease or the problem at hand.

Question 12:

b. Suppose you need to deploy one of these models in a real-world application. Which model would you choose, and what factors would influence your decision?

Answer 12b:

The choice of the model for deployment depends on several factors:

Performance Metrics: The model with the highest accuracy, F1 score, and true positive rate is usually preferred as it indicates better overall performance and the ability to correctly identify positive cases.

Model Complexity: A simpler model like Logistic Regression or Naive Bayes might be preferred if interpretability and explainability are essential in the real-world application.

Computational Resources: The computational cost of training and deploying the model should be considered, especially if the application requires real-time predictions.

Robustness: The model should be robust to variations in the input data and generalizable to unseen samples.

Model Update: If the real-world data distribution changes over time, the ability to update and retrain the model is crucial.

Based on these factors, XGBoost could be a strong candidate for deployment due to its overall high performance, the ability to handle imbalanced data, and robustness to variations. However, the final decision would depend on the specific requirements and constraints of the real-world application.

Question 13:

a. How do you prevent overfitting in machine learning models? Do you think any of the models in this analysis may have been affected by overfitting? Why or why not?

Answer 13a:

To prevent overfitting in machine learning models, several techniques can beused:

Cross-Validation: Performing k-fold cross-validation helps to evaluate the model's performance on multiple subsets of the data, which reduces the risk of overfitting by providing a more reliable estimate of generalization error.

Regularization: Applying regularization techniques, such as L1 (Lasso) and L2 (Ridge) regularization, adds penalty terms to the model's cost function, discouraging overly complex models and preventing overfitting.

Feature Selection: Selecting relevant and informative features can prevent the model from learning noise in the data, thus reducing overfitting.

Early Stopping: Monitoring the model's performance on a validation set during the training process and stopping when the performance starts to degrade can prevent overfitting.

Ensemble Methods: Ensemble models, like Random Forest and XGBoost, tend to be less prone to overfitting compared to individual models, as they combine multiple weak learners to make predictions.

Regarding the models in this analysis, overfitting is a possibility, especially if the model is too complex relative to the size of the dataset or if no regularization techniques are used. It is essential to pay attention to the difference between the training and testing performance. If the training performance is significantly higher than the testing performance, it could indicate overfitting.

Question 14:

a. How do you think the model's performance would change if the dataset size was significantly larger? Would the results still hold?

Answer 14a:

If the dataset size was significantly larger, the model's performance might improve or stabilize. With more data, the model can capture more diverse patterns, leading to better generalization and less risk of overfitting. As the dataset grows, the model can learn from more examples and become more representative of the true data distribution.

However, the results might not be exactly the same as the dataset size increases. Performance improvements might start to plateau after a certain point, and it might become more computationally expensive to train and evaluate the model with a larger dataset.

Additionally, the impact of class imbalance might change with a larger dataset. SMOTE and other techniques used to handle imbalanced data might have varying effects when applied to larger datasets. Therefore, it is essential to continue monitoring the model's performance and possibly reevaluate the choice of hyperparameters as the dataset size increases.

Question 15:

a. Suppose you need to deploy the best-performing model in a production environment. What factors would you consider before deploying it? How would you ensure that the model continues to perform well over time?

Answer 15a:

Before deploying the best-performing model in a production environment, several factors need to be considered:

Real-World Performance: Evaluate the model's performance in a real-world setting to ensure that it generalizes well to unseen data and produces accurate predictions.

Scalability: Verify that the model can handle the anticipated volume of data and make predictions efficiently in a production environment.

Robustness: Test the model's performance under various scenarios and edge cases to ensure it remains reliable in different situations.

Monitoring: Implement monitoring mechanisms to track the model's performance over time and detect any deviations from expected behavior.

Data Drift: Check for data drift to ensure that the model remains accurate as the data distribution evolves over time.

Retraining Strategy: Define a retraining strategy to update the model periodically as new data becomes available. This could be on a fixed schedule or triggered by significant changes in the data distribution.

Version Control: Implement version control for the deployed model to track changes and roll back to previous versions if needed.

Backup Models: Have backup models in place to switch to in case the primary model encounters issues.

Security: Implement security measures to protect the model from potential attacks or unauthorized access.

Ensuring that the model continues to perform well over time requires constant monitoring, periodic retraining, and regular evaluation of the model's performance against the established metrics.

Question 16:

a. Apart from SMOTE, what other techniques can be used to address imbalanced datasets? Can you suggest some scenarios where these techniques might be more suitable?

Answer 16a:

Apart from SMOTE, several other techniques can be used to address imbalanced datasets:

Under-Sampling: This involves randomly removing instances from the majority class to balance the class distribution. Under-sampling can be suitable when the majority class has a large number of redundant or similar instances.

Synthetic Minority Over-sampling Technique (SMOTE-NC): SMOTE-NC extends SMOTE to handle datasets with both numerical and categorical features.

Adaptive Synthetic Sampling (ADASYN): ADASYN is an extension of SMOTE that generates more synthetic samples for difficult-to-learn instances.

Random Over-Sampling: Randomly duplicating instances from the minority class to balance the class distribution.

Cost-Sensitive Learning: Modifying the algorithm's cost function to penalize misclassification of the minority class more than the majority class.