Machine Learning

Second assignment solutions

Mobin Nesari
Dr. Hadi Farahani

April 20, 2023

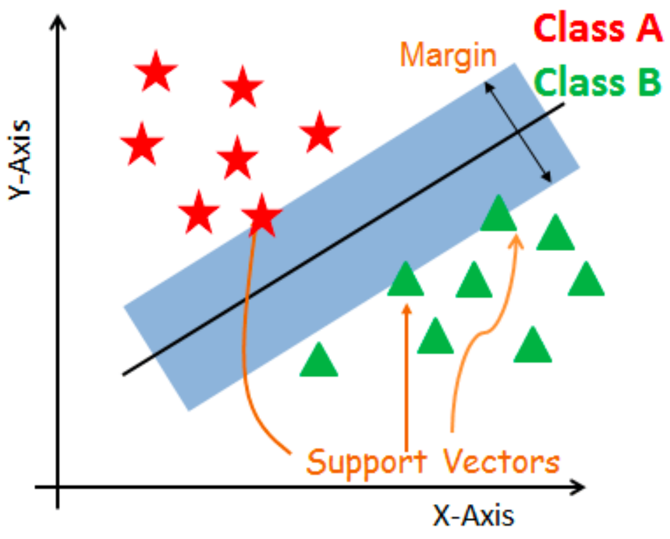
Question 1:

Is it possible for an SVM classifier to provide a confidence score or probability when making predictions on a particular instance? Explain it.

Answer:

Yes, it is possible for an SVM classifier to provide a confidence score or probability when making predictions on a particular instance.

SVM classifiers work by finding the hyperplane that best separates the different classes in the data. When making predictions on a new instance, the classifier calculates the distance of that instance from the hyperplane. This distance can be used as a measure of how confident the classifier is in its prediction.

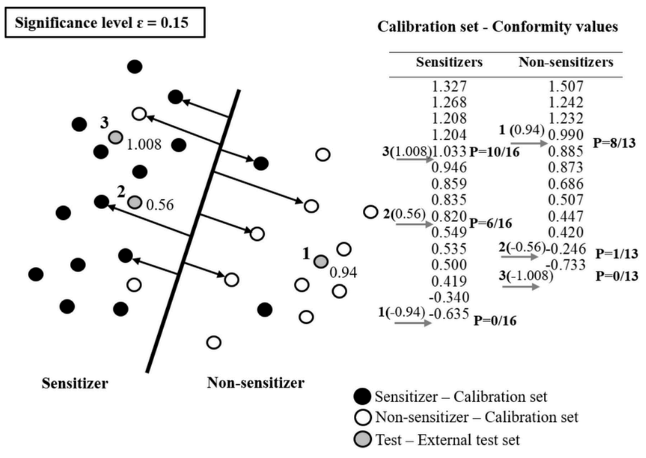


SVM Intuition

More specifically, SVM classifiers can produce a decision function that assigns a score to each input instance indicating which class it belongs to. The sign of this score determines the predicted class label, and its magnitude reflects the degree of confidence in the prediction.

In some cases, this score can be transformed into a probability estimate using a calibration function, such as Platt scaling or isotonic regression.

These methods adjust the output of the decision function to produce an estimated probability value between 0 and 1, which quantifies the level of confidence the classifier has in its prediction.



SVM with calibration set

It's important to note, however, that not all SVM implementations provide probability estimates natively, and additional steps may be needed to obtain such estimates.

Question 2:

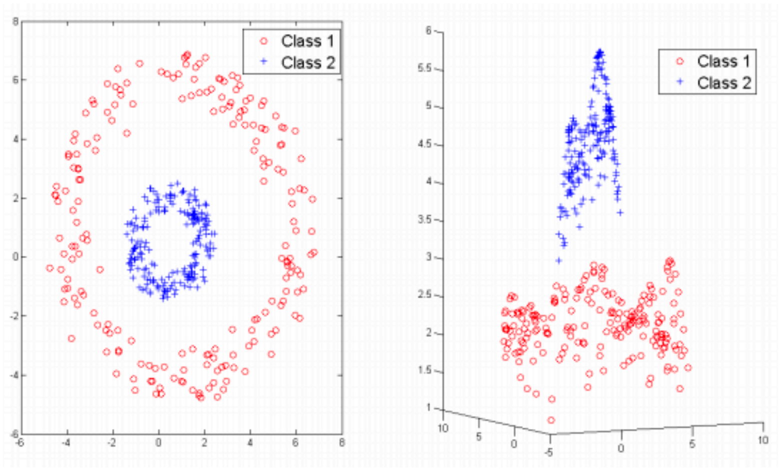
What actions should you take if your have trained an SVM classifier using an RBF kernel but notice that it underfits the training set? Would it be appropriate to increase or decrease the value of γ or C, or both?

Answer:

If an SVM classifier has been trained on a dataset using RBF kernel and it underfits the training set, increasing the value of γ, C or both should be considered.

γ is a hyperparameter that controls the shape of the decision boundary in the SVM classifier. A low γ value results in a wider decision boundary, which means that each data point has a more significant influence on the classification. Conversely, a high γ value results in a tighter decision boundary, which means that only nearby points have a significant impact on the classification.

C is another hyperparameter that controls the trade-off between achieving a low training error and a low testing error. A low C value allows for more misclassifications on the training set, resulting in a larger margin and potentially better generalization performance. Conversely, a high C value penalizes misclassifications more severely, resulting in a smaller margin and potentially worse generalization performance.



Radial Basis Function Kernel Intuition

If the SVM classifier underfits the training set, it means that the model is not complex enough to capture the underlying patterns in the data. In this case, increasing either γ or C (or both) can help increase the model's complexity and improve its ability to fit the training data. However, it's important to be careful when increasing these hyperparameters since too large values can lead to overfitting, where the model becomes too complex and starts to fit the noise in the data.

As such, you may need to experiment with different values of γ and C to find the optimal balance between underfitting and overfitting. A common approach is to use grid search or randomized search to explore a range of hyperparameter values and choose the ones that yield the best cross-validation performance.

Question 3:

What does it mean for a model to be ϵ-insensitive?

Answer:

ϵ-insensitive is a term used to describe the loss function of Support Vector Regression (SVR), which is an extension of Support Vector Machines (SVM) for regression problems. In an ϵ-insensitive SVR, a tolerance level (ϵ) is specified such that any prediction error within this tolerance level is treated as zero in the loss function, while any prediction error outside this tolerance level contributes to the loss according to its magnitude.

In other words, an ϵ-insensitive model allows some degree of error or slack in the predictions. If the actual target value and the predicted value differ by less than epsilon, it's considered a correct prediction, and no penalty is incurred. This means that the model focuses on reducing the errors that are larger than ϵ. That means an ϵ-insensitive loss function will be:

$$L_{\epsilon-ins} = \frac{1}{m}\sum_{i=1}^{m} \max\{\,|y_i - \hat{y}_i| - \epsilon, 0\}$$

The ϵ value is a hyperparameter that controls the trade-off between the magnitude of the errors and the number of support vectors used to fit the data. A smaller ϵ value results in a more strict model that tries to minimize the error as much as possible, potentially leading to overfitting. Conversely, a larger ϵ value allows for more flexibility in the model and can result in better generalization performance.

Overall, the epsilon-insensitive property of SVMs/SVRs allows for greater robustness against outliers and noise in the data, and gives more control over the trade-off between accuracy and simplicity of the model.

Question 4:

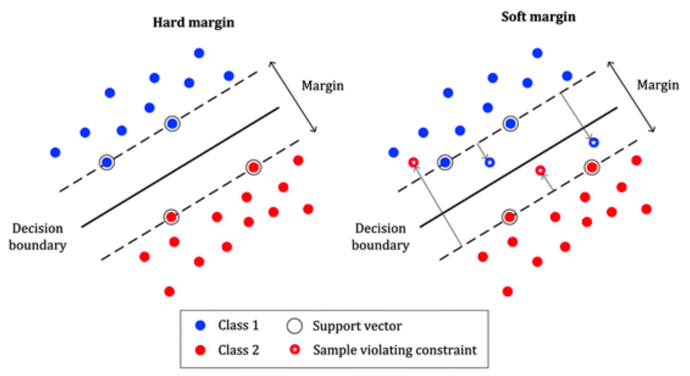
What is the difference between hard margin and soft margin SVM? When would you use each one?

Answer:

The difference between hard margin and soft margin SVM lies in how they handle cases where the data is not linearly separable.

In a hard margin SVM, the algorithm requires that the data be linearly separable without any errors or misclassifications, meaning that there is a hyperplane that perfectly separates the different classes. This approach can work well when the data is well-behaved and noise-free, but it can fail when there are outliers or when the data is inherently noisy.

In contrast, a soft margin SVM allows for some misclassifications or errors on the training data. The goal of the model is to find a hyperplane that separates the two classes as accurately as possible, while still allowing some errors within a certain tolerance level. This tolerance level is controlled using a hyperparameter called C, which determines the trade-off between maximizing the margin and minimizing the error.



Visual Comparison between Hard and Soft Margin

When to use hard margin or soft margin SVM depends on the nature of the problem at hand. Hard margin SVM is appropriate when we have a linearly separable problem with clean data and no outliers. It gives us a guarantee of finding the optimal decision boundary in the feature space. However, it may not work well when the data is noisy or not linearly separable.

On the other hand, soft margin SVM is more flexible and can handle cases where the data is not linearly separable, or when there are outliers in the dataset. By allowing some errors in classification, the model can better generalize to unseen data and avoid overfitting. However, choosing the right value of C can be challenging, and if the tolerance level is set too high, the model may underfit the data.

Question 5:

Is a node's Gini impurity generally lower or greater than its parent's? Is it generally lower/greater, or always lower/greater?

Answer:

A node's Gini impurity is generally lower than its parent's after splitting, but it's not always the case.

The Gini impurity measures the degree of impurity or randomness in a set of samples, where a lower value indicates a higher purity and a higher value indicates a higher level of impurity. When constructing a decision tree, the goal is to split the data at each node in a way that maximally reduces the impurity of the child nodes. The formula to calculate Gini impurity is :

$$Gini(D) = 1 - \sum_{i=1}^{k} p_i^2$$

Where D is a dataset with k classes and p_i indicates the probability of a sample being in ith class.

Thus, when making a binary split, the Gini impurity of the two resulting child nodes should be less than or equal to the Gini impurity of the parent node. However, it is possible that a split could increase the impurity of the child nodes compared to the parent node, although this is generally not desirable.

In practice, decision tree algorithms use heuristics to determine the optimal split points that maximize the reduction in impurity, such as the ID3, C4.5, or CART algorithms. These algorithms select the split point that results in the lowest weighted sum of the impurities of the child nodes, which ensures that the impurity of the child nodes is lower than or equal to the impurity of the parent node.

Therefore, while it is possible to have a case where the Gini impurity of a node's children is higher than its parent's, it is generally true that the Gini impurity of a node's children will be lower than its parent's after splitting.

Question 6:

Is it a good idea to consider scaling the input features if a Decision Tree underfits the training set?

Answer:

Scaling input features is generally not necessary for decision trees because the algorithm does not rely on the magnitude of the values in the features. However, there may be cases where scaling can help improve the performance of a decision tree.

If a decision tree underfits the training set, it means that the model is too simple and is not able to capture the underlying patterns in the data. In this case, it may be helpful to consider scaling the input features to bring them into a similar range or distribution. This can help prevent some features from dominating others and make the tree more sensitive to variations in the data.

In addition, if the input features are on different scales, they may have different weights when splitting the nodes, leading to suboptimal splits. By scaling the features, we can ensure that each feature contributes equally to the decision-making process and improves the quality of the splits.

However, it's important to note that scaling the features may not always improve the performance of the decision tree, especially if the data has a natural scale or if the target variable is not related to the feature magnitudes. In such cases, scaling may even hurt the performance of the model.

Therefore, it's essential to experiment with different preprocessing techniques, including scaling, to find the optimal approach for the specific problem at hand.

Question 7:

How can you use tree-based models for feature selection?

Answer:

There are two main ways which we can use Tree-Based models for feature selection:

1) **Feature Importance:** The first way is to use the feature importance attribute of tree-based models. This attribute measures how important each feature is in determining the target variable. By pruning features whose feature importance is below a certain threshold, we can perform feature selection.

2) **Recursive Feature Elimination:** The second way is to use recursive feature elimination (RFE) with tree-based models. In this approach, we start by training the model on all available features and then recursively remove the least important feature(s). We continue this process until we

reach the desired number of features or until the performance of the model starts to degrade significantly.

Overall, tree-based models are a powerful tool for feature selection as they can provide insights into the importance of each feature and can automate the selection process using RFE.

Question 8:

How do you tweak the hyperparameters of the following model in mentioned circumstances:

- AdaBoost -Underfitting
- Gradient Boosting -Overfitting

Answer:

If AdaBoost is underfitting, we can try the following hyperparameter tuning techniques:

A) **Increase the number of estimators**: AdaBoost works by aggregating multiple weak learners to create a strong learner. If the model is underfitting, it might help to increase the number of weak learners (n_estimators) until we get good performance.

B) **Decrease the learning rate**: The learning rate shrinks the contribution of each weak learner in the model. By decreasing the learning rate, we can allow more weak learners to contribute to the final model and thus improve its performance.

C) **Increase the depth of the base estimator**: AdaBoost uses decision trees as base estimators. If the model is underfitting, it might help to increase the maximum depth of the decision tree to allow for more complex models.

If Gradient Boosting is overfitting, we can try the following hyperparameter tuning techniques:

A) **Regularization**: We can add regularization terms to the objective function being optimized by the gradient boosting algorithm. This will penalize large coefficients and prevent overfitting.

B) **Reduce the learning rate**: Reducing the learning rate (shrinkage) reduces the impact of each individual tree on the final prediction, and can therefore help to reduce overfitting.

C) **Early stopping**: We can stop the training process early if the validation loss plateaus or starts to increase. This can be done by monitoring the performance of the model on a hold-out validation set at regular intervals during training.

D) **Decrease the max depth of the trees**: Gradient boosting also uses decision trees as base estimators. If the model is overfitting, it might help to decrease the maximum depth of each tree to make the model less complex.

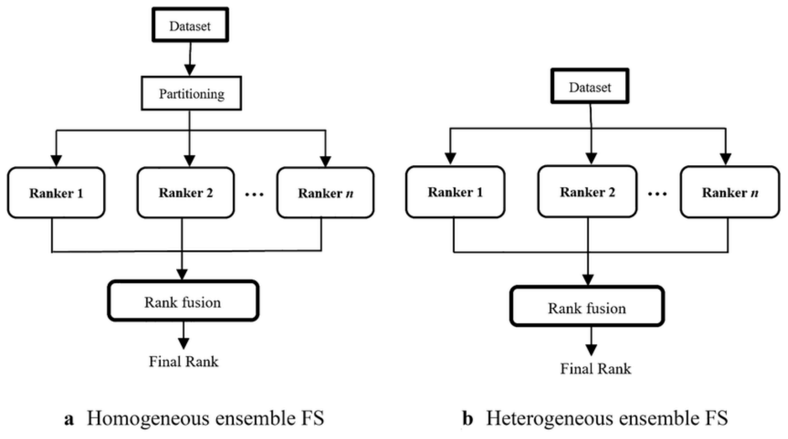
Question 9:

What is the difference between homogeneous and heterogeneous ensembles? Which one is more powerful?

Answer:

Homogeneous ensembles, also known as homogeneous classifiers, are ensembles where each individual model used in the ensemble is of the same type and has the same parameters. For example, a homogenous ensemble might consist of 10 decision trees, all with the same maximum depth and other hyperparameters. The idea behind this approach is to create multiple copies of a good model and combine them to form an even better one.

In contrast, heterogeneous ensembles, also known as heterogeneous classifiers, are ensembles where each individual model used is of a different type and/or has different parameters. For example, a heterogeneous ensemble might consist of a mix of decision trees, k-nearest neighbors (KNN), and support vector machines (SVM) models. The idea behind this approach is to leverage the strengths of different models and combine them to form a more robust prediction.



Diagrams of Homogeneous and Heterogeneous Ensembles

Both homogeneous and heterogeneous ensembles can be powerful, depending on the problem at hand. Homogeneous ensembles tend to be simpler and easier to train, but they may not perform as well on complex problems where different types of models are needed. Heterogeneous ensembles, on the other hand, can be more complex and harder to train, but they have the potential to outperform homogeneous ensembles by leveraging the strengths of multiple models.

In summary, the effectiveness of an ensemble depends on many factors including the nature of the problem, the size of the dataset, and the quality of individual models being combined. In many cases, a well-designed heterogeneous ensemble can outperform a homogeneous ensemble.

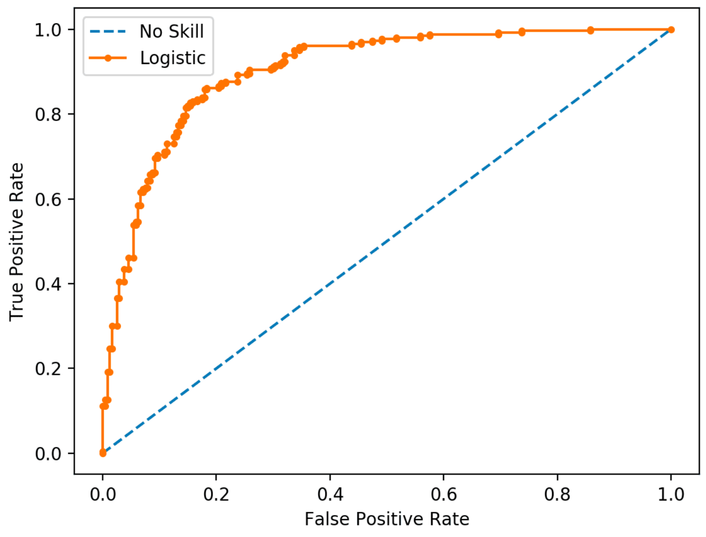
Question 10:

How ROC and AUC are being used in the evaluation of classification performance?

Answer:

An important point which should be considered is ROC (Receiver Operating Characteristic) curve and AUC (Area Under the Curve) are widely used to evaluate the performance of binary classification models.

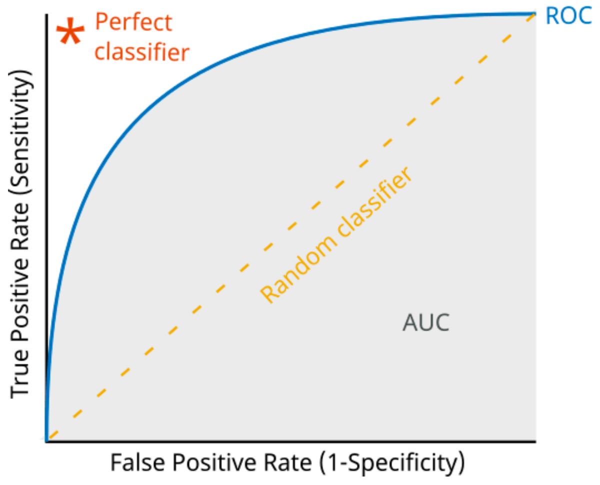
ROC curve is a plot between True Positive Rate (TPR) and False Positive Rate (FPR) at different probability thresholds. TPR is the ratio of correctly classified positive instances to the total positive instances in the dataset, while FPR is the ratio of negative instances that are incorrectly classified as positive to the total negative instances in the dataset.



ROC Curve Instance

AUC represents the area under the ROC curve. It summarizes the overall performance of a binary classifier by providing a single number between 0 and 1, where 0.5 denotes random guessing, 1 denotes perfect classification, and 0 denotes complete misclassification.

In general, higher AUC values indicate better classification performance. A value of 0.5 indicates that the classifier is no better than random guessing, while a value closer to 1 indicates a better performing model.


Perfect classifier
ROC
True Positive Rate (Sensitivity)
Random classifier
AUC
False Positive Rate (1-Specificity)

ROC and AUC

The ROC curve and AUC are particularly useful when dealing with imbalanced datasets, where the number of instances in one class far exceeds the other. In such cases, accuracy can be a misleading metric due to its tendency to favor models that predict the majority class. The ROC curve and AUC provide a more robust evaluation of a classifier's ability to distinguish between the two classes, irrespective of their relative frequencies.

Overall, ROC curves and AUC are important tools for evaluating binary classifiers and comparing the performance of different models.

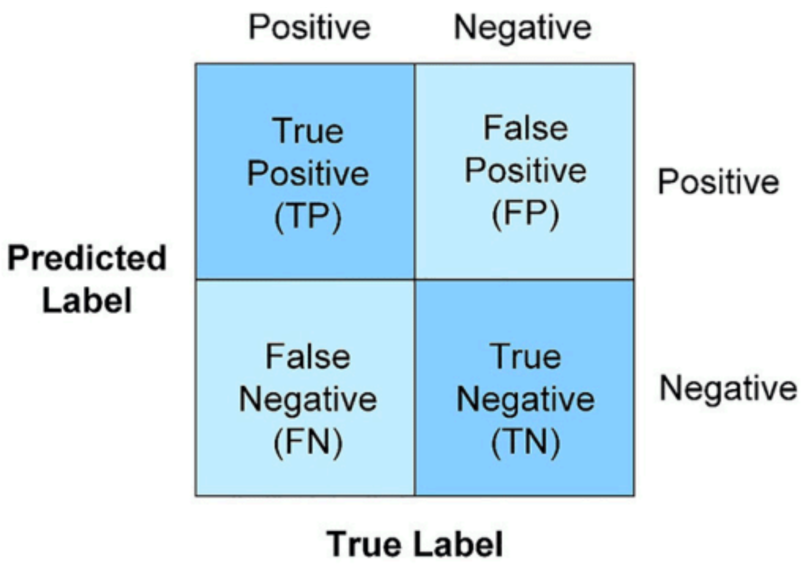
Question 11:

How does the threshold value used in the evaluation of classification performance? This value specifies a cut-off for an observation to be classified as either 0 or 1. Can you explain the trade-off between false positive and false negative rates, and how the choice of threshold value impacts precision and recall?

Answer:

The threshold value is used to define the probability cut-off point for classification in binary classification tasks. For example, if the threshold value is set to 0.5, any observation with a predicted probability greater than or equal to 0.5 will be classified as positive (1), and any observation with a predicted probability less than 0.5 will be classified as negative (0).

The choice of the threshold value can have a significant impact on the performance metrics of a binary classifier. Specifically, there is a trade-off between the false positive rate (FPR) and false negative rate (FNR). Lowering the threshold value will increase the true positive rate (TPR) and decrease FNR, but at the cost of increasing FPR. Conversely, raising the threshold value will decrease TPR and increase FNR, but at the cost of decreasing FPR.



Precision and recall are two commonly used metrics to evaluate the performance of binary classifiers. Precision measures the proportion of correctly identified positive instances among all instances that are predicted as positive, while recall measures the proportion of correctly identified positive instances among all actual positive instances.

The choice of threshold value impacts precision and recall because they depend on the number of true positives, false positives, true negatives, and false negatives. As we change the threshold value, we change the balance between true positives, false positives, true negatives, and false negatives, which affects precision and recall.

For example, if we lower the threshold value, we will identify more positive instances, which will increase recall but may also increase false positives, decreasing precision. On the other hand, if we raise the threshold value, we will identify fewer positive instances, which will decrease recall but may also decrease false positives, increasing precision.

In summary, the choice of threshold value is important in binary classification and should be carefully considered to optimize performance metrics such as precision and recall. It involves a trade-off between the false positive and false negative rates, and its impact on these rates should be carefully evaluated.

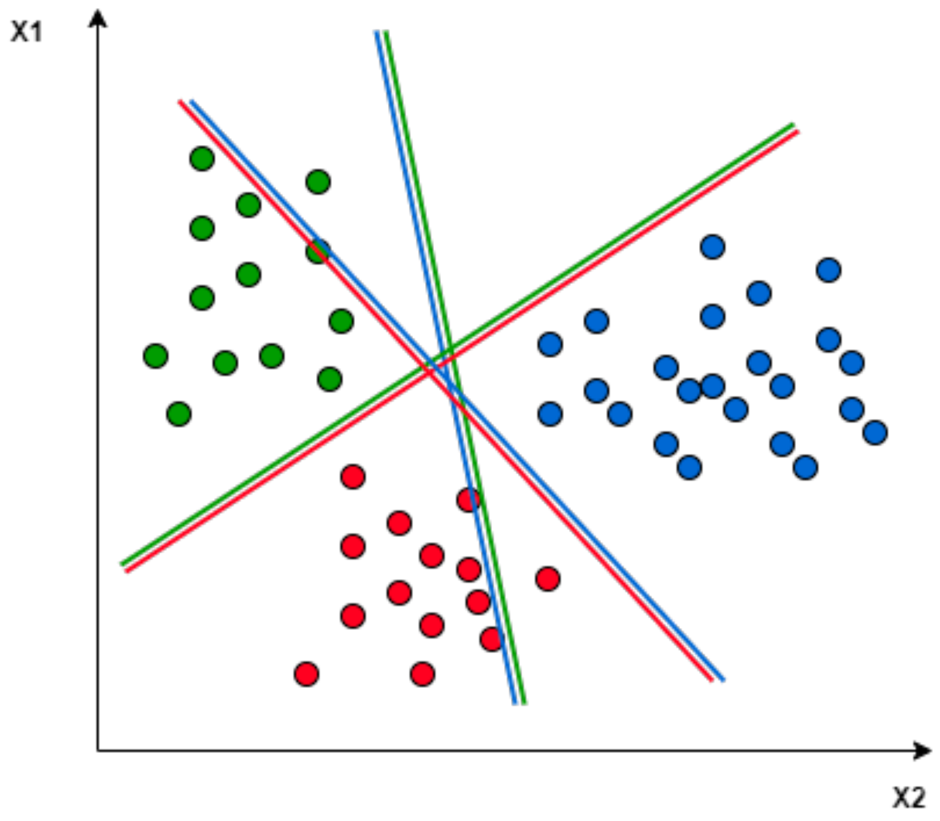
Question 12:

What is the difference between one-vs-one and one-vs-all multi-class classification approaches in classifiers? Under what circumstances would you use one over the other?

Answer:

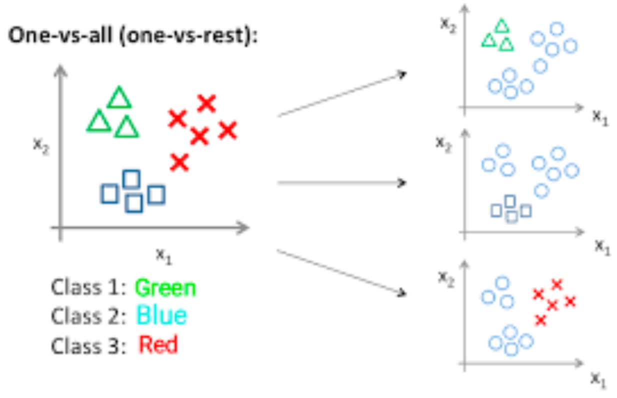
One-vs-one and one-vs-all are two common approaches for performing multi-class classification using binary classifiers.

One-vs-one (OvO): In this approach, a separate binary classifier is trained for each possible pair of classes. For example, if there are k classes, we would train $\frac{k(k-1)}{2}$ binary classifiers. To make a prediction for a new observation, each binary classifier produces a class prediction, and the class with the most votes is selected as the final prediction.



OVO Intuition in SVM

One-vs-all (OvA): In this approach, a separate binary classifier is trained for each class versus all other classes. For example, in a classification problem with K classes, we would train K binary classifiers. To make a prediction for a new observation, each binary classifier predicts whether the observation belongs to its respective class or not. The class with the highest predicted probability is then chosen as the final prediction.



OVA Intuition

The choice between one-vs-one and one-vs-all depends on various factors such as the number of classes, the size of the dataset, and the complexity of the problem. Here are some general guidelines:

- One-vs-one is generally preferred when the number of classes is small (less than 10) and the dataset is large enough to train multiple binary classifiers. This is because OvO provides more fine-grained information about the relationships between different classes.
- One-vs-all is generally preferred when the number of classes is large, and training multiple binary classifiers is computationally expensive. This is because OvA requires only K binary classifiers, regardless of the number of classes, and is therefore more efficient.
- In cases where both OvO and OvA are feasible, OvO may be preferred when the classes are imbalanced, or when there are many overlapping features between different classes. This is because OvO can better handle situations where some pairs of classes are hard to distinguish.

In summary, the choice between one-vs-one and one-vs-all depends on various factors, and there is no one-size-fits-all approach. It is important to carefully evaluate the trade-offs between these two approaches and choose the one that works best for the specific problem at hand.

Question 13:

In this part, you are going to work with the Vehicle Insurance Claim Fraud Detection dataset. You will implement multiple classification models using the Scikit-Learn package to predict if a claim application is fraudulent or not, based on about 32 features. You are expected:

- Perform Exploratory Data Analysis(EDA) on the dataset.
- Try to tackle the problem using the following models:

    - Logistic Regression
    - SVM
    - Decision-Tree
    - Random Forest
    - Other Classifiers: KNN, Naive Bayes, Ensemble Models (Extra Point)
- Use stratified cross-validation to report your models' performance
- Check whether this dataset is imbalanced or not, if yes, try some techniques to overcome this issue. (Including over-sampling, under-sampling, weight-based approaches, etc.)
- Try to boost the performance of the SVM and Random Forest models that you have used in the above section by utilizing various methods (including hyperparameter tuning, different preprocessing methods, feature engineering, etc.). Don't limit yourself only to the aforementioned methods, based on the quality of your work, extra scores may be granted.

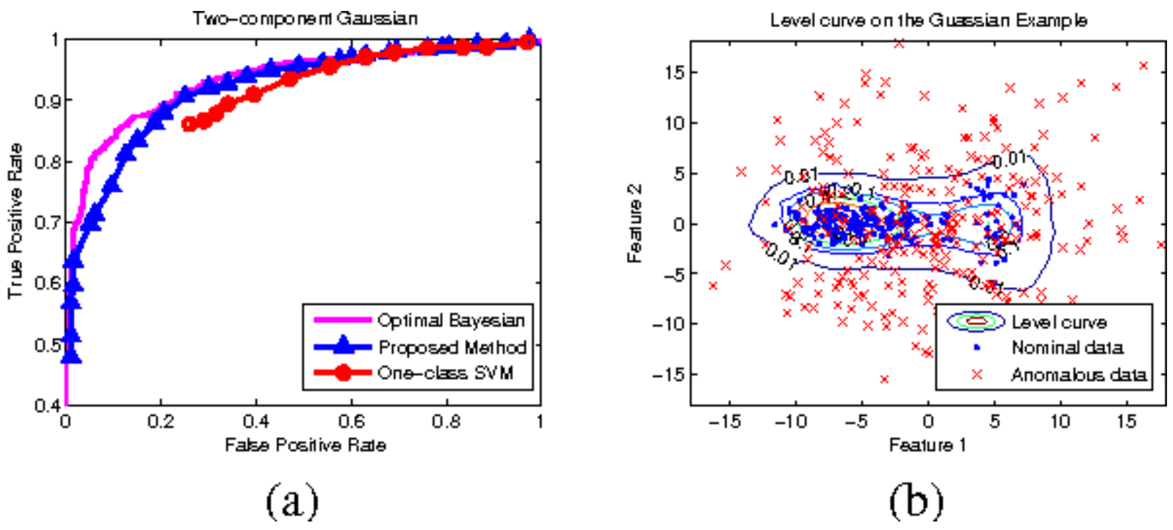
Question 14:

How can you use SVM for anomaly detection? What are the challenges in using SVM for anomaly detection? (Extra Point)

Answer:

Support Vector Machines (SVM) is a popular machine learning algorithm that can be used for anomaly detection. In SVM-based anomaly detection, the algorithm learns a decision boundary that separates normal data from anomalous data.

The basic idea behind SVM-based anomaly detection is that it tries to find a hyperplane that maximally separates the normal data points from the anomalous ones. The distance between the hyperplane and the closest data points on either side is called the margin. The greater the margin, the better the SVM will perform in detecting anomalies.



Using One Class SVM for Anomaly Detection

One of the main challenges in using SVM for anomaly detection is determining the appropriate choice of kernel function and its parameters. Choosing an inappropriate kernel or setting incorrect kernel parameters can lead to poor performance. Another challenge is choosing an appropriate threshold for classifying instances as anomalous or normal. This threshold can have a significant impact on the performance of the SVM-based anomaly detector.

In addition, SVM-based anomaly detection can be computationally expensive, particularly when dealing with large datasets. Finally, another potential challenge is that SVM-based anomaly detectors may struggle with identifying anomalies that are similar to normal data points or where the boundary between normal and anomalous data is not well-defined.

Question 15:

Implement a Bagging Classifier from scratch. You can use sklearn for the base model. Test your model on the Penguins Dataset. (Extra Point)

Question 16:

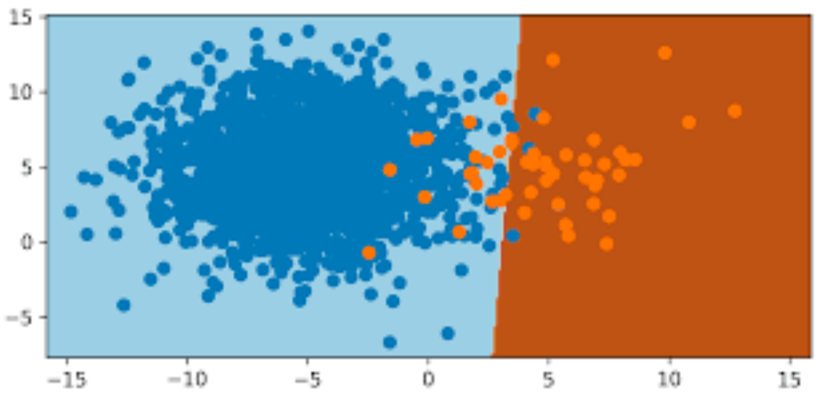
How do you handle the class imbalance in Ensemble Learning? Provide some techniques and explain their working. (Extra Point)

Answer:

Class imbalance is a common problem in many real-world classification problems, where the number of instances in one class is significantly larger than that in another class. Ensemble learning methods can help to address this issue by combining the predictions of multiple models and improving the overall performance of the classifier. Here are some techniques to handle class imbalance in ensemble learning:

- **Resampling**: Resampling techniques involve either oversampling the minority class or undersampling the majority class to create a balanced dataset. Oversampling techniques include random oversampling, synthetic minority oversampling technique (SMOTE), adaptive synthetic sampling (ADASYN), etc. Undersampling techniques include random undersampling, cluster-based undersampling, Tomek links, etc.
- **Cost-sensitive learning**: In cost-sensitive learning, the misclassification costs for different classes are not equal. This can be used to adjust the classification threshold for each class based on its cost. For example, if the cost of misclassifying the minority class is higher than the majority class, then the threshold for the minority class can be lowered to capture more instances of that class.

- **Threshold adjusting**: The decision threshold of an ensemble model can be adjusted to increase the sensitivity of the classifier towards the minority class. This can be done by either decreasing the threshold for the minority class or increasing it for the majority class.
- **Data augmentation**: Data augmentation techniques can be used to generate new training data from the existing data by applying different transformations such as rotation, scaling, translation, etc. This can help to increase the diversity of the data and improve the performance of the classifier.
- **Hybrid approaches**: A combination of the above techniques can be used to handle class imbalance. For example, a hybrid approach of resampling followed by cost-sensitive learning can be used to train an ensemble model that is robust to class imbalance.



Imbalanced Data