Mehrdad Baradaran

.

99222020

.

Assignment 2

.

Master: Mr. Farahani

.

Supervised Learning (Classification)

Q1: 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, an SVM classifier can give a certainty score or likelihood when making expectations on a specific occurrence. The separate between the test occurrence and the choice boundary can be utilized as yield, so we are able utilize that as a certainty score. Be that as it may, we cannot utilize this score to specifically change over it into course probabilities.

Logistic regression could be a measurable strategy for analyzing a dataset in which there are one or more free factors that decide an result. The result is measured with a dichotomous variable (in which there are as it were two conceivable results). In Logistic regression, the subordinate variable is parallel or dichotomous, i.e., it as it were containing information coded as 1 (Genuine) or (Wrong).

On the other hand, SVM may be a directed machine learning calculation that can be utilized for classification or regression. It works by finding the hyperplane that maximizes the margin between two classes. SVM works well with unstructured and semi-structured information like content and pictures.

The most difference between Logistic regression and SVM is that Logistic regression could be a probabilistic model whereas SVM isn't . Logistic regression yields probabilities whereas SVM yields course names.

In case you need a substantial probability score, you'll be able utilize logistic regression. Since SVMs don't create probabilities, you cannot get a likelihood score from fair an SVM.

In Scikit-Learn, there's a way to progress the exactness of the probabilities by employing a strategy called likelihood calibration. Usually done by setting a parameter to Genuine when building the SVM demonstrate. The show at that point performs extra computations after training, utilizing Logistic regression on the SVM scores, to change over the certainty score into dependable probabilities.

Once the likelihood calibration is done, the SVM show can be utilized to form expectations, and the predict_proba() and predict_log_proba() strategies can be utilized to get the probabilities of the predicted classes. This may offer assistance us make more educated choices based on the level of certainty we have within the expectations.

Q2: What actions should you take if you 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 (gamma) or C, or both?

Answer:

If you have trained an SVM classifier using an RBF kernel but notice that it underfits the training set, you can try increasing the value of gamma (γ) or C or both.

Gamma is a parameter of the RBF kernel and controls the shape of the decision boundary. A small gamma will create a more flexible decision boundary and a large gamma will create a more rigid decision boundary.

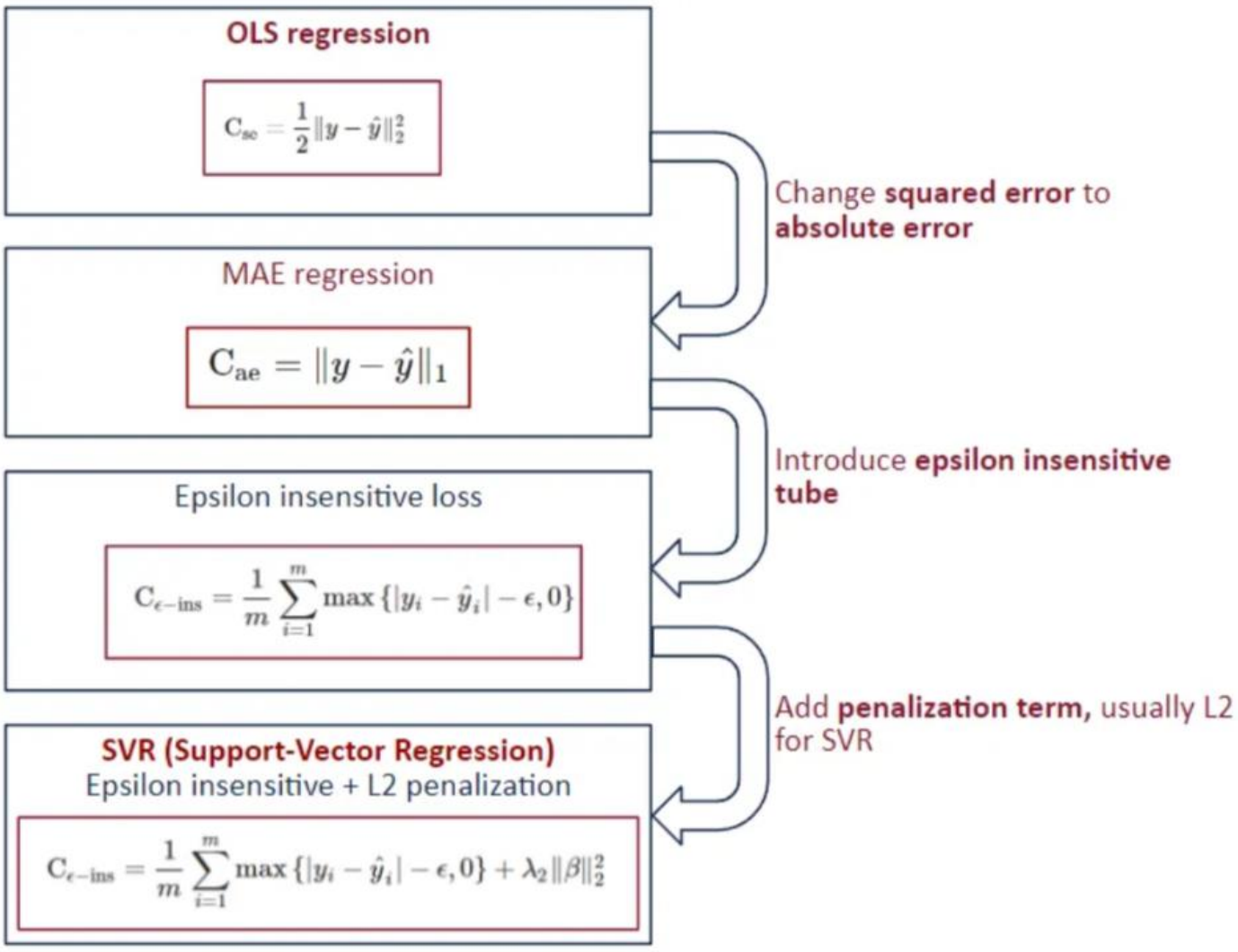
C is the regularization parameter that controls the trade-off between achieving a low training error and a low testing error. A small C will create a wider margin and allow more misclassifications while a large C will create a narrower margin and allow fewer misclassifications.

So, as a conclusion, in order to decreasing underfit, we should increase either gamma or C hyper-parameter.

## Q3: What does it mean for a model to be ε-insensitive?

## Answer:

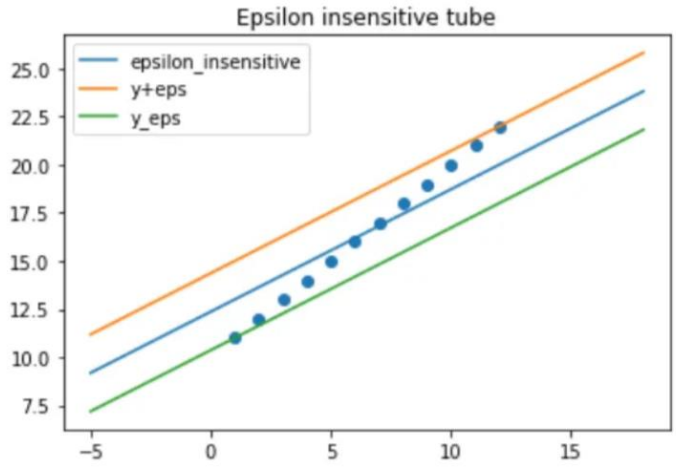
An ε-insensitive model in machine learning only cares about errors that are bigger than a certain amount called epsilon (ε). It punishes those errors more, but ignores smaller errors. This method of measuring loss is often used in support vector regression.

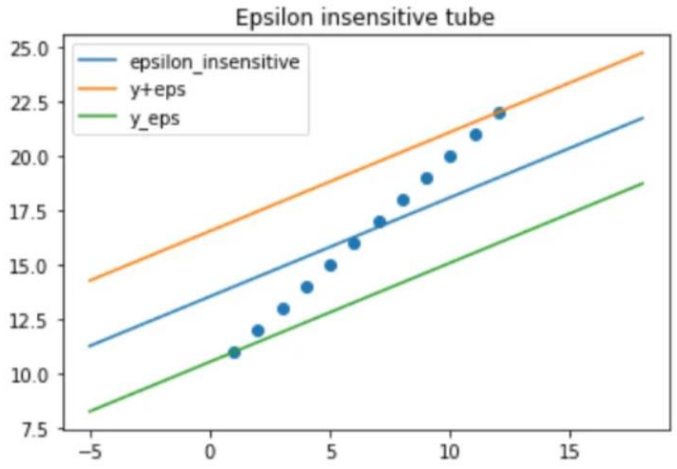



The epsilon-insensitive loss function is a way to measure mistakes in linear models and it helps deal with data that isn't very clear. When you use L2 regularization

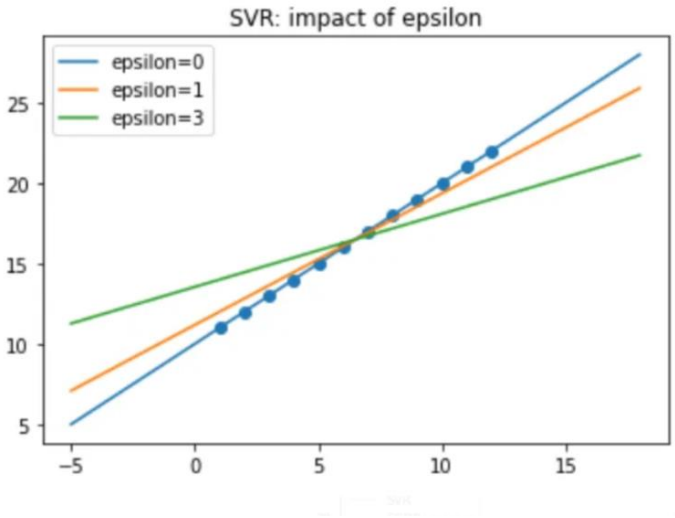
with something called Support Vector Regression (SVR). The idea of the epsilon-insensitive tube in SVR needs a punishment to prevent endless answers. Huber regression is a method that is not very easily affected by extreme values, and it uses positive numbers for big numbers, similar to another method called SVR. When we talk about "epsilon-insensitive," we mean that there is a zone where the error is zero. But even when we look at bigger values, the amount of error can still affect the model a lot.

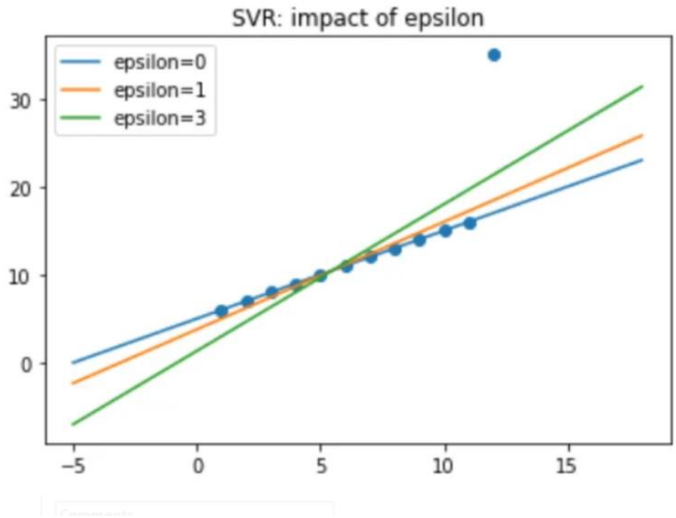
The epsilon-insensitive tube is a way to see how the epsilon-insensitive loss function works in Support Vector Regression (SVR). If epsilon is big enough, all the data points will be inside the tube. This tube will be very strict, like the hard line in SVM classification.



To prevent having too many answers, we need to give out punishments. The aim is to find the answer with the gentlest slope or smallest overall amount. The epsilon-insensitive loss function is good because it doesn't care about tiny mistakes for some data, so the model only needs to focus on important data. This makes the model work better. When you have strange values in your data, a number called epsilon helps decide how important they are. If epsilon is small, the strange values won't have much impact on your results. If epsilon is big, the strange values will have a bigger impact. If epsilon is really big, we use the penalization idea to make the coefficients as small as possible.



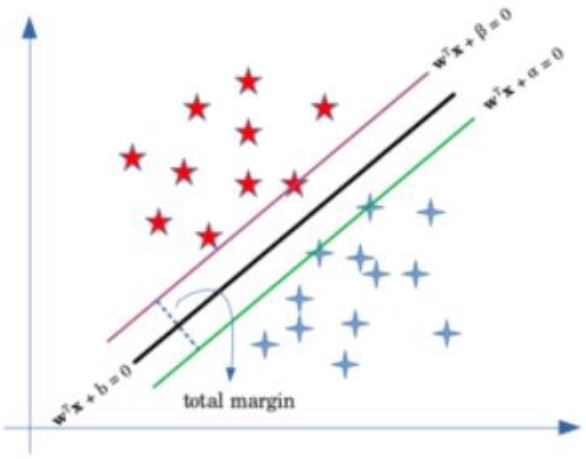
Q4: What is the difference between hard margin and soft margin SVM? When would you use each one?

Answer:

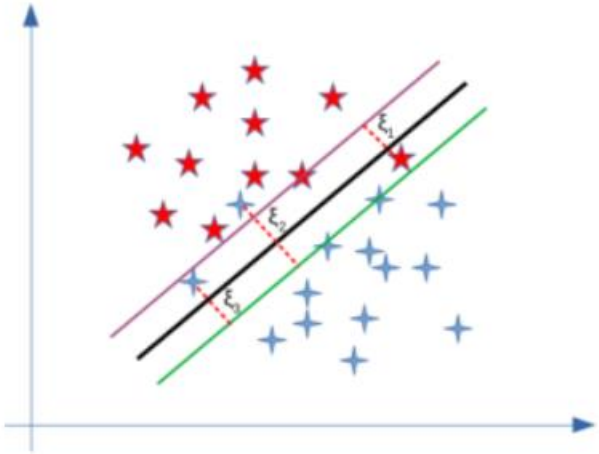
Hard margin and soft margin support vector machines have different abilities when it comes to dealing with data that cannot be separated through a straight line. When the data points are in a straight line and there is space between them, we use Hard margin SVM. It tries to find a line that separates the data perfectly with the most space possible. Soft margin SVM is used when the data can't be completely separated and allows some data points to be wrong.

To deal with datasets that can't be easily separated, the soft margin SVM uses a tool called a slack variable. This gives us some freedom to have data points on the wrong side of the margin or hyperplane. The soft margin SVM is a method of finding a good dividing line between groups of data, even if a few pieces of data are placed on the wrong side. This method can create a line that is not too close to the data points, and works well with new data.

In real life, deciding whether to use hard margin or soft margin SVM depends on the data and the issue you are trying to solve. We should only use hard margin SVM when we are sure that the data can be separated in a straight line. But in real-life data, this is not usually possible. However, soft margin SVM is able to work with datasets that cannot be separated and can create a stronger and more adaptable model. But, if we use a soft margin, there's a higher chance of making mistakes when trying to learn from the data. To avoid this, we need to use the right techniques to make sure we're not overlearning.




Hard margin




Soft margin

Q5: Is a node’s Gini impurity generally lower or greater than its parent’s? Is it generally lower/greater, or always lower/greater?

Answer:

Gini impurity tells us how often we might label something wrong if we choose labels randomly from a dataset.

CART is a computer program that makes decision trees. It looks at different features and picks the best one to split the tree, making the most gain in knowledge. The process divides every part of the data in a way that makes the children groups as pure as possible. This makes sure that a point's Gini impurity is usually less than that of its parent.

Q6: Is it a good idea to consider scaling the input features if a Decision Tree underfits the training set?

Answer:

If a decision tree doesn't fit the training set well, it may or may not help to scale the input features. It depends on why it's not fitting well.

Changing the size of the different parts of the information we use can make a decision tree model work better. This is especially true if the sizes of the parts are very different from each other. Decision trees are a way of organizing information based on certain features. Sometimes, one feature is more important than others and can get in the way of understanding the other features. By making all the characteristics the same size, we can guarantee that they have the same importance and help the tree make decisions fairly.

Making the input features bigger or smaller doesn't always fix the problem of underfitting in decision trees. Decision trees may not work well if there isn't enough information or if the design is too basic. Changing the size of the features may not fix these problems and could even make them worse if the change makes the data lose important information.

So, before you change the size of the input features, it's important to figure out why the decision tree isn't working very well. To make the model better, trying different scaling techniques can help. This way, you can find the method that works best. If there are other problems that are making the model not fit well, we might need to try other things like making the model more complicated, getting more information or using a different method.

## Q7: How can you use tree-based models for feature selection?

## Answer:

We can use tree-based models to select features in different ways.

Feature importance can be measured using tree-based models like Decision Trees, Random Forests, and Gradient Boosted Trees. Feature importance shows how important each feature is for predicting the outcome. You can use this information to pick the most important parts for a model.

Recursive Feature Elimination is a way to pick out the most important features in data. It uses tree-based models to take away features one by one until the best ones are left. First, everything is included. Then, the unimportant things are removed until it reaches a certain number of things. We use tree-based models to discover which features are most important in each step.

Select From Model is a way to select important features in scikit-learn by using a threshold. This method chooses important features by looking at trees and only keeps those with a high importance score.

Correlation-Based Feature Selection (CFS) is a way to pick only the most important features by using models to find features that are similar to each other. CFS finds and gets rid of similar features to make the list of features shorter.

In simpler terms: Tree-based models are a useful and strong tool to choose features for machine learning models. It's important to be careful when selecting features for a model. If you remove important features, it can make the model perform worse.

Q8: How do you tweak the hyperparameters of the following model in mentioned circumstances:

- AdaBoost - Underfitting
- Gradient Boosting – Overfitting

Answer:

When AdaBoost isn't working well, we can try changing some settings called hyperparameters to see if it can improve.

Adding more estimators: If we add more parts to the model, it can learn better and do a better job. We need to be cautious about making the number too high, because it can cause problems with overfitting.

Lowering the learning rate means changing how much each part of the model affects the end result. When we make the learning rate lower, each part of the model becomes less important. This can help the model do a better job at predicting things in general.

If the basic machine learning model is not complicated enough, it might not be able to understand all the details in the data. If we make the foundation of the model more complicated, it could do better.

If we notice that Gradient Boosting is trying too hard to fit the training data, we can make some adjustments called hyperparameters to try to fix this. Here are some things we can try.

Regularization helps to stop overfitting by adding a punishment to the loss function. There are two regular ways to make gradient boosting better, they are L1 and L2 regularization.

To prevent overfitting, reducing the impact of each estimator can be helpful, just like in AdaBoost. This can be done by lowering the learning rate.

Early stopping means stopping the training process of a model when it stops getting better results on a validation set. This can stop the model from learning too much from meaningless information in the data and making mistakes.

If the starting point is too difficult, it might fit the wrong things and cause problems. If we make the basic model simpler, it could make the overall model better at doing its job.

It's important to try out different settings and check how well the model works on a testing set to find the best options for a particular problem.

Q9: What is the difference between homogeneous and heterogeneous ensembles?

Which one is more powerful?

Answer:

There are two kinds of group models used in machine learning called Homogeneous and Heterogeneous ensembles.

A homogeneous group means having many copies of the same kind of base model, like many decision trees or neural networks that look the same. The purpose of homogeneous ensembles is to improve the accuracy of predictions by using multiple copies of the same model together. Some groups of things that are the same throughout are Random Forests and Bagging.

Heterogeneous ensembles are groups of different models. For instance, a mixed group could have a combination of decision trees, logistic regression models or support vector machines. The purpose of using heterogeneous ensembles is to combine the advantages of different models so that the result is a more precise prediction. Some groups of things that are not the same include AdaBoost and Gradient Boosting.

Both groups of things with similar characteristics (homogeneous) and groups with different characteristics (heterogeneous) can both have great potential in different ways. Using a group of similar objects can be simpler and effective for situations that have a definite model to start with. Using different models together can be better when there isn't one clear best model and the data is complicated.

The decision of whether to use a group that is all the same or different depends on the specific problem and data. It's a good idea to try both types of groups and see which one works better for a problem by checking how well they do on a validation test.

Q10: How ROC and AUC are being used in the evaluation of classification performance?

Answer:

ROC curves and AUC are used to measure how well a machine learning algorithm can classify things. Here's a quick explanation of how people use them:

ROC curves show how well a tool or program can distinguish between two options. The ROC curve shows how often a test correctly identifies a positive result versus how often it incorrectly identifies a positive result at different levels of accuracy. The TPR measures how many sick people are correctly identified, and the FPR measures how many healthy people are mistakenly identified as sick. By changing the way we decide if something belongs in a certain group, we can choose how often we're right and how often we're wrong. This can help us reach different levels of success. A very good way to sort things would show all correct guesses and no wrong guesses.

The AUC is a number that shows how well a tool can tell whether something belongs in one group or another. The AUC helps us find out how likely it is that the classifier will rank a positive example higher than a negative example. An AUC score of 0.5 means the classifier is as good as guessing, but a score of 1.0 means the classifier is perfect.

ROC curves and AUC are good tools to see how well a classifier works when there are more of one type of thing than the other. In medical testing, there are usually more healthy people than sick people. Sometimes, being accurate isn't enough. For example, if a classifier always predicted that someone is healthy, it would seem accurate because most people are healthy. But it wouldn't be helpful for finding

sick people. ROC curves and AUC help evaluate how well a classifier can tell the difference between good and bad results at different levels.

Q11: How does the threshold value used in classification affect the model's 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 number used to separate things in groups affects how well the model works. It decides how often the model is wrong either by saying something is correct when it's not (false positive rate), or by saying something is wrong when it's supposed to be right (false negative rate). These are ways to measure how correct the model is.

The threshold value tells us if something is good or bad. It looks at a score and decides if it's positive or negative. If we set the limit at 0.5, anything above it is good and anything below it is bad.

If you change the threshold, the model's FPR and FNR can also change. If we use a lower number for the threshold (like 0.3), we will wrongly identify more things as positive but also miss fewer positive things. If we use a higher number (like 0.7), we will wrongly identify fewer things as positive but also miss more positive things. The FPR is when we wrongly say a negative case is positive, and the FNR is when we wrongly say a positive case is negative.

The threshold you choose also affects how accurate and complete your model is. Precision is how accurate the model is when it predicts positive cases. Recall is how many positive cases the model found compared to how many positive cases there actually are in the data. When we make it easier to classify something as positive, we usually get more true positive results but also more false positives. When we make it harder to classify something as positive, we usually get fewer true positive results but also fewer false positives.

It's important to balance the risk of being wrong in different ways based on what we're doing and how much it would cost if we made a mistake. Suppose a doctor thinks someone is not sick when they actually are, that can be very bad for their health. But if the doctor thinks someone is sick when they aren't, they might get treatments they don't really need. If you want to avoid wrongly identifying something as a problem, it might be better to have a higher standard for what counts as a problem. In cases where we're trying to detect fraud, it's better to have a false positive (thinking fraud happened when it didn't) than a false negative (missing fraud when it really happened). This is because we can always investigate a false positive to make sure it's not fraud, but missing actual fraud could cause big financial problems. Sometimes, it's better to choose a lower limit that makes it more likely to find all relevant results.

The decision of what number to use as a threshold for classifying things depends on what you want to use it for, how much it costs to make mistakes, and how important it is to be very accurate or not miss anything important. Trying out different limit values and checking how well the model works on a test set can help find the best limit for a specific issue.

Q12: What is the difference between one-vs-one and one-vs-all multiclass classification approaches in classifiers? Under what circumstances would you use one over the other?

Answer:

There are two ways to group things into different categories machine learning. One way is to compare two things at a time and see which one fits best. The other way is to compare each thing to all the others and see which one fits best overall.

- One-vs-one (OVO) classification trains a group of classifiers, each comparing two classes at a time. After all classifiers make their predictions, the OVO classification puts their predictions together to make the final decision. If there are k groups of things, we train k(k-1)/2 pairs of classifiers to compare each pair of groups.

- One-vs-all classification is a way to train a computer to recognize different groups of things. It trains a computer to tell apart one group from all the other groups. We choose the classifier that has the highest chance of being right as our final prediction. If there are different types of things to classify (let's say k of them), OVA has to train k sets of classifiers to tell them apart.

The main way these methods are different is in how many machines that tell things apart are created. OVO trains many small groups of classifiers, while OVA trains fewer but bigger groups. Basically, OVO method can be more expensive than OVA, but it can give better predictions if the classifiers and data are good.

To decide which approach to use, it depends on the kind of data and type of classifier being used. OVO is better when there are lots of different types of data and they are easy to tell apart. This is because it can avoid focusing too much on

one type of data and ignoring the others, which OVA might do. OVA is a good choice when one class has more examples than the other or when the classes are similar. This is because it can handle the areas where the classes overlap better.

It's good to try two ways and see which works better on a test. This helps find the best way to solve a problem. Some classifiers prefer one way of doing things more than the other, which can affect which approach is used.

Q14: 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) can find anomaly data by separating them from normal things using a line. SVM is good for finding anomaly data in lots of information and can work out connections between different parts of the information, even if they're not simple.

We can follow these steps to use SVM for spotting anomaly data:

- Teach a computer program to recognize normal data using SVM.
- Use the model we made to guess what category each piece of information belongs in.
- Find the distance from each point to the straight line.
- Put things in groups. If something is far away from the others, consider it anomaly data.

But, using SVM for finding anomaly data can be difficult.

- Choosing the right way the computer solves a problem (kernel function) and its settings (hyperparameters) are very important for SVM (a type of computer program) to work well. When picking a kernel function, it's important to consider if it can deal with relationships that aren't straight lines. It's also crucial to choose hyperparameters that can find a good balance between fitting too perfectly and not fitting enough.

- Figuring out how far something can be before it's considered different can be hard because it depends on the information and the situation. Using a low

threshold can give incorrect results, and using a high threshold can fail to detect real problems.

- Sometimes, anomaly data are not common and only make up a very small part of all the data. This is called imbalanced data. This means that there may be more regular data than unusual data, which could make the dataset uneven. SVM may have difficulty handling imbalanced data and might require alteration or integration with other methods to solve the problem.

- SVM may be difficult to use with a lot of data that has many different parts. We can make it better by using methods to lower the amount of information, like PCA or picking specific features.

SVM is a good method for finding things that are anomaly data, but we need to choose things like the type of math we use and how far apart things need to be in order to get really good results.

Q16: How do you handle the class imbalance in Ensemble Learning? Provide some techniques and explain their working. (Extra Point)

Answer:

In machine learning, class imbalance happens when there are way less examples of one type of thing compared to others. This means that some models might only focus on the most common class and not perform well for the less common one. Ensemble learning is when you use more than one model to improve how well you can tell the difference between things. This is helpful when there are more of one thing than another, as it helps you to better classify them. Here are some ways to deal with uneven class sizes in group learning:

Resampling means changing the training data by making more or less of certain categories. This helps make the data more balanced. You can make more samples using methods like SMOTE or ADASYN. You can do undersampling in different ways, like randomly selecting some samples, or using something called Tomek links. These methods can make sure each class has a fair amount of training data and make the group work better.

Cost-sensitive learning means giving different costs for wrong classifications to different categories in order to handle class imbalance. In group learning, we can make the minority class more important during training by giving it higher weights. This helps to improve its prediction in the end. You can do this by using methods such as weighted random forests or boosting algorithms like AdaBoost.

A group of classifiers can be trained together, where each one learns from a different part of the training data. We can use two methods to help machines learn better: bagging and boosting. Bagging means we train machines using different parts of data. Boosting means we train machines using the wrong answers from

the previous machine. The last guess is made by putting together all the guesses of the classifiers.

Detecting things that are very different from what's normal is called anomaly detection. We use this technique to find anomalies. You can use these examples to teach a machine or leave them out to make sure the results are fair.

You can use one or many ways to deal with unequal class numbers in group learning. The way we do things depends on the information we have and the computer program we are using to learn from it. Try out different methods and test how well they work on a test group. This will help you figure out the best way to solve a particular problem.