**Support Vector Machines**

A Support Vector Machine (SVM) is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection. It is one of the most popular models in Machine Learning, and any‐ one interested in Machine Learning should have it in their toolbox. SVMs are partic‐ ularly well suited for classification of complex but small- or medium-sized datasets.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



**Types of SVM**

SVM can be of two types:

* **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
* **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

**NOTE:**

**Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

**Support Vectors:**

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

**How does SVM works?**

**Linear SVM:**

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image.

So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes.



Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



**Non-Linear SVM:**

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

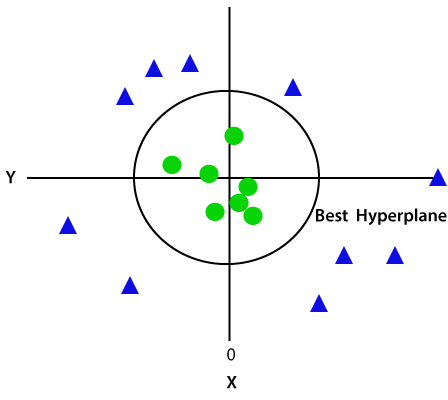
z=x2 +y2

By adding the third dimension, the sample space will become as below image:

So now, SVM will divide the datasets into classes in the following way. Consider the below image:

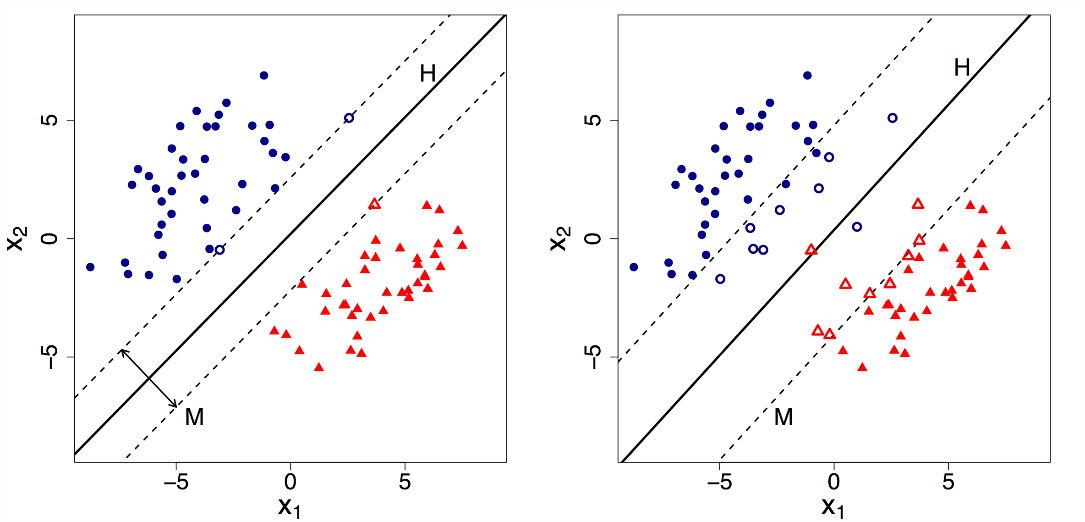


Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

## Hard and Soft SVM



We can now clearly state that HP1 is a Hard SVM(left side) while HP2 is a Soft SVM(right side).

*By default, Support Vector Machine implements Hard margin SVM*. It works well only if our data is linearly separable.

Hard margin SVM does not allow any misclassification to happen.

In case our data is non-separable/ nonlinear then the Hard margin SVM will not return any hyperplane as it will not be able to separate the data. Hence this is where Soft Margin SVM comes to the rescue.

Soft margin SVM allows some misclassification to happen by relaxing the hard constraints of Support Vector Machine.

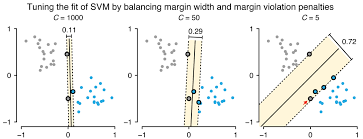
Soft margin SVM is implemented with the help of the **Regularization parameter (C).**

**Regularization parameter (C):**It tells us how much misclassification we want to avoid.

– *Hard margin SVM generally has large values of C.*

– *Soft margin SVM generally has small values of C.*

Now that we know what the Regularization parameter (C) does. We need to understand its relation with Support Vector Machine.



– As the value of C increases the margin decreases thus Hard SVM.

– If the values of C are very small the margin increases thus Soft SVM.

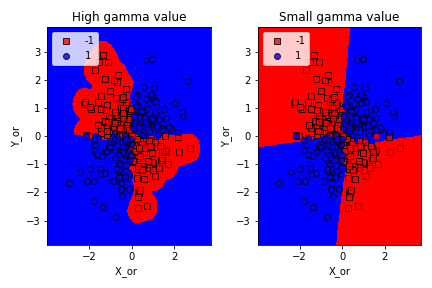
– Large value of C can cause overfitting therefore we need to select the correct value using Hyperparameter Tuning.

## Other Parameters of SVM

Other significant parameters of Support Vector Machine are the **Gamma** values. It tells us how much will be the influence of the individual data points on the decision boundary.

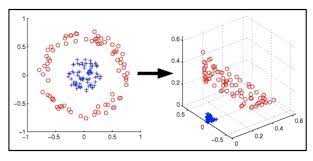
– Large Gamma: Fewer data points will influence the decision boundary. Therefore, decision boundary becomes non-linear leading to overfitting

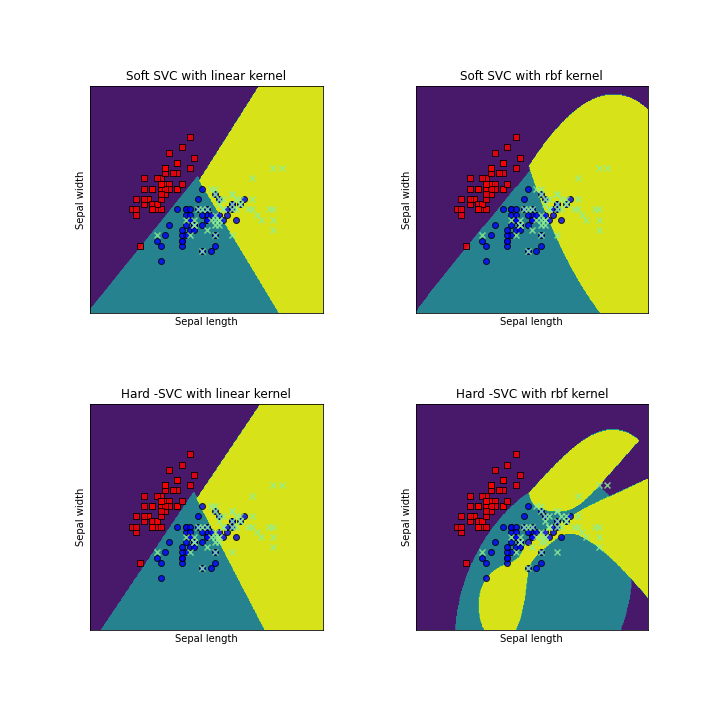
– Small Gamma: More data points will influence the decision boundary. Therefore, the decision boundary is more *generic.*



## Kernel -trick in SVM

Support Vector Machine deals with nonlinear data by transforming it into a higher dimension where it is linearly separable. Support Vector Machine does so by using different values of Kernel. We have various options available with kernel like, **‘linear’, “rbf”, ”poly”** and others (default value is “rbf”). *Here “rbf” and “poly” are useful for non-linear hyper-plane.*





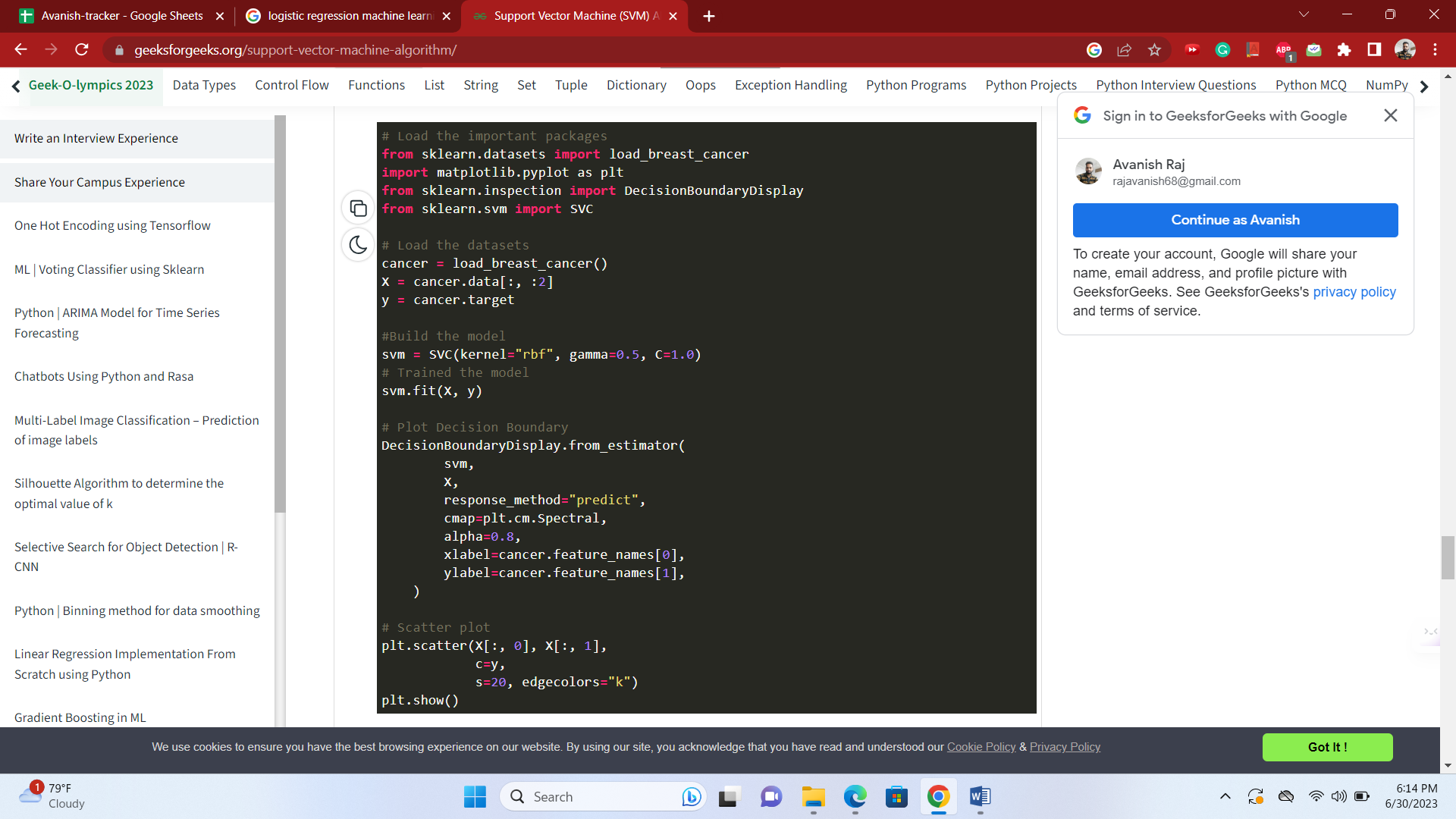
From the above figure, it is clear that choosing the right kernel is very important in order to get the correct results.

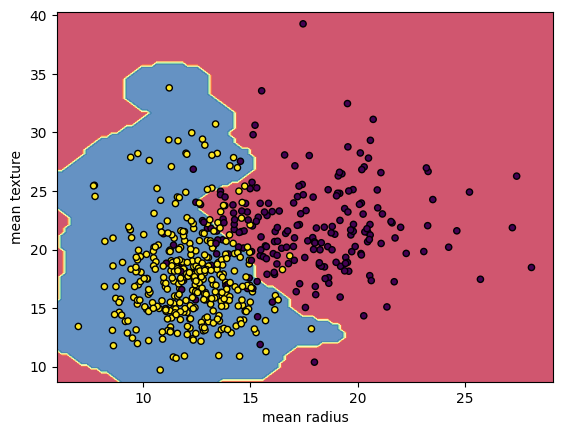
### **SVM implementation in Python**

Predict if cancer is Benign or malignant. Using historical data about patients diagnosed with cancer enables doctors to differentiate malignant cases and benign ones are given independent attributes.

#### Steps

* Load the breast cancer dataset from sklearn.datasets
* Separate input features and target variables.
* Buil and train the SVM classifiers using RBF kernel.
* Plot the scatter plot of the input features.
* Plot the decision boundary.

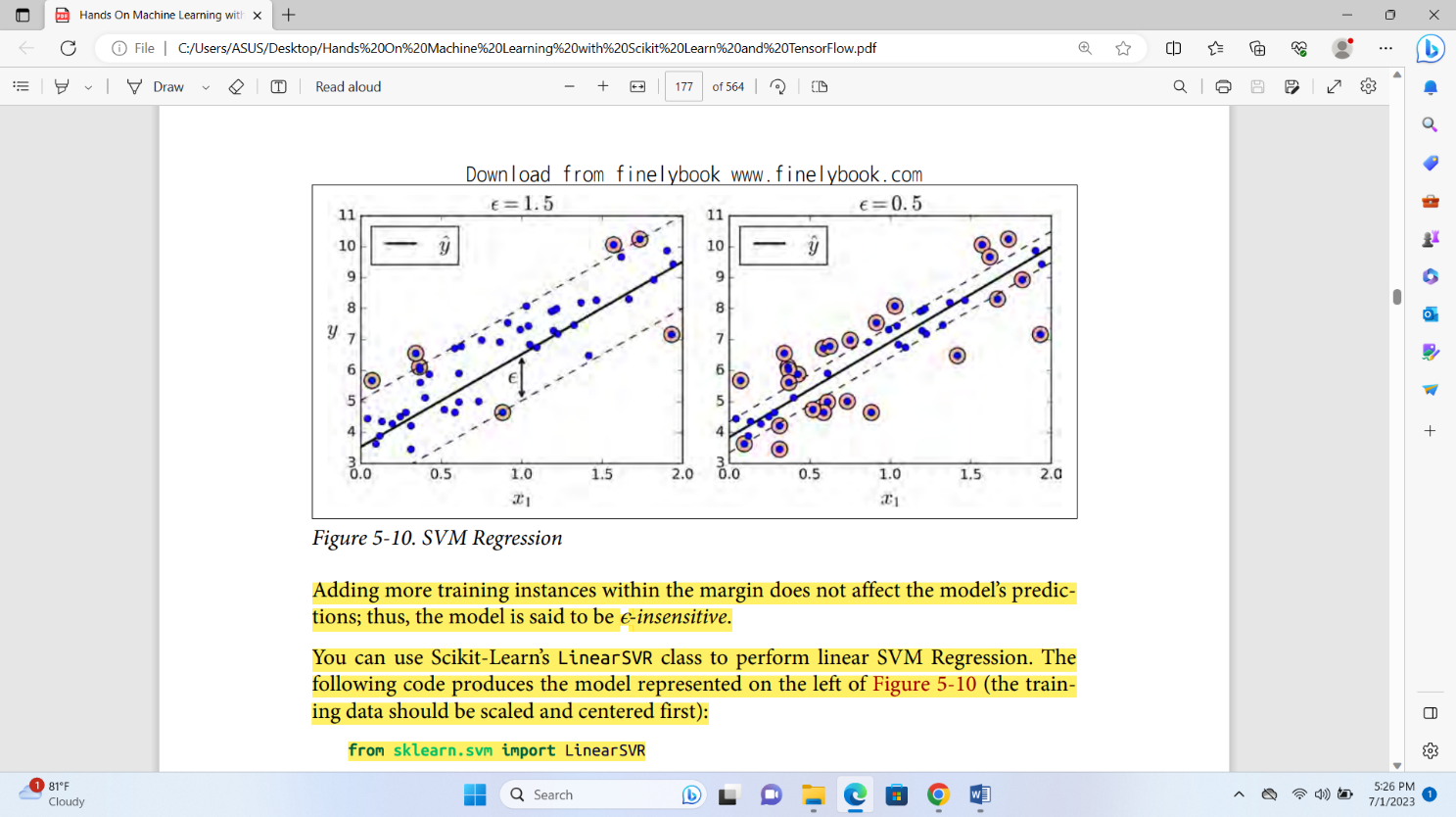




*Breast Cancer Classifications with SVM RBF kernel*

**SVM Regression**

As we mentioned earlier, the SVM algorithm is quite versatile: not only does it sup‐ port linear and nonlinear classification, but it also supports linear and nonlinear regression. The trick is to reverse the objective: instead of trying to fit the largest pos‐ sible street between two classes while limiting margin violations, SVM Regression tries to fit as many instances as possible on the street while limiting margin violations (i.e., instances o the street). The width of the street is controlled by a hyperparame‐ ter ϵ. Figure 5-10 shows two linear SVM Regression models trained on some random linear data, one with a large margin (ϵ = 1.5) and the other with a small margin (ϵ = 0.5).

****

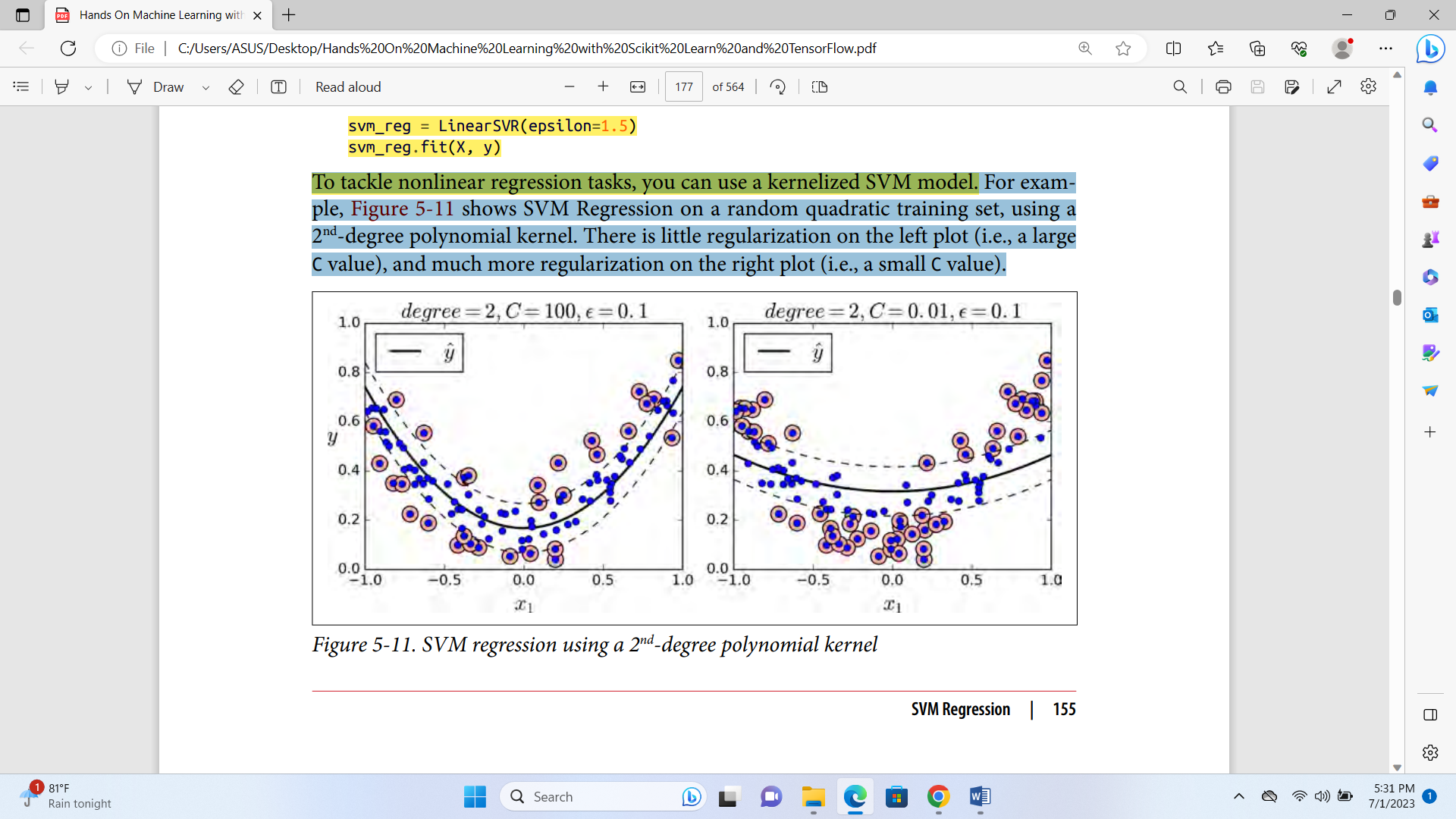
Adding more training instances within the margin does not affect the model’s predic‐ tions; thus, the model is said to be ϵ-insensitive. You can use Scikit-Learn’s LinearSVR class to perform linear SVM Regression. The following code produces the model represented on the left of Figure 5-10 (the training data should be scaled and centered first):

from sklearn.svm import LinearSVR

svm\_reg = LinearSVR(epsilon=1.5)

svm\_reg.fit(X, y)

To tackle nonlinear regression tasks, you can use a kernelized SVM model. For exam‐ ple, Figure 5-11 shows SVM Regression on a random quadratic training set, using a 2 nd-degree polynomial kernel. There is little regularization on the left plot (i.e., a large C value), and much more regularization on the right plot (i.e., a small C value).



The following code produces the model represented on the left of Figure 5-11 using Scikit-Learn’s SVR class (which supports the kernel trick). The SVR class is the regres‐ sion equivalent of the SVC class, and the LinearSVR class is the regression equivalent of the LinearSVC class. The LinearSVR class scales linearly with the size of the train‐ ing set (just like the LinearSVC class), while the SVR class gets much too slow when the training set grows large (just like the SVC class).

from sklearn.svm import SVR

svm\_poly\_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)

svm\_poly\_reg.fit(X, y)

**SVM classification vs SVM regression**

SVM can be used for both classification and regression tasks. The main difference between SVM classification and SVM regression lies in their objectives and how they handle the target variable.

SVM Classification:

In SVM classification, the objective is to separate the data into distinct classes by finding an optimal hyperplane that maximizes the margin between the classes. The target variable in SVM classification is categorical or discrete, representing different classes or labels. The SVM algorithm aims to find the decision boundary that best separates the classes in the feature space.

SVM Regression:

In SVM regression, the objective is to predict a continuous target variable rather than class labels. SVM regression aims to find a function that approximates the relationship between the input features and the target variable. Instead of maximizing the margin between classes, SVM regression focuses on finding a hyperplane that fits as many data points within a specified margin.

To achieve this, SVM regression introduces a margin called the ε-insensitive tube around the target variable. Data points falling within this tube are considered correctly predicted, while points outside the tube contribute to the loss function. The goal is to minimize the errors while keeping as many data points within the tube as possible.

Additionally, SVM regression utilizes a loss function and a regularization term to balance the trade-off between fitting the training data and controlling model complexity. The regularization term helps to avoid overfitting and control the flatness or smoothness of the fitted function.

In summary, while SVM classification focuses on finding the optimal decision boundary to separate classes, SVM regression aims to find a function that approximates the relationship between features and a continuous target variable, while considering a margin around the target variable. The specific formulations and optimization procedures differ between SVM classification and SVM regression, tailored to their respective objectives and target variable types.

**Assumptions, advantages and disadvantages of SVM**

Support Vector Machines (SVMs) are a popular machine learning algorithm used for classification and regression tasks. Here are some assumptions, advantages, and disadvantages of SVMs:

Assumptions:

1. SVM assumes that the data points are linearly separable. However, with the use of kernel functions, SVMs can handle nonlinear data as well.
2. SVM assumes that the training examples provided are representative of the entire dataset.
3. SVM assumes that the data is independent and identically distributed (i.i.d).

Advantages of SVMs:

1. Effective in high-dimensional spaces: SVMs perform well in datasets with a large number of features, even when the number of features exceeds the number of samples. This makes them suitable for tasks involving text classification, gene expression analysis, and image recognition.
2. Robust against overfitting: SVMs are less prone to overfitting due to the concept of maximum margin, which encourages the model to find the best decision boundary that separates the classes while maximizing the distance between the boundary and the data points.
3. Versatile: SVMs can be used for both classification and regression tasks. For classification, SVMs can handle binary as well as multi-class problems through techniques such as one-vs-one and one-vs-all. For regression, SVMs can be used to approximate functions by fitting a hyperplane with a soft margin.

Disadvantages of SVMs:

1. Computational complexity: SVMs can be computationally expensive, especially for large datasets. Training an SVM requires solving a quadratic programming problem, which can be time-consuming.
2. Sensitivity to parameter selection: SVMs have hyperparameters that need to be tuned, such as the regularization parameter (C) and the choice of kernel function. The performance of an SVM can be sensitive to these parameter settings, and selecting optimal values often requires experimentation and cross-validation.
3. Lack of interpretability: SVMs produce black-box models, meaning they provide little insight into the underlying reasons for their predictions. It can be challenging to understand the decision-making process of an SVM, unlike some other algorithms like decision trees or logistic regression.

It's important to note that the advantages and disadvantages listed above are general observations, and the suitability of SVMs depends on the specific problem and dataset at hand.

**How does SVM vary when datasets are large and small**

The behavior of Support Vector Machines (SVMs) can vary when dealing with large and small datasets. Here's how SVMs can be affected by the dataset size:

1. Large datasets:
   * Computational complexity: SVMs can be computationally expensive for large datasets, as the training time generally increases with the number of training examples. SVMs require solving a quadratic programming problem, which can be time-consuming when the dataset size is large. This can lead to longer training times and increased memory requirements.
   * Generalization performance: In general, larger datasets provide more representative samples, reducing the risk of overfitting. SVMs can benefit from large datasets by learning more robust decision boundaries, leading to improved generalization performance. With a large number of examples, SVMs can better capture the underlying patterns and make more accurate predictions.
   * Kernel selection: In large datasets, using complex kernel functions can be computationally expensive. SVMs with simple linear kernels or computationally efficient kernel approximations (e.g., using the Nyström method) are often preferred to reduce the computational burden.
2. Small datasets:
   * Overfitting risk: SVMs have the potential to overfit small datasets, especially when the number of features is large compared to the number of training examples. The margin-based formulation of SVMs aims to find the hyperplane with the maximum margin, which can result in models that are highly sensitive to individual data points, leading to overfitting. Regularization techniques, such as adjusting the regularization parameter (C) or using a soft-margin SVM, can help mitigate overfitting in small datasets.
   * Limited data representation: Small datasets may not fully represent the underlying distribution, resulting in limited coverage of the input space. This can make it challenging for SVMs to find an optimal decision boundary. In such cases, feature engineering or the use of nonlinear kernels (e.g., polynomial or Gaussian kernels) can help capture more complex relationships and improve the performance of SVMs on small datasets.
   * Cross-validation: When working with small datasets, it becomes crucial to employ proper evaluation techniques, such as k-fold cross-validation, to obtain reliable estimates of the model's performance. Cross-validation allows for better assessment of the SVM's generalization capability and helps in parameter tuning.

In summary, while SVMs can be effective for both large and small datasets, they present computational and overfitting challenges in large datasets, while in small datasets, the risk of overfitting and limited data representation become more prominent. Careful parameter selection, regularization, and appropriate evaluation techniques are important considerations for achieving good performance with SVMs in both cases.

**SVM sensitivity to outlier**

Support Vector Machines (SVMs) can be sensitive to outliers in the dataset. An outlier is an observation that significantly deviates from the other data points. Here's how outliers can affect SVMs:

1. Influence on the decision boundary: SVMs aim to find the optimal decision boundary that maximizes the margin between classes. Outliers, especially those close to the decision boundary, can have a substantial impact on the placement of the boundary. An outlier with a large deviation can distort the boundary, causing it to be biased towards the outlier or even misclassify other instances.
2. Margin violations: In SVMs, instances that lie within or on the margin are referred to as support vectors. These support vectors play a crucial role in defining the decision boundary. Outliers that fall within the margin or violate the margin can change the position of the decision boundary and influence the margin width. SVMs strive to maximize the margin, and outliers that violate the margin can lead to less robust or suboptimal decision boundaries.
3. Increased complexity: Outliers introduce noise and can disrupt the underlying patterns in the data. SVMs try to fit the data while maximizing the margin, which means they may attempt to accommodate outliers and capture their patterns. This can result in complex and less interpretable models that may not generalize well to new, unseen data.

To mitigate the sensitivity of SVMs to outliers, several approaches can be considered:

1. Outlier detection and removal: Prior to training an SVM, identifying and removing outliers can be beneficial. Outlier detection techniques, such as the Z-score method, the Mahalanobis distance, or clustering-based approaches, can help identify potential outliers. Removing outliers that are deemed truly anomalous can help create a cleaner dataset and improve SVM performance.
2. Robust kernels: SVMs can use different kernel functions to handle nonlinear data. Some kernel functions, such as the Radial Basis Function (RBF) kernel, are more sensitive to outliers. However, using robust kernels, such as the robust RBF kernel or the Huberized loss function, can provide increased robustness to outliers by reducing their influence on the decision boundary.
3. Soft-margin SVM: The standard SVM formulation aims to find a hard-margin decision boundary, which is highly sensitive to outliers. However, using a soft-margin SVM with a regularization parameter (C) allows for some misclassifications and margin violations. This regularization parameter can be adjusted to control the balance between maximizing the margin and allowing for margin violations caused by outliers.
4. Data preprocessing and feature scaling: Preprocessing the data and applying feature scaling techniques, such as normalization or standardization, can help reduce the impact of outliers. Scaling the features can make the SVM less sensitive to extreme values, potentially reducing the influence of outliers.

It's important to note that the appropriate approach to handling outliers in SVMs depends on the specific problem and the nature of the outliers. Careful consideration should be given to the dataset and the desired trade-off between capturing outlier patterns and generalizing to new data.

**Effect of missing values on SVM**

Missing values can have an impact on Support Vector Machines (SVMs) and may require careful handling. Here's how missing values can affect SVMs:

1. Handling missing values: SVMs typically assume that the input data is complete and does not contain missing values. Therefore, one common approach is to pre-process the data and handle missing values before applying SVMs. Some possible strategies for handling missing values include:
   * Deleting instances: If the number of instances with missing values is relatively small, it may be feasible to remove those instances from the dataset. However, this approach may result in loss of information and may not be suitable if the missing values are not randomly distributed.
   * Imputation: Missing values can be imputed or replaced with estimated values. Common imputation methods include mean imputation (replacing missing values with the mean of the feature), regression imputation (predicting missing values using regression models), or model-based imputation (using probabilistic models to estimate missing values). Imputation can help preserve the overall structure of the data but introduces some level of uncertainty.
   * Indicator variables: Another approach is to introduce indicator variables that represent the presence or absence of missing values for each feature. This approach allows the SVM to capture potential patterns associated with missing values as a separate category. However, it increases the dimensionality of the feature space.
2. Impact on model performance: Missing values can introduce bias in the SVM's learning process and affect the performance of the model. If missing values are associated with specific classes or patterns in the data, the absence of those values may impact the decision boundary learned by the SVM. In such cases, imputation or handling missing values appropriately becomes important to minimize the potential bias and maintain the integrity of the learning process.
3. Sensitivity to imputation methods: The choice of imputation method can influence the performance of SVMs. Different imputation techniques have their own assumptions and limitations. The imputation method selected should be appropriate for the nature of the missing values and the characteristics of the dataset. It is important to carefully evaluate and compare the performance of SVMs using different imputation methods to ensure the imputations do not introduce artificial patterns or bias into the data.
4. Consideration of missingness as a feature: In some cases, the missingness of values itself may contain valuable information. Instead of imputing missing values, treating the missingness indicator as a separate feature may help the SVM capture patterns associated with the absence of values. This approach can be useful if the missingness mechanism is informative and can be meaningful in the context of the problem.

It's important to note that the choice of how to handle missing values in SVMs depends on the specifics of the dataset, the extent and pattern of missingness, and the specific objectives of the analysis. The handling of missing values should be done carefully and with consideration for the potential impact on the model's performance and the validity of the results.

**Effect of correlation on SVM**

Correlation among features in a dataset can have an impact on Support Vector Machines (SVMs). Here's how correlation can affect SVMs:

1. Redundancy and multicollinearity: High correlation between features indicates that they are redundant or highly related. In such cases, the presence of highly correlated features can introduce multicollinearity, which can adversely affect SVMs. Multicollinearity can lead to instability in the model estimation, making it difficult for SVMs to accurately determine the importance of each feature and the optimal decision boundary. It may also cause instability in the feature weights or coefficients, making the model less interpretable.
2. Feature selection: Correlated features can affect feature selection methods used in SVMs. If two or more features are highly correlated, they may provide similar information, and including all of them in the model may not necessarily improve the performance. In feature selection techniques, correlated features may compete with each other, leading to inconsistency in the selected features. Consequently, it is important to consider the correlation among features when performing feature selection to avoid redundant or irrelevant features and to improve model performance.
3. Model interpretability: SVMs are often valued for their interpretability, as they can provide insights into the decision boundary and the importance of different features. However, high correlation among features can make it challenging to interpret the model. Correlated features may have similar coefficients or weights, making it difficult to determine their individual contributions to the decision boundary. This can reduce the interpretability and understandability of the SVM model.
4. Dimensionality reduction: When dealing with highly correlated features, dimensionality reduction techniques can be beneficial. Techniques such as Principal Component Analysis (PCA) or other feature extraction methods can help in reducing the dimensionality while capturing the essential information present in the correlated features. By transforming the original features into a lower-dimensional space, these techniques can reduce the impact of correlation and improve the performance of SVMs.
5. Regularization and parameter tuning: High correlation among features can affect the regularization and hyperparameter tuning process in SVMs. In cases of multicollinearity, the model may have difficulties in determining the appropriate regularization parameter or C value. The choice of hyperparameters can be influenced by the interplay between correlated features, making the tuning process more challenging. Careful consideration and experimentation with different hyperparameter values are necessary to find the optimal balance and avoid overfitting or underfitting caused by correlation.

In summary, high correlation among features can introduce challenges in SVMs, such as redundancy, multicollinearity, reduced interpretability, and difficulties in feature selection and hyperparameter tuning. Handling correlation appropriately, through techniques like dimensionality reduction or careful feature selection, can help mitigate these challenges and improve the performance and interpretability of SVM models.

**Feature Engineering, Feature Selection and Feature Importance for SVM algorithm**

Feature engineering, feature selection, and feature importance are important steps in preparing data for the SVM algorithm. Here's a breakdown of each step and its relevance to SVM:

1. Feature Engineering:
   * Feature engineering involves transforming or creating new features from the existing ones to improve the representation of the data and enhance the SVM's performance.
   * Techniques such as polynomial features, logarithmic transformations, scaling, or normalization can be applied to the features to improve their distribution or capture nonlinear relationships.
   * Domain knowledge and understanding of the problem can guide the creation of meaningful features that capture relevant information.
2. Feature Selection:
   * Feature selection aims to identify a subset of the most relevant features for training the SVM model while discarding redundant or irrelevant features.
   * Irrelevant or noisy features can negatively impact the SVM's performance, increase computation time, and introduce overfitting.
   * Techniques like univariate feature selection, recursive feature elimination, or lasso regularization can be employed to select a subset of features based on statistical measures, model performance, or regularization.
3. Feature Importance:
   * Feature importance refers to quantifying the contribution of each feature towards the SVM's decision-making process.
   * Understanding feature importance can help in identifying the most influential features and gaining insights into the underlying data patterns.
   * Various methods can estimate feature importance, such as analyzing feature coefficients or weights in linear SVMs, feature permutation importance, or using decision tree-based methods like random forests or gradient boosting.

The relevance of these steps to SVM lies in improving model performance, reducing dimensionality, and enhancing interpretability. By engineering informative features, selecting relevant ones, and determining feature importance, the SVM model can be fine-tuned and optimized for better results. It is important to note that the specific choice and application of these steps depend on the dataset, problem domain, and the characteristics of the features themselves. It may require experimentation and careful evaluation to determine the most effective combination of feature engineering, selection, and importance estimation techniques for a given SVM application.

**Overfitting handling in SVM**

Handling overfitting in Support Vector Machines (SVMs) involves similar principles to other machine learning algorithms. SVMs are powerful models for both classification and regression tasks, and they can suffer from overfitting if not properly managed. Here are strategies to handle overfitting in SVMs:

1. **Regularization (C parameter):**
   * SVMs have a regularization parameter, often denoted as 'C'. A smaller C value encourages the model to have a larger margin even if it means more training points are misclassified. This helps in preventing overfitting by controlling the trade-off between fitting the training data perfectly and generalizing to new data.
2. **Kernel Selection:**
   * SVMs can use different kernels (e.g., linear, polynomial, radial basis function) to map data into higher-dimensional spaces. Using complex kernels, especially with high degrees, can lead to overfitting. Start with simpler kernels and gradually experiment with more complex ones if needed.
3. **Feature Selection:**
   * Similar to other models, selecting relevant features is crucial to prevent overfitting in SVMs. Include only the most informative features and discard irrelevant ones.
4. **Cross-Validation:**
   * Employ k-fold cross-validation to assess your SVM's performance on unseen data and help you choose appropriate hyperparameters, like C or kernel parameters.
5. **Early Stopping:**
   * Monitor the SVM's performance on a validation set during training and halt training if the validation performance starts to degrade.
6. **Data Augmentation:**
   * Generate synthetic training examples to add diversity to your training set and help the SVM generalize better.
7. **Reducing Model Complexity:**
   * Control the complexity of the SVM by adjusting the kernel parameters or using a simpler kernel altogether. This can help prevent the model from fitting noise.
8. **Collect More Data:**
   * Increasing your training dataset's size can help the SVM capture more underlying patterns and reduce overfitting.
9. **Regularization Techniques:**
   * Some SVM formulations support explicit regularization techniques. For example, the LinearSVC in scikit-learn allows you to set the 'penalty' parameter, which is used to apply L1 or L2 regularization to the model.
10. **Ensemble Methods:**
    * Consider using ensemble techniques that combine multiple SVMs to mitigate overfitting. Methods like Bagging or Boosting can help improve generalization.
11. **Tuning Hyperparameters:**
    * SVMs have hyperparameters such as C, the kernel parameters, and potentially others depending on the SVM variant. Fine-tune these hyperparameters using techniques like grid search or random search.
12. **Model Selection:**
    * If overfitting remains a significant issue, consider trying different algorithms, such as decision trees, random forests, or gradient boosting, to find a model that handles the data better.

Balancing model complexity, regularization, and hyperparameter tuning is essential when dealing with SVMs to prevent overfitting. Regular validation and testing on new data are crucial to ensure your model generalizes well.

Top of FormTop of Form