Support Vector Machines: A Beginner's Guide with Python and Google Colab

Preface

Welcome to the world of Support Vector Machines (SVMs)! This book is designed to guide you through the fundamentals of SVMs using Python and Google Colab. We'll break down complex concepts into manageable steps, ensuring you grasp the core ideas behind this powerful machine learning algorithm.

1 The Curious Case of the Flickering Camera: An Introduction to Model Evaluation

Imagine you're facing a frustrating problem: your camera keeps flickering. You've been using it just fine for weeks, but suddenly, it starts acting up. What do you do?

Your first instinct might be to update the drivers, hoping that fixes the issue. It works for a while, but the flickering returns. Next, you try using the Zoom application directly, thinking it might bypass some system-level problem. Unfortunately, that fails too. Finally, you switch to a different browser altogether, hoping the issue lies with your browser.

This scenario, while seemingly simple, highlights a fundamental concept in machine learning: **model evaluation**. When one approach fails, we seek a better one. We're constantly evaluating and refining our methods to achieve the best possible results. This is exactly what machine learning is about: identifying a problem, finding solutions using different models, and continually evaluating and improving those models. We evaluate the parameters of the systems in our machine learning model to see if it is getting better with new parameters or not.

Just like you tried different solutions to fix your camera, in machine learning, we try different model parameters and evaluate different metrics, to finally select the one that serves us the best.

2 Understanding Classification and Evaluation

SVMs are powerful tools for **classification**, which is the task of assigning data points to specific categories. Imagine you want to build a system that automatically detects spam emails. Each email would be classified as either "spam" or "not spam."

To evaluate how well our SVM is performing, we need a way to measure its accuracy. Let's consider a simplified example: we have a dataset of emails, and for each email, we know whether it's actually spam or not (the "actual value"). Our SVM makes a

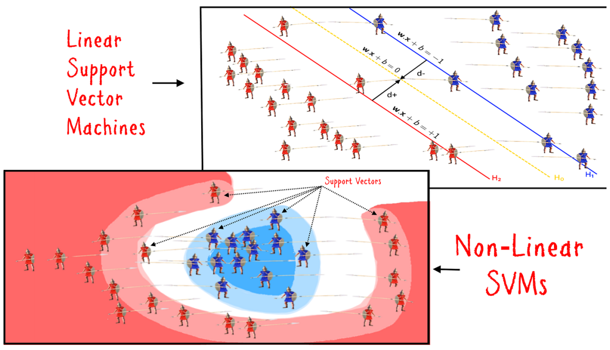

Figure 1: Linear and Non Linear SVMs

prediction for each email (the "predicted value"). We can then compare the actual and predicted values to see how accurate our model is.

Here is an example data sheet:

Actual Value	Predicted Value
0	0
0	1
1	0
1	1

Table 1: Example classification data sheet.

Remember that this is classification, so 0 and 1 might refer to categories "not spam" and "spam."

Now, there are four possible outcomes for each email:

1. **True Positive (TP):** The email is actually spam (actual value is 1), and our SVM correctly predicts that it's spam (predicted value is 1).
2. **True Negative (TN):** The email is actually not spam (actual value is 0), and our SVM correctly predicts that it's not spam (predicted value is 0).
3. **False Positive (FP):** The email is actually not spam (actual value is 0), but our SVM incorrectly predicts that it's spam (predicted value is 1).
4. **False Negative (FN):** The email is actually spam (actual value is 1), but our SVM incorrectly predicts that it's not spam (predicted value is 0).

Of these, TP and TN are "wanted," whereas FP and FN are "unwanted." Knowing the possible outcomes of this prediction, we can calculate "evaluation metrics," which give us a better idea of the effectiveness of our models. Let's discuss some of the important evaluation metrics.

3 The Confusion Matrix: Visualizing Classification Results

To better understand our model's performance, we can organize these outcomes into a **confusion matrix**. The confusion matrix is a table that summarizes the classification results. It has two rows and two columns, representing the actual and predicted classes.

Here's the general structure of a confusion matrix:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Table 2: Structure of a confusion matrix.

4 Key Evaluation Metrics for SVMs

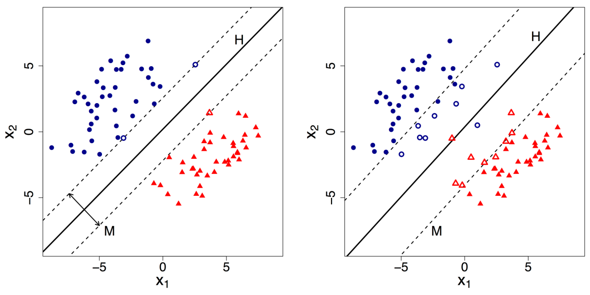

Figure 2: SVMs

Based on the confusion matrix, we can calculate several key evaluation metrics:

1. **Accuracy:** Measures the overall correctness of the model. It's calculated as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

   It essentially represents the ratio of correctly classified instances to the total number of instances.

2. **Precision:** Measures the accuracy of the positive predictions. It's calculated as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

   It answers the question: "Out of all the instances predicted as positive, how many were actually positive?"

3. **Recall:** Measures the ability of the model to find all the positive instances. It's calculated as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

   It answers the question: "Out of all the actual positive instances, how many were correctly identified by the model?"

4. **F1-Score:** A balanced measure that combines precision and recall. It's calculated as:

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

The F1-Score is useful when you want to find a balance between precision and recall, especially when dealing with imbalanced datasets.

5. **Support:** Represents the number of actual occurrences of the class in the specified dataset. For instance, if the dataset has 50 data points labeled 0 and 70 data points labeled 1, the support for 0 is 50 and the support for 1 is 70.

These metrics provide a comprehensive view of our SVM's performance, allowing us to identify areas for improvement.

5 Implementing Evaluation Metrics in Python with Google Colab

Now, let's put our knowledge into practice. Using Google Colab, we can easily calculate these evaluation metrics using Python's scikit-learn library. Let's start with the classification data sheet:

```python
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score,
    precision_score, recall_score, f1_score

# Load the data from the CSV file
df = pd.read_csv("classification_results.csv")

# Separate actual and predicted values
y_true = df["actual"]
y_pred = df["predicted"]

# Calculate the confusion matrix
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", cm)

# Calculate the evaluation metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

Listing 1: Python code for calculating evaluation metrics

In the code above, `classification_results.csv` contains the actual and predicted values of your model. The data will then be converted to evaluation metrics.

6 Case Study: Bank Note Authentication

Let's consider another real-world example: bank note authentication. Imagine you want to build a system that can automatically detect counterfeit bank notes based on image analysis. You have a dataset of bank note images, and for each image, you have extracted features like variance, skewness, kurtosis, and entropy. Your goal is to classify each bank note as either "genuine" or "counterfeit."

To implement this, you can again use Google Colab, in a very similar manner to the previous example. This is an example of the data-banknote-authentication dataset:

variance	skewness	kurtosis	entropy	class
3.6216	8.6661	-2.8073	-0.44699	0
4.5459	8.1674	-2.4586	-1.4621	0
3.866	-2.6383	1.9242	0.10645	0
3.4566	9.5228	-4.0112	-3.5944	0
0.32924	-4.4552	4.5718	-0.9888	0

Table 3: Example bank note authentication dataset.

Here is an example of a model for this dataset:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report,
    confusion_matrix

# Load the data from the text file
df = pd.read_csv("data_banknote_authentication.txt",
    header=None, names=['variance', 'skewness', 'kurtosis',
    'entropy', 'class'])

# Separate features (X) and target (y)
X = df[['variance', 'skewness']]  # Using only variance and
    skewness for simplicity
y = df['class']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create an SVM classifier
classifier = SVC(kernel='rbf', gamma='scale', C=1)
```

```

# Train the classifier
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)

# Print the confusion matrix and classification report
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n",
    classification_report(y_test, y_pred))
```

Listing 2: Python code for bank note authentication with SVM

C and **Gamma** are two very important parameters of Support Vector Machine. You can play around with these parameters and check how your models perform by changing these parameters.

With these lines of code, the process is the same: the model trains on the data, and the accuracy of the model is measured. In a model that can classify if a bank note is counterfeit with high accuracy, you have used the power of SVM!