A Tutorial on Comparative Analysis of Support Vector Machines and Neural Networks for Recommendation Systems

Tutorial by,

KRISHNA VIJAYAN

STUDENT ID:23083552

GITHUB LINK:
https://github.com/Krishnavijayan2301/TutorialonRecommendationsystems/tree/main

INDEX

Introduction:

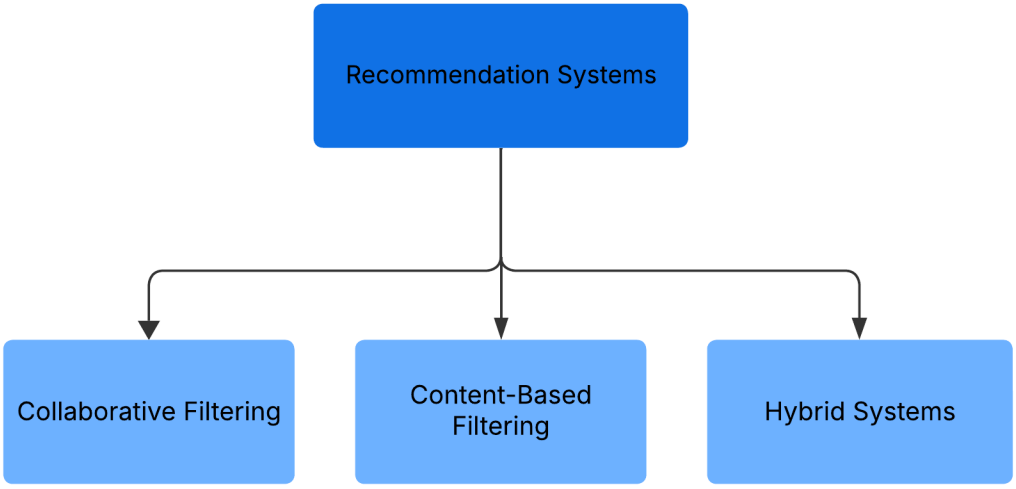
Imagine if the following situation sounds familiar:

Imagine being in your own world, completely immersed in your favorite song. The outside world fades away, and for those few minutes, nothing else matters. And then suddenly, the song changes, and you wake up and look at your screen, only to find it's another one of your favorites. Wonder what happened? Was it the playlist's seamless transition? It doesn't matter and you start smiling as you realize that the music is guiding you, curating the perfect soundtrack for your moment, without you even having to think. You sit back, let the music take over again, and let yourself drift back into this musical world.

It is obviously tailored to your taste, like most of the things that we use today. From Amazon recommending their products to Netflix suggesting movies and Spotify curating personalized playlists, it's all recommendation systems working behind the scenes. These systems analyze vast amounts of data, learning your preferences, behavior, and patterns to offer you content that feels just right. They're designed to enhance your experience. While you're enjoying a song or shopping online, you're not just interacting with a website or app; you're engaging with algorithms that are constantly learning to provide you with the best choices tailored specifically to you. And that exactly my friends, is machine learning, machines learning stuff from the input without necessarily being taught, continuously improving and adapting to your needs without explicit programming. It's a world where technology evolves based on your actions, making the digital world feel more personal every day.

So, What Exactly are Recommendation Systems?

Just like humans learn from data, machines also learn from data. For example, if a person chooses cheesecake over a donut 95 out of 100 times, it's highly likely that they'll choose cheesecake again the 101st time. In much the same way, recommendation systems analyze historical data, understanding patterns and preferences to make predictions. By observing past behaviors, they learn what users are most likely to enjoy and provide tailored suggestions, whether it's a song, movie, or product. Just as we rely on our experiences to make decisions, recommendation systems rely on data to make informed recommendations, improving over time as they gather more insights. There are three main types of recommendation systems:



Collaborative filtering systems are based on user interactions with the product, leveraging data to make recommendations. Content-based filtering, on the other hand, focuses solely on the features of the product itself, such as genre, keywords, or other characteristics. Hybrid systems combine both approaches to offer more personalized recommendations by utilizing both user interaction and product features.

Now, delving deeper into what really matters, this tutorial presents a comparative analysis of two main methods for making recommendations:

Traditional Methods – Here, we will explore how **Support Vector Machines (SVM),** which can be used to classify and predict user preferences based on historical data, focusing on their application in recommendation systems.

Neural Networks – Using advanced techniques such as **TensorFlow** and **Keras**, we'll investigate how neural networks can learn complex patterns from large datasets, offering powerful solutions for making recommendations.

In this tutorial, we will analyse both methods, understand their strengths and limitations, and explore how each can be applied to recommendation systems in real-world scenarios.

Support Vector Machines:

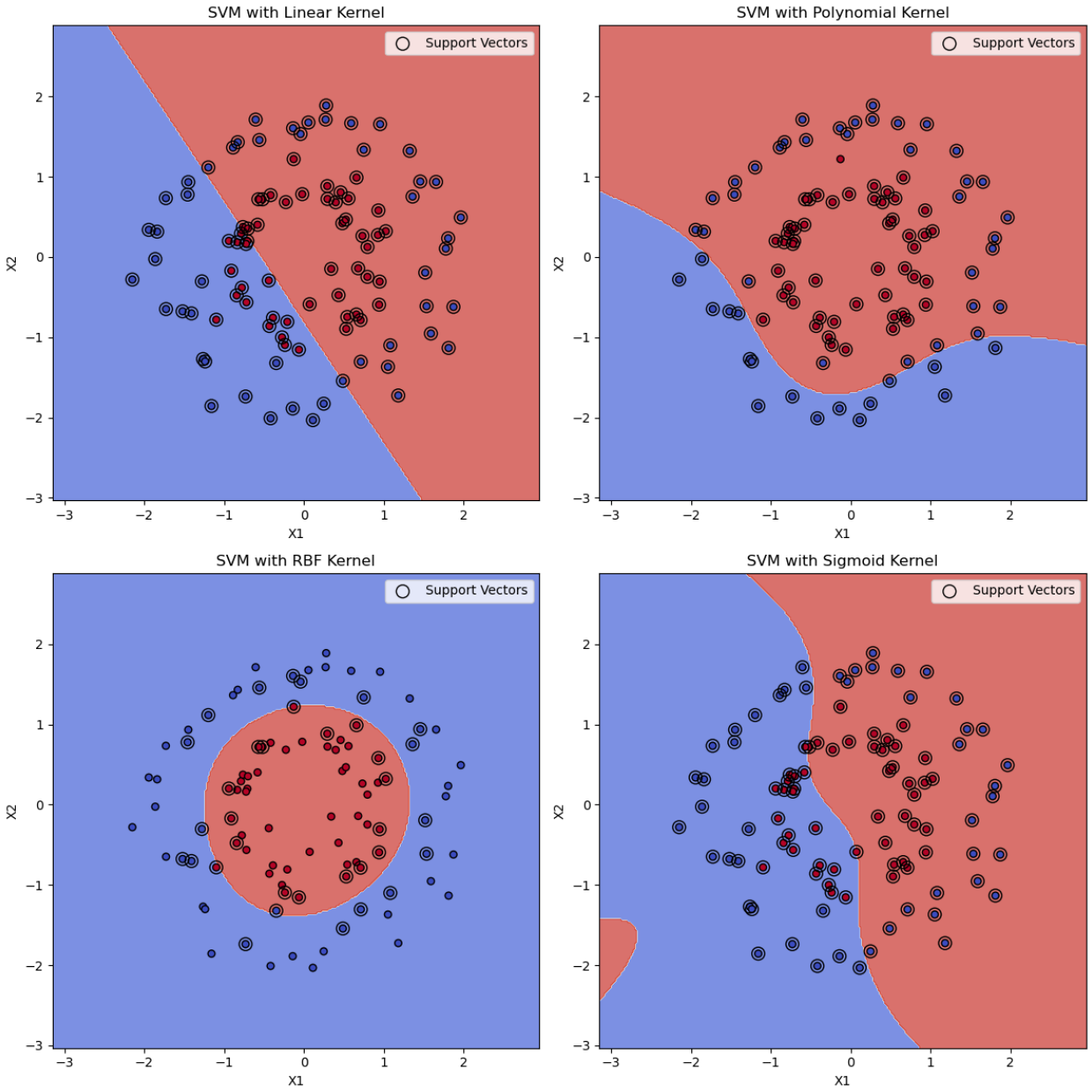
Imagine you want to sort your playlist. Let's say you don't like the songs you added two years ago anymore. So, you decide to divide your songs into two categories: the ones you like and the ones you don't. In real life, this seems simple. However, when you're dealing with large datasets, things get a bit more complex. This is where Support Vector Machines (SVM) come in. SVMs help categorize data by finding a boundary, or "hyperplane," that separates different categories, like the songs you like from the ones you don't.

Now, SVMs use something called **kernels** to help find the best boundary between these categories, and the choice of kernel depends on the data you're working with. Common kernels include:

1. **Linear Kernel**: This is the simplest one and works well when there is a linear relation between the categories.
2. **Polynomial Kernel**: This kernel helps when the data is not linearly separable but can be separated using a polynomial function.
3. **Radial Basis Function (RBF) Kernel**: It’s effective when the data points have non-linear relationships. The RBF kernel transforms the data into a higher-dimensional space where a linear separator may exist.
4. **Sigmoid Kernel**: This kernel uses a sigmoid function.

In recommendation systems, this choice is crucial as user preferences are often non-linear. For instance, applying an RBF kernel in music recommendation may help the model learn subtle patterns such as users liking songs with similar tempos. The choice of kernel depends on your data's characteristics. A kernel, like danceability in songs (as a criterion to depict if you like the song or not), acts as an additional feature to help SVM find the best boundary between categories. Think of kernels as adding a third dimension; if data can't be separated in 2D, applying a kernel lets you view it from a 3D perspective, making the separation

clearer.



A code snippet of practical work on SVM using the Spotify songs dataset is shown below, which recommends songs based on the fact that a person likes it if it has a danceability greater than 0.7. The link to the GitHub repository containing a complete jupyter notebook with interactive coding exercises and

explanation to the code is at the beginning of this pdf along with the dataset.

```
#Import neccessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

#importing dataset
df = pd.read_csv('dataset.csv')
df['user_preference'] = np.where(df['danceability'] > 0.7, 1, 0)

#drop columns with null values
df.isnull().sum()
df = df.dropna()

#feature selection and scaling
features = ['danceability', 'energy', 'loudness', 'tempo']
X = df[features].values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
y = df['user_preference'].values

#seperating to test and training data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

#Train SVM model
svm_model = SVC(kernel='linear', C=1)
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
```

```
#Finding Accuracy
accuracy = accuracy_score(y_test, y_pred)

#Testing the model
new_song = np.array([[0.8, 0.75, -5.0, 120]])
new_song_scaled = scaler.transform(new_song)
prediction = svm_model.predict(new_song_scaled)

if prediction == 1:
    print("Recommend this song!")
else:
    print("Do not recommend this song.")
```

This example shows that the accuracy score is 100 percent. You must be thinking, “OMG! That is Awesome:”, Well, not really, it’s actually a sign of overfitting in most cases. However, in this case, it is due to how easy the task is. The decision rule (danceability > 0.7) is too simple and clear-cut. It would be concerning if the accuracy wasn’t 100 percent, as the rule itself is so straightforward and easy to learn from

the dataset. The task is so simple that anybody should be able to get it right, and it's concerning if a machine doesn't.

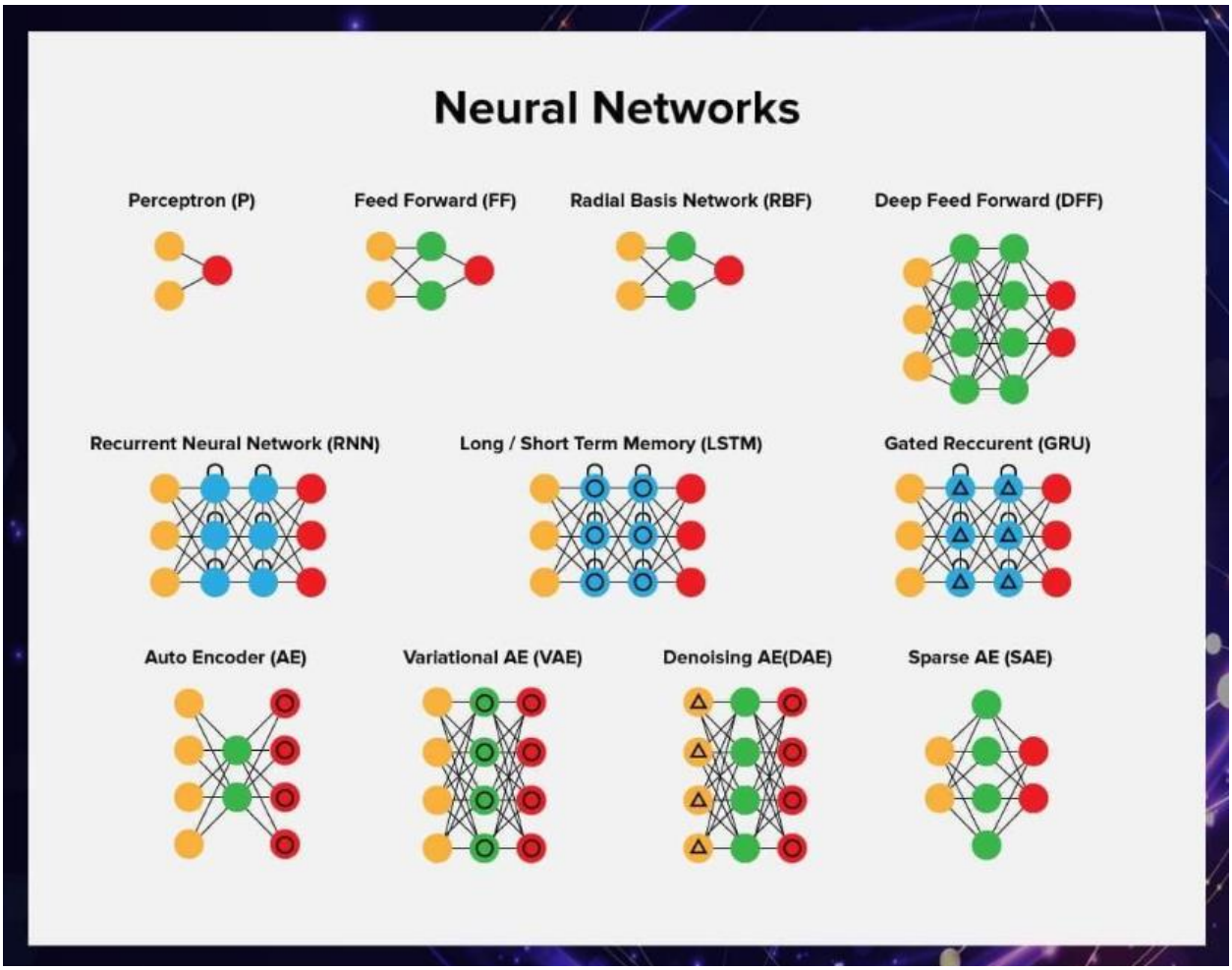
Neural networks:

Just like the human brain uses neural networks to solve problems, machines mimic that process to tackle complex tasks. It's pretty much the machine's way of thinking! In our music recommendation system, just like you vibe to your favourite tunes, we use a cool tool called **TensorFlow**. TensorFlow helps us build models that learn patterns from huge piles of data, kind of like how you get better at choosing songs for Instagram stories over time. So, TensorFlow helps neural network to figure out what tunes you'll love next based on what you've liked before. It's like teaching the machine to have its own "music taste". Thats pretty amazing, right?

A neural network consists of three main components:

1. **Input Layer**: The input layer receives the features of the dataset
2. **Hidden Layers**: These are the intermediate layers where the actual learning takes place. Each hidden layer consists of several neurons.
3. **Output Layer**: The output layer produces the final prediction.

For recommendation systems, neural networks follow a neat formula to figure out what you'll love next. But how about we put this to the test and actually build something? You might be wondering, "What dataset will we use?" Well, we have the Spotify songs dataset! But this time, instead of recommending a song based on your personal taste, we'll build a system that recommends the next song a DJ should play based on their history. It's like training a machine to be the ultimate DJ picking the next track based on the previous ones.



To make this happen, we'll use TensorFlow and Keras. TensorFlow is a powerful open-source library for machine learning, and Keras, which is built on top of TensorFlow, simplifies building neural networks. We also use LSTM, which stands for Long Short-Term Memory, which is a type of recurrent neural network (RNN). It's used for sequential data (like time series or ordered data) because it can capture long-term dependencies in the data. Together, they'll help us create a model that takes the DJ's past song choices and predicts what track will vibe perfectly next. Let's get started!

The code snippet is given below:

```
#install tensorflow
pip install tensorflow

#import neccessary libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split

#import data
df = pd.read_csv('dataset.csv')

#feature extraction
secondset_features = ['danceability', 'energy', 'loudness', 'tempo']
X = df[secondset_features].values


df['next_song_preference'] = df['danceability'].shift(-1) > 0.7
y = df['next_song_preference'].values[:-1]

#feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = X_scaled.reshape((X_scaled.shape[0], 1, X_scaled.shape[1]))

#train test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled[:-1], y, test_size=0.2, random_state=20)

#model creation
model = Sequential([
    LSTM(64, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(32, activation='relu'),
    Dense(1, activation='linear')
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

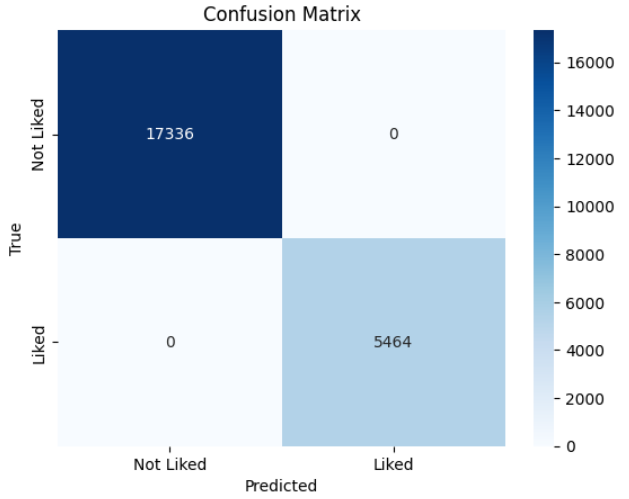
```python
#model fit
history = model.fit(X_train, y_train, epochs=50, batch_size=62, validation_data=(X_test, y_test))
sample_song = X_test[0].reshape(1, 1, 4)
predicted_preference = model.predict(sample_song)[0][0]

#output
if predicted_preference > 0.5:
    print("The DJ might play a song with similar features.")
else:
    print("The DJ might not play a song with similar features.")

#prediction
sample_danceability = sample_song[0][0][0]

lower_bound = df['danceability'].quantile(0.25)
upper_bound = df['danceability'].quantile(0.75)

similar_songs = df[(df['danceability'] > lower_bound) & (df['danceability'] < upper_bound)]


if len(similar_songs) > 0:
    recommended_song_name = similar_songs['track_name'].values[0]
    print(f"For example, the DJ might play: {recommended_song_name}")
else:
    print("No songs with similar danceability found in the dataset.")

#find accuracy and loss
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

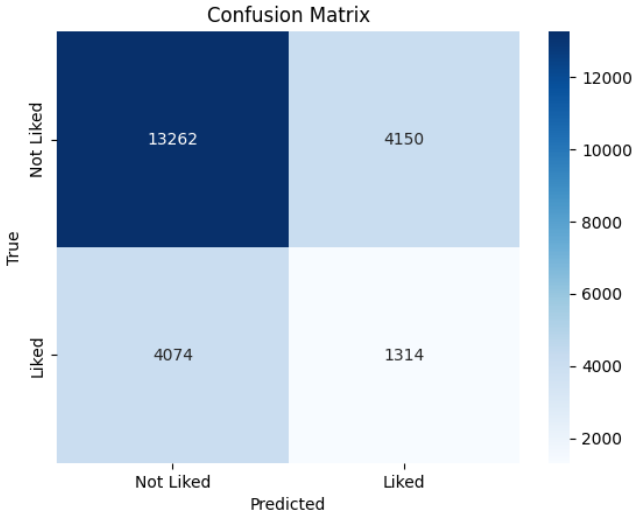
Comparison: SVM VS NN in Recommendation systems

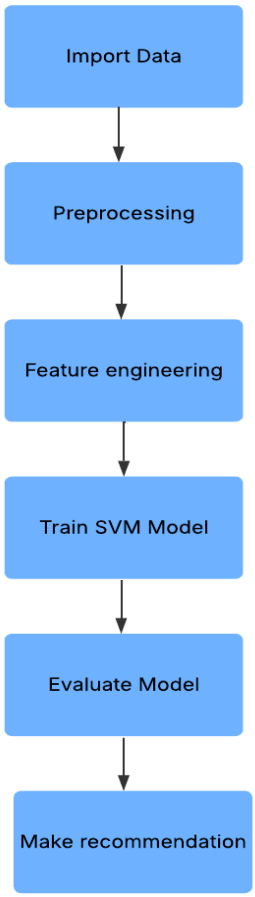
Working:

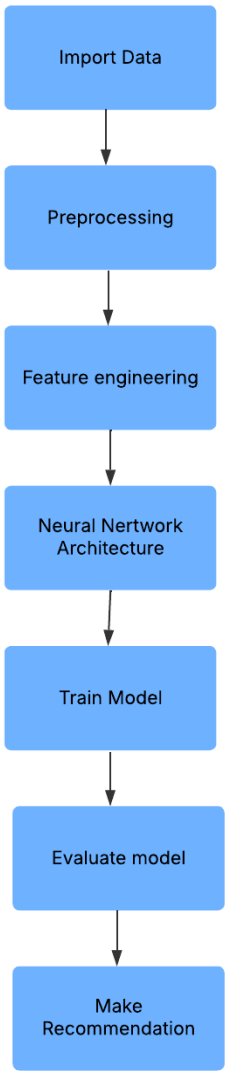
Confusion Matrix of both is given first one is of SVM, and the second one is of neural networks. Following that, a flow chart showcasing the infrastructure followed in the code is also given for comparison.

The important thing is, the results are completely dependent on dataset and the problem we are trying to solve, you should never think that this is always the case or this is how recommendation systems work in SVMs and NNs.


Confusion Matrix
Not Liked
Liked
True
17336
0
0
5464
Not Liked
Liked
Predicted
16000
14000
12000
10000
8000
6000
4000
2000
0


Confusion Matrix
Not Liked
Liked
True
13262
4150
4074
1314
Not Liked
Liked
Predicted
12000
10000
8000
6000
4000
2000





1. Performance and Accuracy:

Support Vector Machines:

- These are traditional machine learning techniques mainly used for classification and regression, as shown in the example, we used it to classify whether a user might like the song or not using past preferences.
- These Perform well with smaller datasets and when the problem you are trying to solve is not very complex.

Neural Networks:

- These are more complex and can deal very well with non- linear relationships, like in our example when we used this to predict the next song the Dj is going to play.
- These can perfrom very well with large datasets and find intricate patterns and trends that are often hard for svm model to learn.

2. Model complexity

- SVM is relatively simple compared to neural networks. It requires less tuning and fewer parameters. However, it becomes computationally expensive for large datasets
- Neural networks can handle more complex relationships but require more tuning, such as selecting the appropriate number of layers, neurons, and activation functions. Training deep networks also requires more computational resources, such as powerful GPUs, and more time.

3. Training time

- SVM models usually train faster on smaller datasets but may experience a slow down as the dataset size increases.
- Neural networks tend to require longer training times, especially when using large datasets or complex architectures. However, with modern GPU acceleration, this issue is somewhat mitigated.

4.Use cases: When to use which one?

Support vector machines:

- ❖ When datasets are Smaller with Simpler patterns
- ❖ Tasks that don't require heavy computational sources

Neural Networks:

- ❖ Large Datasets with non-linear patterns
- ❖ Tasks when computational sources are readily available

Conclusion:

In this tutorial, we explored two powerful machine learning techniques, Support Vector Machines and Neural Networks, and their application in building recommendation systems. We discussed their strengths, weaknesses and demonstrated how they can be implemented in Python using Spotify music data.

We also discussed the importance of evaluation metrics like precision, recall, and F1-score in assessing the effectiveness of the recommendation system(which is in jupyter notebook).

Understanding these metrics helps ensure that the model is not only accurate but also effective in providing meaningful recommendations to users.

Note: Don't forget to take a look at the GitHub repository and the jupyter notebook that has been created in it for you, feel free to use the code and experiment with various parameters and datasets. By experimenting with these models and exploring other machine learning approaches, you can continue to enhance your ability to build recommendation systems and apply these skills to real-world problems. So, go Ahead and keep experimenting, who knows? Maybe you can create your own personalised DJ!

References:

1. https://www.restack.io/p/recommendation-systems-answer-svm-cat-ai
2. https://developer.nvidia.com/blog/using-neural-networks-for-your-recommender-system/
3. https://github.com/GiacomoLeoneMaria/Neural-Network-for-Recommendation-Systems?tab=readme-ov-file
4. https://towardsdatascience.com/recommender-systems-a-complete-guide-to-machine-learning-models-96d3f94ea748/
5. https://www.researchgate.net/publication/225139953_Recommender_Systems_Using_Support_Vector_Machines
6. https://scikit-learn.org/stable/modules/svm.html
7. https://www.geeksforgeeks.org/support-vector-machine-algorithm/
8. https://medium.com/data-science-in-your-pocket/recommendation-systems-using-neural-collaborative-filtering-ncf-explained-with-codes-21a97e48a2f7
9. https://medium.com/sciforce/deep-learning-based-recommender-systems-b61a5ddd5456
10. https://serokell.io/blog/deep-learning-and-neural-network-guide#how-do-you-train-an-algorithm%3F
11. https://www.tensorflow.org/
12. https://marutitech.medium.com/what-are-the-types-of-recommendation-systems-3487cbafa7c9
13. https://scikit-learn.org/stable/modules/svm.html
14. https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/

The flow charts are built using lucid app, and the code on creating other graphs has been included in the jupyter notebook.