5. Support Vector Machine (SVM)

October 18, 2024

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```python
plt.rcParams['figure.figsize'] = [19, 8]
```

[3]:
```python
import warnings
warnings.filterwarnings('ignore')
```

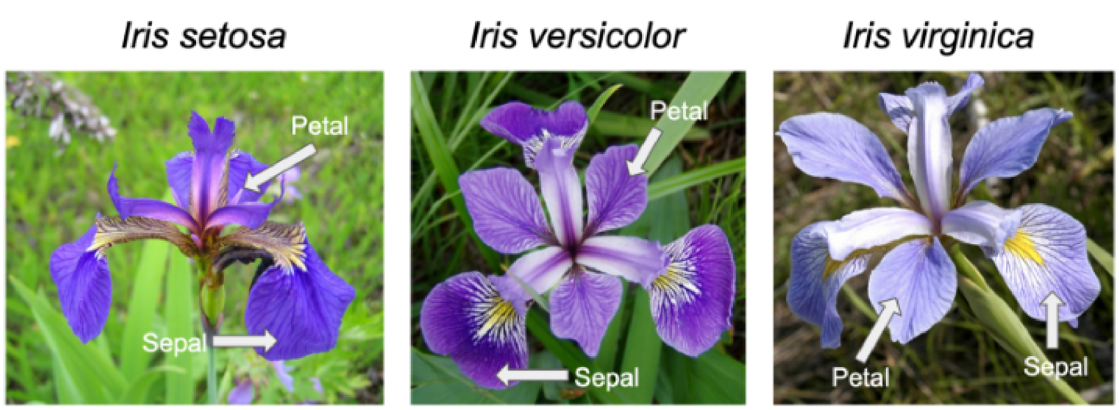
1 Introduction

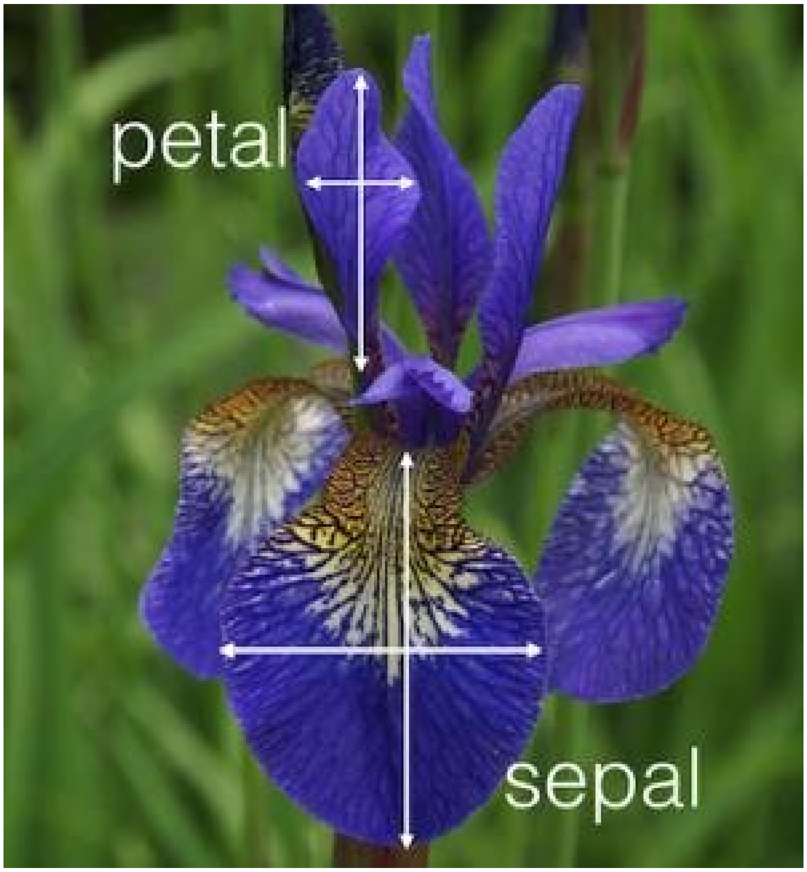
Support Vector Machine is a very popular classification algorithm.

Support Vector Machine (SVM) is a Machine Learning model that works on the principle of hyper plane.

Support Vector Machine draws a hyper plane and divides the data points into different classes.

2 Case Study - Iris Flower Species Identification





2.1 Data

[4]:
```
from sklearn.datasets import load_iris
```

[5]:
```
iris = load_iris()
```

[6]:
```
dir(iris)
```

[6]:
```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

[7]:
```
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

iris_df.head()
```

[7]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

[8]:
```
iris_df['target'] = iris.target
```

[9]:
```
iris_df.head()
```

[9]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

[10]:
```
iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    int32
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

[11]:
```
iris.target_names
```

[11]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

[12]:
```
iris_df.duplicated().sum()
```

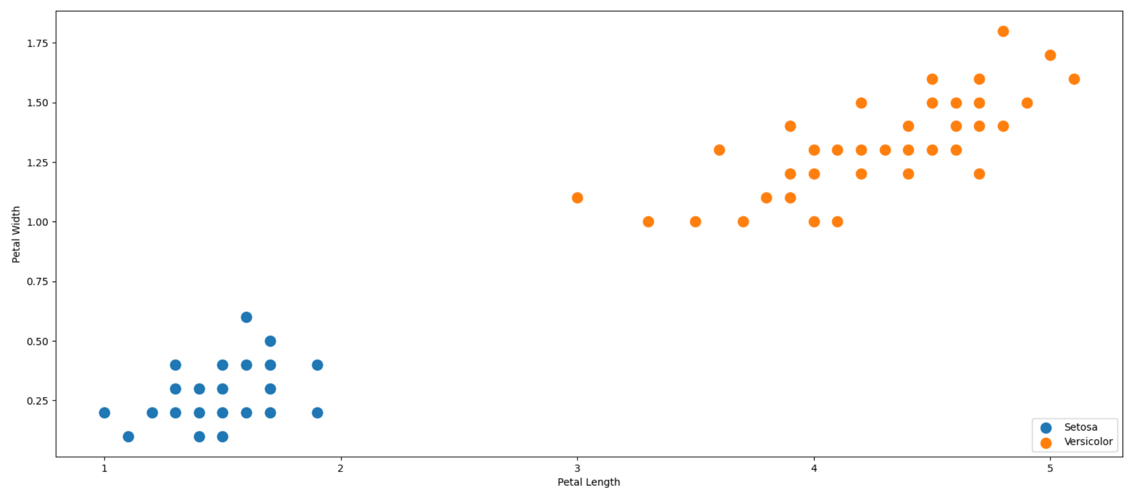
[12]:
```
1
```

```
[13]: iris_df.drop_duplicates(inplace=True)
```

2.2 Data Visualization

```
[14]: iris_setosa = iris_df.loc[iris_df['target'] == 0, :]

iris_versicolor = iris_df.loc[iris_df['target'] == 1, :]

iris_virginica = iris_df.loc[iris_df['target'] == 2, :]
```

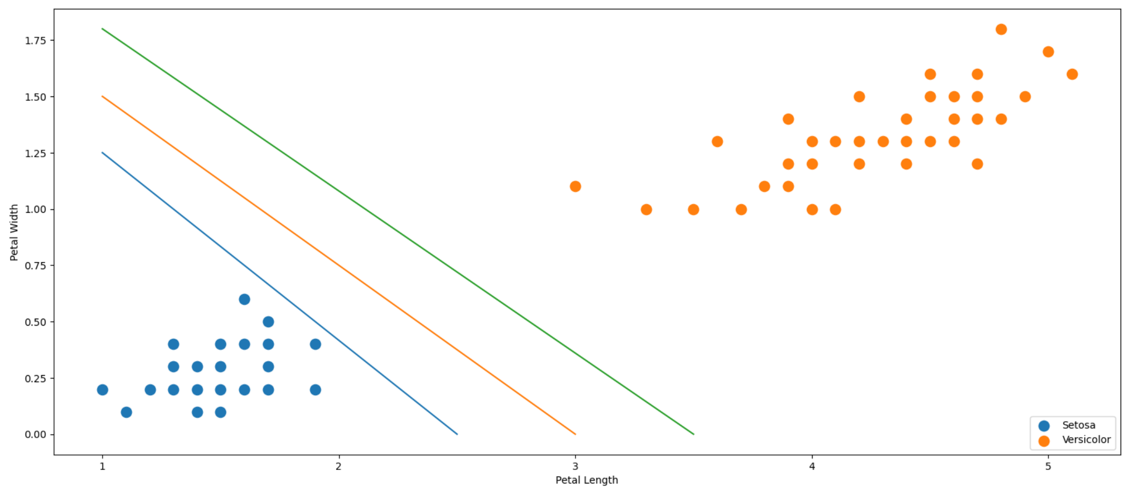
```
[15]: sns.scatterplot(data=iris_setosa, x='petal length (cm)', y='petal width (cm)',␣
↪s=150)

sns.scatterplot(data=iris_versicolor, x='petal length (cm)', y='petal width␣
↪(cm)', s=150)

plt.legend(['Setosa', 'Versicolor'], loc='lower right')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.show()
```



There are many possible ways to draw a classification boundary to separate the 2 groups.

```
[16]: sns.scatterplot(data=iris_setosa, x='petal length (cm)', y='petal width (cm)',␣
↪s=150)
```

```
sns.scatterplot(data=iris_versicolor, x='petal length (cm)', y='petal width␣
↪(cm)', s=150)

sns.lineplot(x=[1, 2.5], y=[1.25, 0])

sns.lineplot(x=[1, 3], y=[1.5, 0])

sns.lineplot(x=[1, 3.5], y=[1.8, 0])

plt.legend(['Setosa', 'Versicolor'], loc='lower right')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.show()
```



How to decide the best boundary for the classification? Consider the line near to the data points or far from the data points?

The distance of the data point from the boundary line is called as the margin.

Which line better? One with the lower margin or one with the higher margin?

The line with the higher margin is better as it can classify the 2 groups in a better way.

Support Vector Machine tries to maximize the margin.

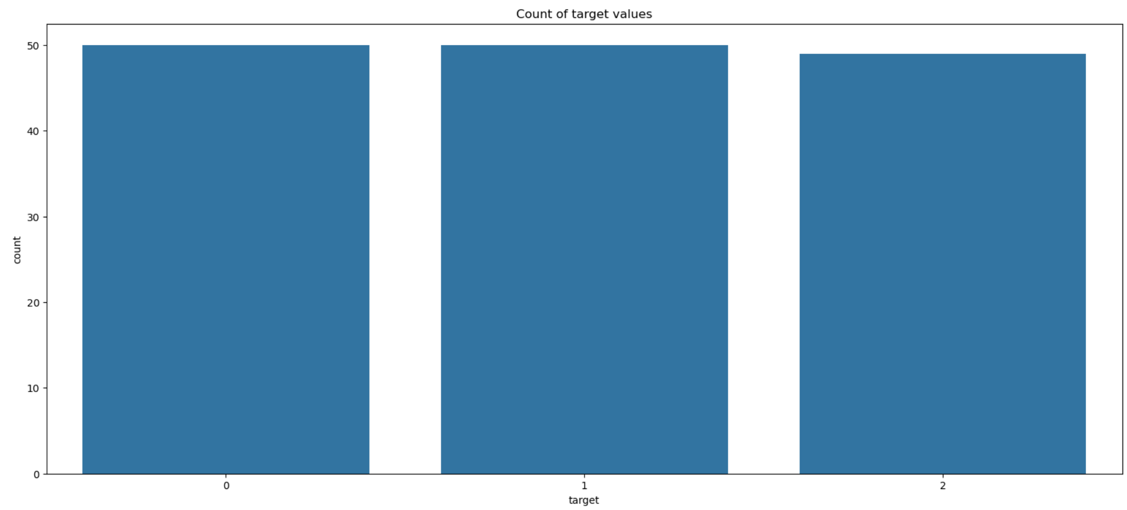
The data points which are closer to the line are called the Support Vectors and hence the name Support Vector Machine.

In case of a 2D space (2 attributes), the classification boundary is a line.

In case of a 3D space (3 attributes), the classification boundary is a plane.

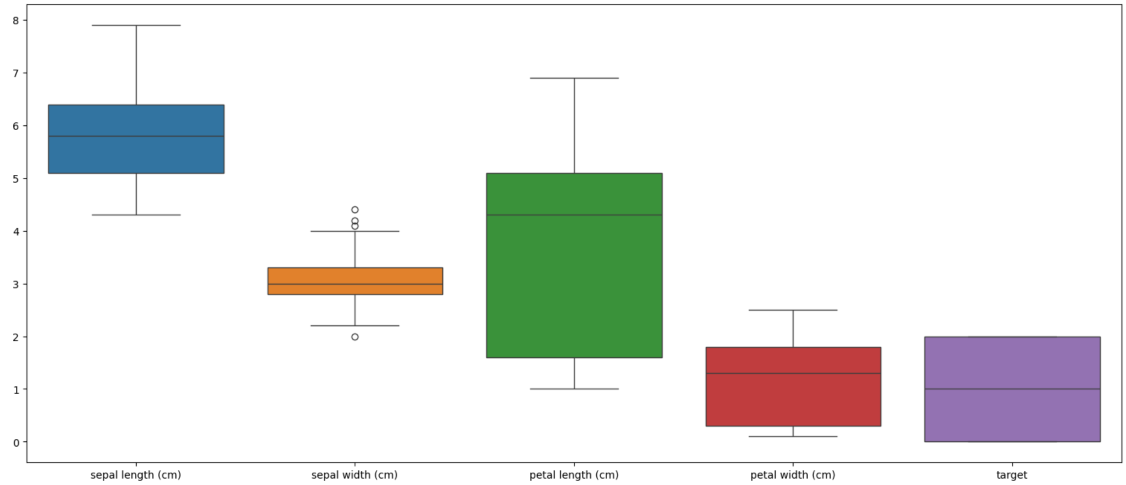
In case of a nD space (n attributes), the classification boundary is a hyper plane.

[17]:
```
sns.countplot(data=iris_df, x='target')
plt.title("Count of target values")
plt.show()
```



2.3 Outlier

[18]:
```
sns.boxplot(iris_df)

plt.show()
```



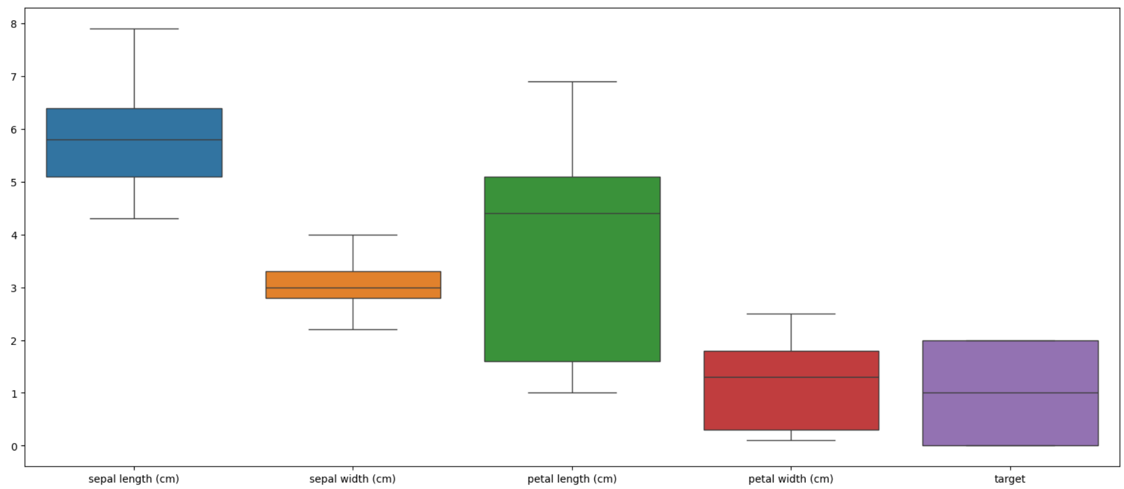
[19]:
```
# obtain the first quartile
Q1 = iris_df.quantile(0.25)

# obtain the third quartile
Q3 = iris_df.quantile(0.75)

# obtain the IQR
IQR = Q3 - Q1

# print the IQR
print(IQR)
```

```
sepal length (cm)    1.3
sepal width (cm)     0.5
petal length (cm)    3.5
petal width (cm)     1.5
target               2.0
dtype: float64
```

[20]:
```
ul = Q3 + 1.5 * IQR

ll = Q1 - 1.5 * IQR
```

[21]:
```
iris_df = iris_df[~((iris_df < ll) |(iris_df > ul)).any(axis=1)]
```

[22]:
```
sns.boxplot(iris_df)

plt.show()
```



2.4 Divide the data frame into independent and dependent variables

[23]:
```python
X = iris_df.loc[:, :'petal width (cm)'].values

y = iris_df.loc[:, 'target'].values
```

[24]:
```python
y
```

[24]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

2.5 Data Normalization

[25]:
```python
from sklearn.preprocessing import StandardScaler
```

[26]:
```python
scaler = StandardScaler()
```

[27]:
```python
X = scaler.fit_transform(X)
```

2.6 Split the Data into train and test data

[28]:
```python
from sklearn.model_selection import train_test_split
```

[29]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=1)
```

2.7 Model Training

[30]:
```python
from sklearn.svm import SVC # Support Vector Classifier
```

2.7.1 Parameters in Support Vector Machine

Gamma parameter Gamma value decides whether the data points close to the hyperplane should be considered to classify a data point or not.

When *gamma* is low, points far from the hyperplane are also included in decision making.

When *gamma* is high, points close to the hyperplane are included.

In the sklearn library, *gamma* can be *auto* or *scale*.

The default value of *gamma* is *scale*.

gamma is calculated using the following formula:

When *gamma* is *auto*, $gamma = 1/\text{no. of independent variables}$.

Whne *gamma* is *scale gamma* = 1/(no. of independent variables * X.var())

Regularization Regularization helps to solve over fitting problem in machine learning.

Over fitting refers to low bias and high variance.

To reduce the varaince, a penalty term is added to the formula used by the model.

In sklearn, regularization is indiacted by C.

When C value is low, SVM uses a larger margin between the data points. The data points closer to the hyper plane may not be classified correctly.

When C value is high, the margin will be small and maximum data points can be classified correctly.

The default value of C is 1.0

Kernel function Sometimes, it may not be possible to divide the data points using a straight line. In such case, we may need to use curved line.

A kernel function is a mathematical formula used to rearrange the data points such that they can be divided clearly by the SVM.

Following are the different types of kernel functions that can be used by the SVM:

- Linear kernel function
- Polynomial kernel function
- Radial Basis function
- Sigmoid kernel function

[31]:
```
model = SVC()
```

[32]:
```
model.fit(X_train, y_train)
```

[32]:
```
SVC()
```

[33]:
```
model.score(X_train, y_train)
```

[33]:
```
0.9741379310344828
```

2.8 Model Evaluation

[34]:
```
model.score(X_test, y_test)
```

[34]:
```
0.9310344827586207
```

2.9 Model Prediction

[35]:
```
y_predict = model.predict(X_test)
```

[36]:
```
y_predict
```

[36]: array([0, 2, 0, 2, 1, 0, 0, 2, 0, 1, 1, 1, 1, 2, 0, 2, 0, 0, 0, 1, 2, 1,
       0, 0, 2, 2, 2, 2, 1])

[37]:
```
y_test
```

[37]: array([0, 1, 0, 2, 1, 0, 0, 2, 0, 1, 1, 1, 1, 1, 0, 2, 0, 0, 0, 1, 2, 1,
       0, 0, 2, 2, 2, 2, 1])

2.10 Confusion Matrix

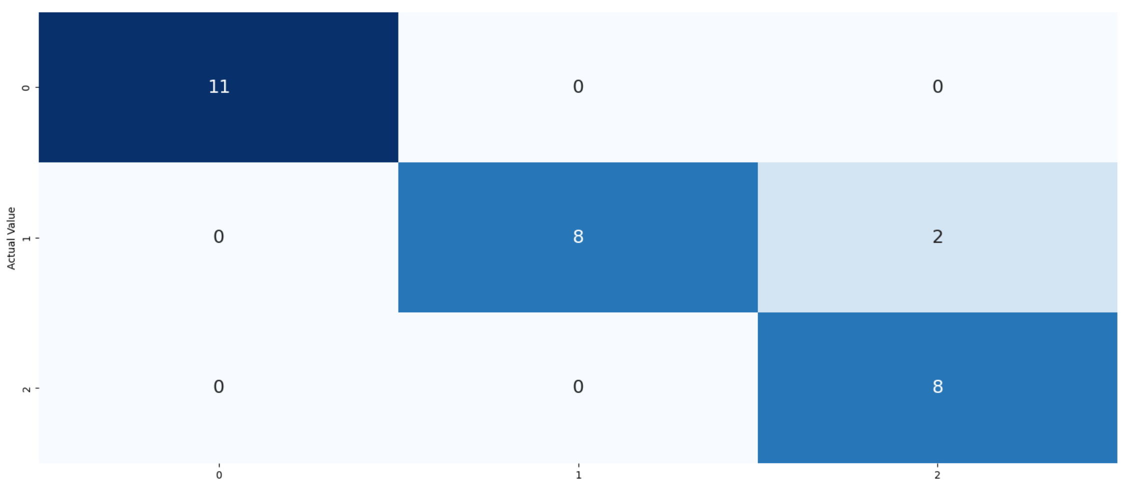
[38]:
```
from sklearn.metrics import confusion_matrix
```

[39]:
```
cm = confusion_matrix(y_test, y_predict)
```

[40]:
```
cm
```

[40]:
```
array([[11,  0,  0],
       [ 0,  8,  2],
       [ 0,  0,  8]], dtype=int64)
```

[41]:
```
sns.heatmap(cm, annot=True, cmap='Blues', cbar=False, annot_kws={"fontsize":18})

plt.xlabel("Predicted Value")

plt.ylabel("Actual Value")

plt.show()
```



[ ]: