Softmax Regression (Multiclass Classification)

`multi_class=' multinomial ', solver=' lbfgs '`

Purpose: A machine learning algorithm used for **multi-class classification**

How Does Softmax Regression Work?

- **Input**: Features (e.g., pixel values of an image, attributes of a fruit).
- **Output**: Probabilities for each class (e.g., `[0.1, 0.7, 0.2]` for 3 classes).
- **Key Idea**: Uses the **softmax function** to convert raw scores (logits) into probabilities.

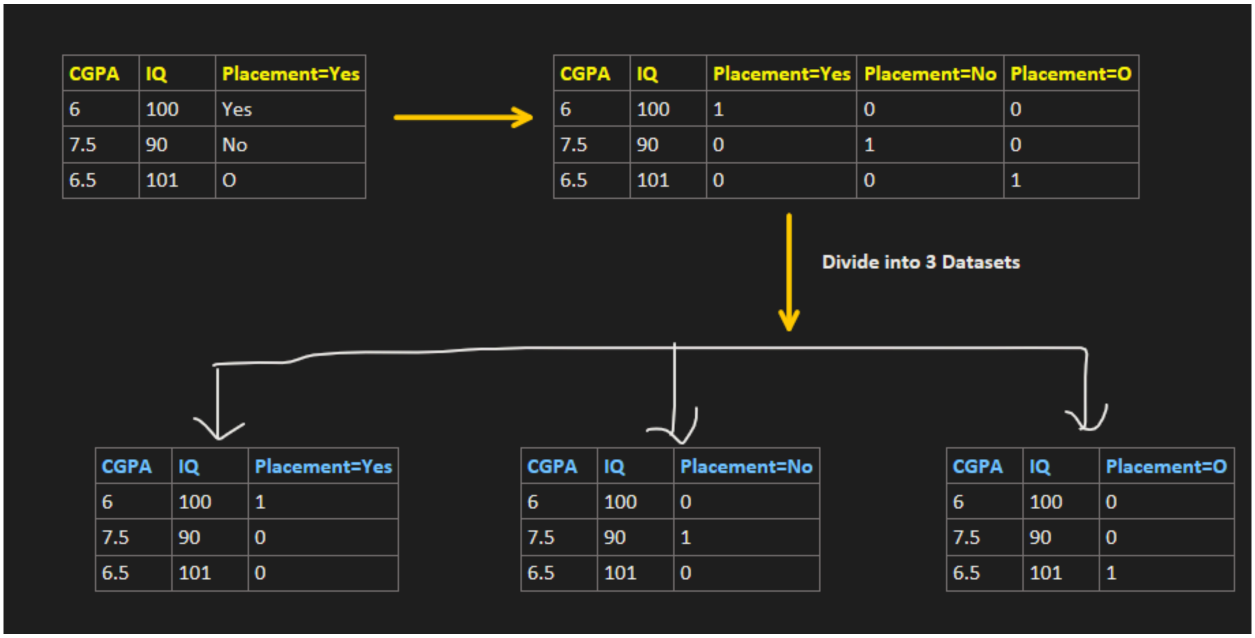
OVR(One-vs-Rest) Approach

`OneVsRestClassifier(log_reg)`

`LogisticRegression(multi_class='ovr')`

- A binary classifier is trained to distinguish between the **target class (positive class)** and all the **other classes (negative class).**
- In the **OvR** approach, the multi-class classification problem is decomposed into multiple binary classification problems.
- For a multi-class classification problem with K classes, we train K separate binary classifiers.

Apply One Hot Encoding & then divide it into 3 datasets



CGPA	IQ	Placement=Yes
6	100	Yes
7.5	90	No
6.5	101	O

CGPA	IQ	Placement=Yes	Placement=No	Placement=O
6	100	1	0	0
7.5	90	0	1	0
6.5	101	0	0	1

Divide into 3 Datasets

CGPA	IQ	Placement=Yes
6	100	1
7.5	90	0
6.5	101	0

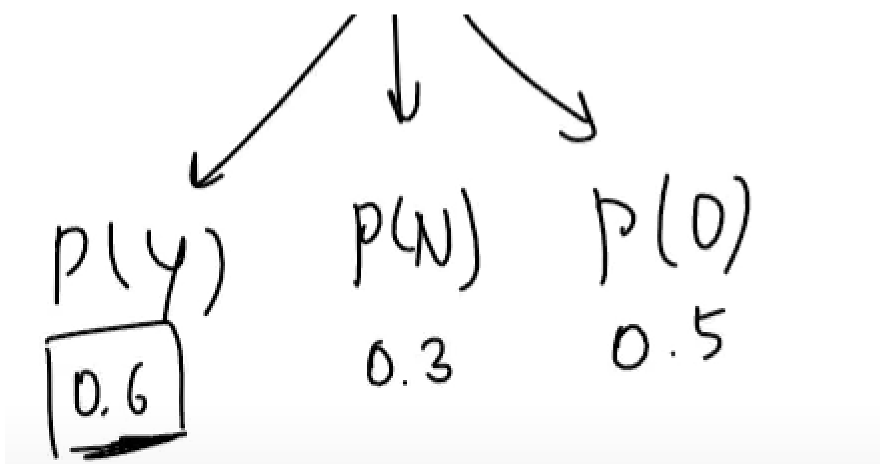
CGPA	IQ	Placement=No
6	100	0
7.5	90	1
6.5	101	0

CGPA	IQ	Placement=O
6	100	0
7.5	90	0
6.5	101	1

- This 👆 became a binary classification.
- **Apply normal Logistic Regression on this.**
- Run **Logistic Regression independently**

During prediction:

- Each classifier gives a score (probability or decision value).
- The class with the highest score is predicted as the output.



- In above example, you choose Y

Disadvantage:

- If the no. of categories and n are more, it will take a lot of time.
- Not efficient with large datasets

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

# Create the Logistic Regression model
log_reg = LogisticRegression(max_iter=10000)

# Wrap the model with One-vs-Rest approach
ovr_model = OneVsRestClassifier(log_reg)

# Train the model
ovr_model.fit(X_train, y_train)
```

```
# Predict on the test set
y_pred = ovr_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=data.target_names, yticklabels=data.target_names)
plt.title('Confusion Matrix (One-vs-Rest)')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Alternative:

```
LogisticRegression(multi_class='ovr')
```

Predict probability

```
# prediction
query = np.array([[3.4,2.7]])
clf.predict_proba(query)
```

```
array([[0.44387139, 0.55512309, 0.00100552]])
```

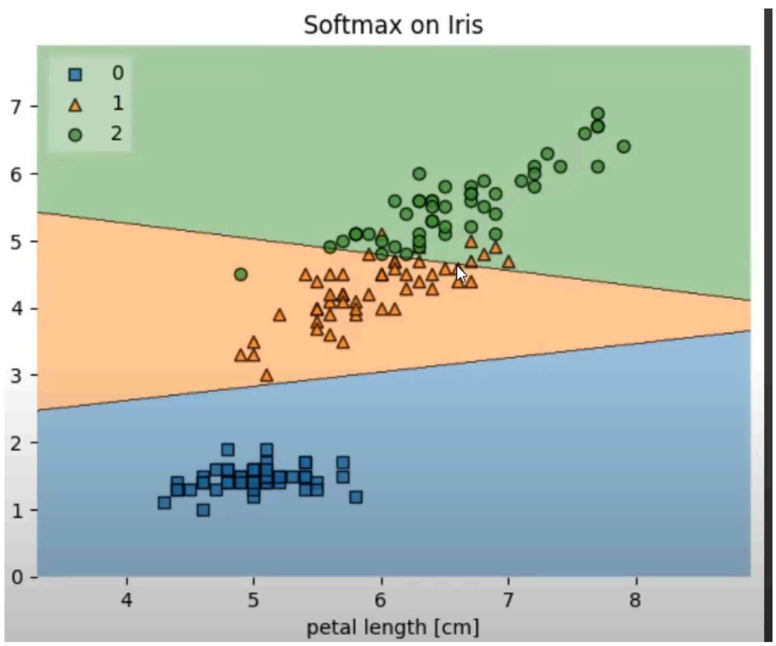
- We provide 3 input values of sepal length, petal width, etc and it gave us prob of all 3 classes.

```
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(X.values, y.values, clf, legend=2)

# Adding axes annotations
plt.xlabel('sepal length [cm]')
plt.xlabel('petal length [cm]')
plt.title('Softmax on Iris')

plt.show()
```



Multinomial/ Softmax Approach:

> 💡 **VIMP for Interview**

SoftMax Function

$$P(y = i \mid X) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- z_k: Raw score (logit) for class k.
- K: Total number of classes.
- e: Euler's number (~2.718).

$\underline{z_1} \quad \underline{z_2} \quad \underline{z_3}$

$$\sigma(z_1) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

1

$$\sigma(z_1) + \sigma(z_2) + \sigma(z_3) = 1$$

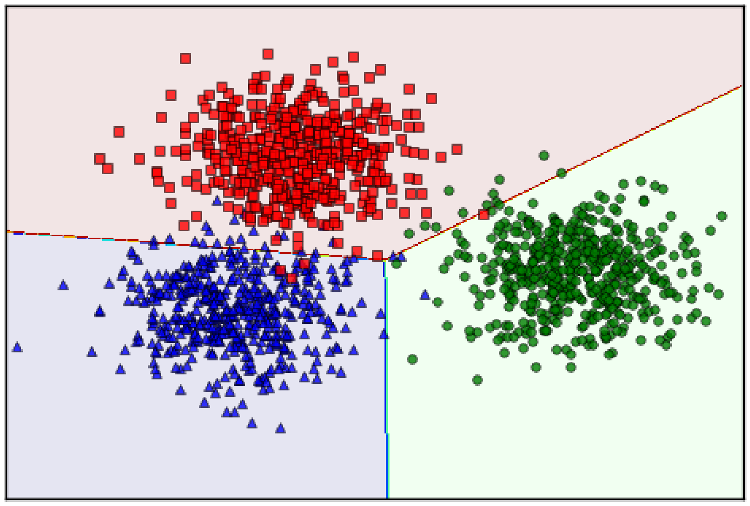
$$\sigma(z_2) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$\sigma(z_3) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

What it does?

- Converts raw scores into probabilities that sum to `1`.
- The class with the highest probability is the predicted class.

Steps in Softmax Regression



1. **Linear Combination**:
   - For each class, compute a raw score (z_k) using features (X), weights (w_k), and bias (b_k):

$$z_k = w_{k1}x_1 + w_{k2}x_2 + \cdots + w_{kn}x_n + b_k$$

2. **Apply Softmax Function:**
   - Convert raw scores into probabilities:

$$P(y = k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

3. **Make a Prediction**:

- The class with the highest probability is the predicted class.

- **Apply One Hot Encoding on training just like OVR**

Loss Function for SoftMax:

$$\text{Loss} = -\sum_{i=1}^{m}\sum_{j=1}^{K} y_{i,j} \log(P(y_j \mid \mathbf{x}_i))$$

Where:

- $y_{i,j}$ is an indicator (0 or 1) representing whether the i-th sample belongs to class j.
- $P(y_j \mid \mathbf{x}_i)$ is the predicted probability of class j for the i-th sample.

- K = No. of classes

Training the Model

- **Goal**: Find the best weights (w) and biases (b) to minimize prediction errors.
- **How**:
  - Use a cost function (e.g., **cross-entropy loss**) to measure errors.
  - Adjust weights and biases using optimization techniques (e.g., **gradient descent**).

Python Implementation

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the Iris dataset
data = load_iris()
X = data.data  # Features (4 features per sample)
y = data.target  # Target (3 classes)

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

# Train the Logistic Regression model with multi-class (Softmax regression) o
ption
clf = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=10
000)
clf.fit(X_train, y_train)

# Predict the classes on the test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Visualize the confusion matrix
fig, ax = plt.subplots(figsize=(6, 6))
ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
ax.set_title('Confusion Matrix')
ax.set_xlabel('Predicted Label')
```

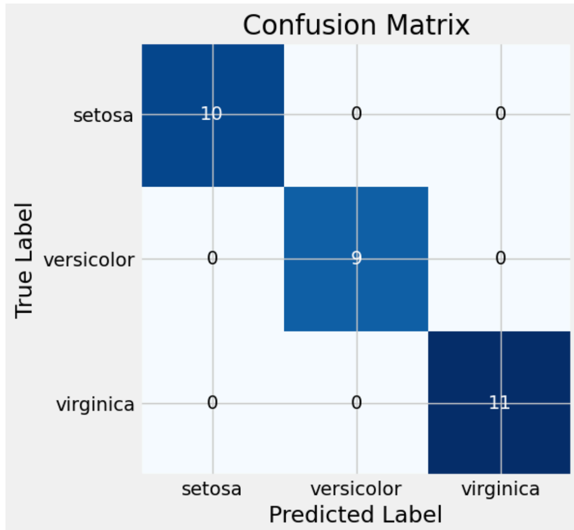
```
ax.set_ylabel('True Label')

# Add labels to the matrix
classes = data.target_names
tick_marks = np.arange(len(classes))
ax.set_xticks(tick_marks)
ax.set_yticks(tick_marks)
ax.set_xticklabels(classes)
ax.set_yticklabels(classes)

# Add text inside the squares
thresh = cm.max() / 2.
for i, j in np.ndindex(cm.shape):
    ax.text(j, i, format(cm[i, j], 'd'),
            ha="center", va="center",
            color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.show()
```

```
Accuracy: 1.00
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```



When to use what?

Use One-vs-Rest (OVR) when:

1. **Classes are Non-Mutually Exclusive:** OVR is appropriate if an instance can belong to more than one class, as each classifier provides an independent probability for each class.
2. **Dealing with Imbalanced Data:** OVR might perform better when class distribution is highly imbalanced since each class gets a dedicated model.

Use Multinomial Logistic Regression (SoftMax Regression) when:

1. **Computational Efficiency is Required:** Softmax Regression is generally more efficient for large datasets and a high number of classes.
2. **Classes are Mutually Exclusive:** SoftMax Regression is a good choice when each instance can only belong to one class.
   - The SoftMax function provides a set of probabilities that sum to 1, fitting well with mutually exclusive classes.
   - For example, if you're classifying animals as "dog," "cat," or "bird," an animal can't be both a dog and a cat at the same time.

3. **Interpretability is Important:** The probabilities output by SoftMax Regression are more interpretable than those from OVR, as they always sum to 1. This can make model predictions easier to explain.