Multiclass classification with Logistic Regression

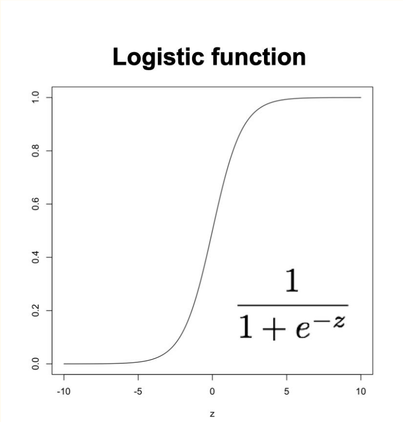
Using Two Strategies:

1. **One vs All(OvA)**
2. **All-pairs(OvO)**

Presented By: Allison Gao, Jingxian Zhang

The math behind ML algorithms

Machine learning algorithms aim to minimize a loss function to improve predictions



Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

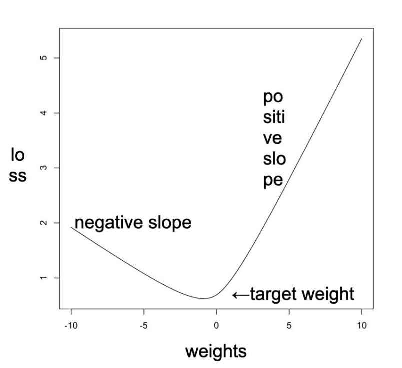
Logistic Loss

$$L(y_i, \hat{y}_i) = -\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right)$$

Stochastic Gradient Descent

$$\frac{\partial L}{\partial w} = (\hat{y}_i - y_i)x_i$$

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$



$$|L_{t+1} - L_t| < \epsilon$$

How does it work?

One-vs-All (OvA)
Pseudo code

Initialize parameters $\mathbf{w}$ for each class, learning rate α, and batch size b
converge $=$ False

while not converge:
  epoch$+=1$
  Shuffle training examples

  for $i = 0, 1, \ldots, \left\lceil \frac{n_{\text{examples}}}{b} \right\rceil - 1$: (iterate over batches)
    $X_{\text{batch}} = X[i \cdot b : (i+1) \cdot b]$ (select the X in the current batch)
    $\mathbf{y}_{\text{batch}} = \mathbf{y}[i \cdot b : (i+1) \cdot b]$ (select the labels in the current batch)
    $\nabla L_{\mathbf{w}} = \mathbf{0}$ (initialize gradient matrix for each class)

    for each pair of training data $(x, y) \in (X_{\text{batch}}, \mathbf{y}_{\text{batch}})$:
      for $j = 0, 1, \ldots, n_{\text{classes}} - 1$:
        if $y = j$:
          $\nabla L_{\mathbf{w}_j} += \left(\sigma(\mathbf{w}_j^T x) - 1\right) \cdot x$ (for correct class, reflects how
        else:
          $\nabla L_{\mathbf{w}_j} += \sigma(\mathbf{w}_j^T x) \cdot x$ (for other classes)
      $\mathbf{w}_j = \mathbf{w}_j - \alpha \cdot \frac{\nabla L_{\mathbf{w}_j}}{\text{len}(X_{\text{batch}})}$ (update weights for each class)

  Calculate this epoch loss
  if $|\text{Loss}(X, \mathbf{y})_{\text{this-epoch}} - \text{Loss}(X, \mathbf{y})_{\text{last-epoch}}| < \text{CONV-THRESHOLD}$:
    converge $=$ True (break the loop if loss converged)

How does it work?

All-pairs(OvO)
Pseudo code

Initialize parameters $\mathbf{w}$ for each pair of classes, learning rate α, and batch size b

converge $=$ False

while not converge:

epoch$+=1$

Shuffle training examples

for $i = 0, 1, \ldots, \left\lceil \frac{n_{\text{examples}}}{b} \right\rceil - 1$: (iterate over batches)

$X_{\text{batch}} = X[i \cdot b : (i+1) \cdot b]$ (select the X in the current batch)

$\mathbf{y}_{\text{batch}} = \mathbf{y}[i \cdot b : (i+1) \cdot b]$ (select the labels in the current batch)

for each unique pair of classes (A, B):

$\nabla L_{\mathbf{w}_{AB}} = \mathbf{0}$ (initialize gradient for each pair (A, B))

for each $(x, y) \in (X_{\text{batch}}, \mathbf{y}_{\text{batch}})$:

if $y = A$ or $y = B$: (focus on examples for classes A and B)

if $y = A$:

$\nabla L_{\mathbf{w}_{AB}} += \left(\sigma(\mathbf{w}_{AB}^T x) - 1\right) \cdot x$ (for class A)

else:

$\nabla L_{\mathbf{w}_{AB}} += \sigma(\mathbf{w}_{AB}^T x) \cdot x$ (for class B)

$\mathbf{w}_{AB} = \mathbf{w}_{AB} - \alpha \cdot \frac{\nabla L_{\mathbf{w}_{AB}}}{\text{len}(X_{\text{batch}})}$ (update weights for the pair (A, B))

Calculate this epoch loss

if $|\text{Loss}(X, \mathbf{y})_{\text{this-epoch}} - \text{Loss}(X, \mathbf{y})_{\text{last-epoch}}| < \text{CONV-THRESHOLD}$:

converge $=$ True (break the loop if loss converged)

Read file data

```
df = pd.read_csv(file_path)
```

Check null or nan values

```
if df.isnull().sum().sum() > 0:
    print("There are missing values in the dataset.")
else:
    print("No missing values in the dataset.")
```

balanced distribution using random undersampling

```
undersample = RandomUnderSampler(random_state=42)
```

numerical conversion

```
y_over.replace(list(np.unique(y_over)), [1, 2, 3, 4, 5, 6, 7], inplace=True)
df_dea = X_over
df_dea['Class'] = y_over
```

splitting train and test data

```
X_train, X_test, y_train, y_test = train_test_split(X_over, y_over, random_state=0, shuffle=True, test_size=.2)
```

Scale data

```
st_x = StandardScaler()
X_train = st_x.fit_transform(X_train)
X_test = st_x.transform(X_test)
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()
```

fit data to our model

```
X_train, Y_train, X_test, Y_test = get_data(DATA_FILE)
num_features = X_train.shape[1]
NUM_CLASS = 7
BATCH_SIZE = 100
CONV_THRESHOLD = 1e-3

X_train_b = np.hstack((X_train, np.ones((X_train.shape[0], 1))))
X_test_b = np.hstack((X_test, np.ones((X_test.shape[0], 1))))

model = MulticlassLogisticRegression(num_features, NUM_CLASS, BATCH_SIZE, CONV_THRESHOLD)
model.train(X_train_b, Y_train)
acc = model.accuracy(X_test_b, Y_test)
print("One-vs-all model accuracy: ",acc)

logistic_regression_model = LogisticRegression(solver='liblinear')
ova_model = OneVsRestClassifier(logistic_regression_model)
ova_model.fit(X_train, Y_train)
print("Library model accuracy: ",ova_model.score(X_test,Y_test))
```

Interesting Things about Multiclass Classification with Logistic Regression(OvA & OvO)

- Modular and Versatile
  - **Decomposes multiclass problems** into simpler binary classification tasks, making the approach modular and interpretable.
  - **OvA** is computationally efficient and suitable for datasets with many classes.
  - **OvO** works well for datasets where inter-class boundaries are complex and requires fine-grained pairwise comparisons.
- Simple but Effective
  - Easy to implement using logistic regression
  - Can adapt to **imbalanced class distributions**, ensuring fair consideration for underrepresented classes.
  - Can **control overfitting** using regularization during weight optimization.

Challenges during Implementation

- Computational Intensity
  - **OvA** requires k classifiers (one per class).
  - **OvO** requires $k(k-1)/2$ classifiers (one for every class pair).
  - **OvO's quadratic growth** in classifiers makes it computationally expensive for datasets with a high number of classes.
- Complex Parameter Tuning
  - Learning rate (α)
  - Batch size
  - convergence threshold
- Interpretability and Aggregation
  - OvA: Decision boundaries for individual classifiers may **overlap.**
  - OvO: **Conflicting votes** from pairwise classifiers can lead to ambiguous final predictions.