Logistička i softmaks regresija

Samostalna primena i primena u neuronskim mrežama

Student: Danilo Veljović, 1120 Profesor: Dr. Suzana Stojković

Univerzitet u Nišu, Elektronski fakultet

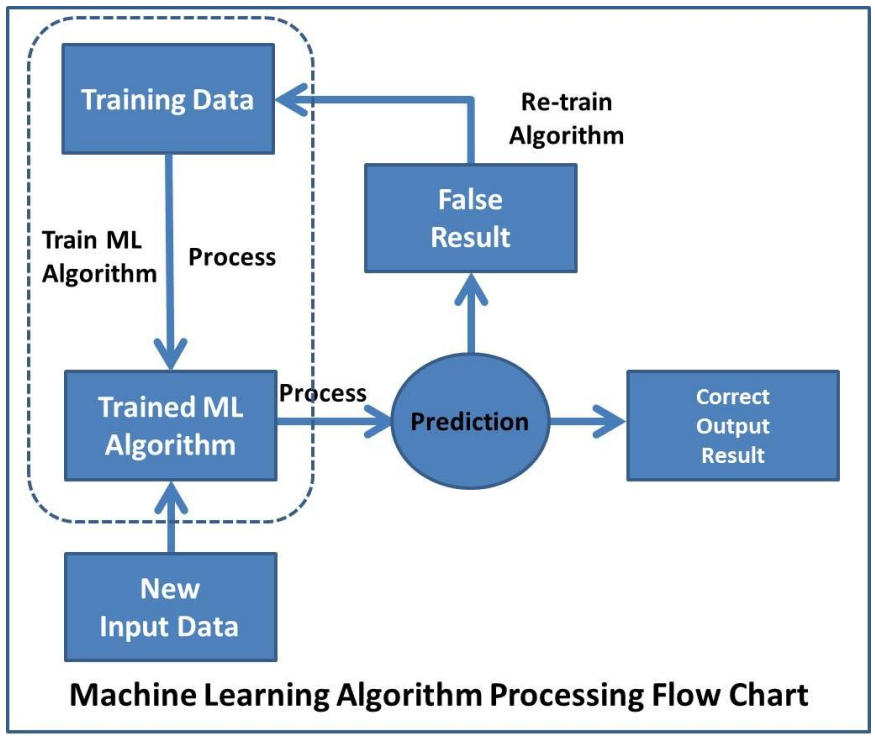
Februar 2021

0 Sadržaj

1 Outline

1 Uvod

- Šta je model mašinskog učenja
- Koje su tehnike mašinskog učenja
- Logistička i softmaks regresija



Slika: Slika 1. Uprošćeni šematski prikaz procesa mašinskog učenja

2 Outline

Elementi modela logističke regresije:

- Logistička funkcija -

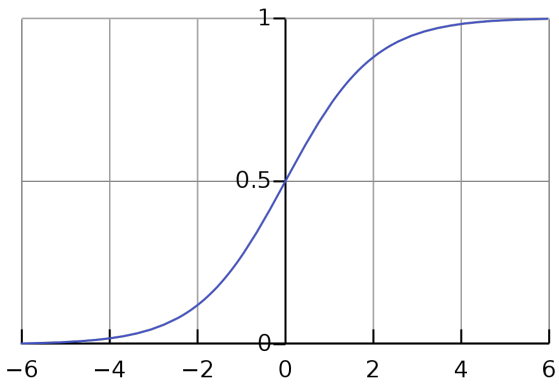
$$\sigma(\alpha) = \frac{e^{\alpha}}{1 + e^{\alpha}} = \frac{1}{1 + e^{-\alpha}} \tag{1}$$

- Funkcija kazne -

$$c(\theta) = [\ y ln(\hat{p}) + (1 - y) ln(1 - \hat{p})] \tag{2}$$

- Treniranje modela - algoritam opadajućeg gradijenta
- Funkcija predikcije -

$$\hat{y} = \begin{cases} 1, & \hat{p} > 0.5 \\ 0, & \hat{p} < 0.5 \end{cases} \tag{3}$$



Elementi modela softmaks regresije:

- Softmaks funkcija -

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{e^{s_k(x)}}{\sum_{j=1}^{K} s_j(x)} \tag{4}$$

- Funkcija kazne -

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} ln(\hat{p}_k^{(i)}) \tag{5}$$

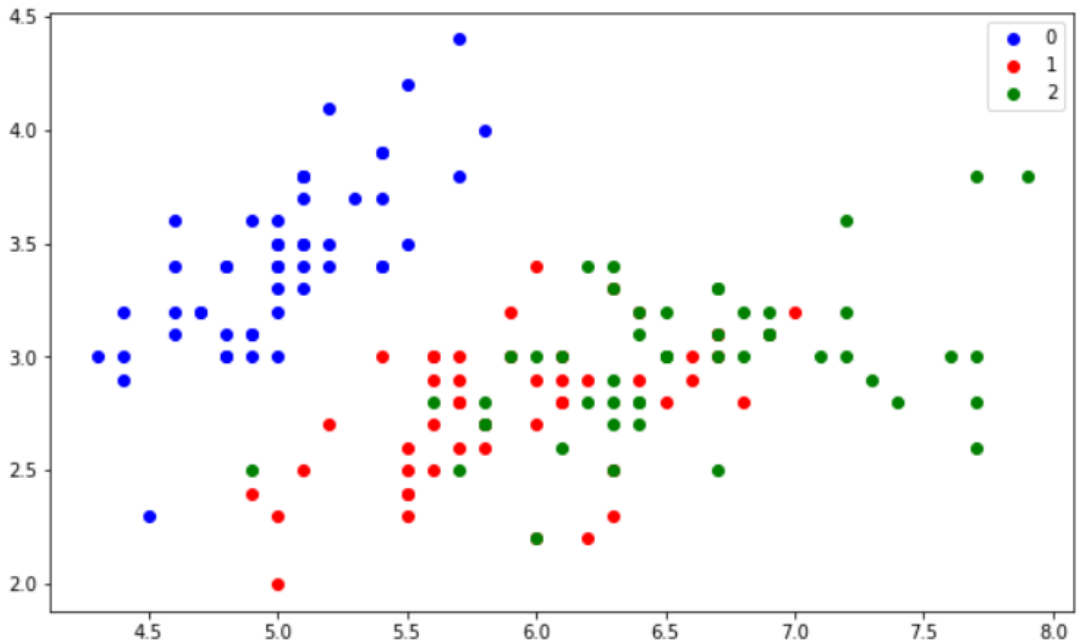
- Treniranje modela - algoritam opadajućeg gradijenta
- Funkcija predikcije - argument koji daje maksimalnu vrednost softmaks funkcije

3 Outline

3 Implementacija modela logističke regresije

In [119]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='b', label='0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='r', label='1')
plt.scatter(X[y == 2][:, 0], X[y == 2][:, 1], color='g', label='2')
plt.legend();
```



Slika: Linearno separabilne klase/vrste perunike

```python
class LogisticRegression:
    def __init__(self, lr = 0.01, num_iter = 10000, fit_intercept = True, verbose = False):
        self.lr = lr
        self.num_inter = num_iter
        self.fit_intercept = fit_intercept
        self.verbose = verbose
```

Slika: Konstruktor modela logističke regresije

```python
    def __sigmoid(self, z):
        return 1/(1 + np.exp(-z))

    def __loss(self, h, y):
        return(-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

Slika: Logistička funkcija i funkcija gubitaka

```
def fit(self, X, y):
    if(self.fit_intercept):
        X = self.__add_intercept(X)

    self.theta = np.zeros(X.shape[1])

    for i in range(self.num_inter):
        z = np.dot(X, self.theta)
        h = self.__sigmoid(z)
        gradient = np.dot(X.T, (h - y)) / y.size
        self.theta -= self.lr * gradient

        if(self.verbose == True and i % 10000 == 0):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            print(f'loss: {self.__loss(h, y)} \t')
```

Slika: Funckija za treniranje modela

```
def predict_prob(self, X):
    if self.fit_intercept:
        X = self.__add_intercept(X)

    return self.__sigmoid(np.dot(X, self.theta))

def predict(self, X):
    return self.predict_prob(X).round()
```

Slika: Funkcija za predikciju

```python
class SoftmaxRegression:
    def __init__(self, K, lr=0.01, num_iter=10000):
        self.lr = lr
        self.num_iter = num_iter
        self.K = K
```

Slika: Konstruktor modela softmaks regresije

```python
    def __softmax(self, z):
        z -= np.max(z)
        return np.exp(z) / np.sum(np.exp(z))

    def __h(self, X, y):
        return self.__softmax(X @ self.theta)

    def __J(self, preds, y, m):
        return np.sum(- np.log(preds[np.arange(m), y]))

    def __T(self, y, K):
        # one hot encoding
        one_hot = np.zeros((len(y), K))
        one_hot[np.arange(len(y)), y] = 1
        return one_hot

    def __compute_gradient(self, theta, X, y, m):
        preds = self.__h(X, theta)
        gradient = 1 / m * X.T @ (preds - self.__T(y, self.K))
        return gradient
```

Slika: Softmaks funkcija i funkcija gubitaka

3 Implementacija modela softmaks regresije


```
def fit(self, X, y):
    hist = {'loss': [], 'acc': []}
    m, n = X.shape
    np.random.seed(0)
    self.theta = np.random.random((n, self.K))
    for i in range(self.num_iter):
        gradient = self.__compute_gradient(self.theta, X, y, m)
        self.theta -= self.lr * gradient

        # loss
        preds = self.__h(X, self.theta)
        loss = self.__J(preds, y, m)

        c = 0
        for j in range(len(y)):
            if np.argmax(self.__h(X[j], self.theta)) == y[j]:
                c += 1
            acc = c / len(y)
            hist['acc'].append(acc)
        # print stats
        if i % 5000 == 0:
            print('{:.2f} {:.2f}%'.format(loss, acc * 100))
```

Slika: Funckija za treniranje modela

```
def predict_prob(self, X):
        return self.__softmax(np.dot(X, self.theta))

def predict(self, X):
        return np.argmax(self.predict_prob(X), axis=1)
```

Slika: Funkcija za predikciju

```
# build the neural network
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation=keras.activations.relu))
model.add(keras.layers.Dense(100, activation=keras.activations.relu))
model.add(keras.layers.Dense(1, activation=keras.activations.sigmoid))
```

Slika: Isečak koda za kreiranje modela

```
model.compile(loss=keras.losses.sparse_categorical_crossentropy,
              optimizer=keras.optimizers.SGD(),
              metrics=[keras.metrics.sparse_categorical_accuracy]
              )

history = model.fit(X_train, y_train, epochs=30, validation_data=(X_valid, y_valid))
```

Slika: Isečak koda za treniranje

```
X_new = X_test[:3]
y_proba = model.predict(X_new) > 0.5
print(y_proba)
print(y_test[:3])
```

```
# build the neural network
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation=keras.activations.relu))
model.add(keras.layers.Dense(100, activation=keras.activations.relu))
model.add(keras.layers.Dense(10, activation=keras.activations.softmax))
```

Slika: Isečak koda za kreiranje modela

```
model.compile(loss=keras.losses.sparse_categorical_crossentropy,
              optimizer=keras.optimizers.SGD(),
              metrics=[keras.metrics.sparse_categorical_accuracy]
              )

history = model.fit(X_train, y_train, epochs=30, validation_data=(X_valid, y_valid))
```

Slika: Isečak koda za treniranje

```
X_new = X_test[:3]
y_proba = model.predict(X_new)
print(y_proba.round(2))
print(y_test[:3])
```

4 Outline

4 Rezultati primene logističke regresije

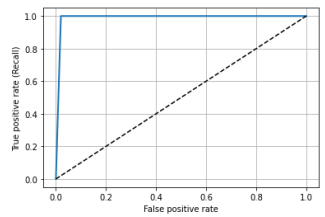

```
In [36]: from sklearn.metrics import confusion_matrix
         confusion_matrix(ret, rety)

Out[36]: array([[49,  0],
                [ 1, 50]], dtype=int64)
```

Slika: Matrica konfuzije

```
In [44]: def plot_roc_curve(fpr, tpr, label=None):
             plt.plot(fpr, tpr, linewidth=2, label=label)
             plt.plot([0,1], [0,1], 'k--')
             plt.axis("on")
             plt.grid()
             plt.xlabel("False positive rate")
             plt.ylabel("True positive rate (Recall)")

         plot_roc_curve(fpr, tpr)
         plt.show()
```



Slika: ROC kriva

```
In [46]: from sklearn.metrics import roc_auc_score
         roc_auc_score(rety, ret)

Out[46]: 0.99
```

Slika: Površina ispod krive

4 Rezultati primene logističke regresije

```
In [119]: from sklearn.metrics import precision_score, recall_score

          precision_score(rety, ret)
Out[119]: 0.9803921568627451

In [120]: recall_score(rety, ret)
Out[120]: 1.0

In [121]: from sklearn.metrics import f1_score
          f1_score(rety, ret)
Out[121]: 0.99009900990099
```

Slika: Metrike

```
In [47]: preds = model.predict(X_test)
         (preds == y_test).mean()
Out[47]: 1.0
```

Slika: Testiranje modela nad test podacima

4 Rezultati primene softmaks regresije


```
In [142]: from sklearn.metrics import confusion_matrix
          confusion_matrix(ret, rety)

Out[142]: array([[50,  0,  0],
                 [ 0, 28,  2],
                 [ 0, 22, 38]], dtype=int64)
```

```
In [157]: from sklearn.metrics import precision_recall_fscore_support as score

          precision, recall, fscore, support = score(rety, ret)

          print('precision: {}'.format(precision))
          print('recall: {}'.format(recall))
          print('fscore: {}'.format(fscore))
          print('support: {}'.format(support))

          precision: [1.         0.93333333 0.63333333]
          recall: [1.   0.56 0.95]
          fscore: [1.   0.7  0.76]
          support: [50 50 40]
```

Slika: Matrica konfuzije i metrike

```
In [146]: y_pred = model.predict(X_test)
          print(y_pred)
          print(y_test)

          [2 2 2 2 2 2 2 2 2 2]
          [2 2 2 2 2 2 2 2 2 2]
```

Slika: Testiranje modela nad test podacima

4 Rezultati primene logističke regresije u NN


```
[[47192  4449]
 [ 2301  1058]]
```

Slika: Matrica konfuzije

```
precision: [0.91384752 0.31497469]
recall: [0.95350858 0.19211912]
fscore: [0.93325687 0.23866456]
```

Slika: Metrike

```
[[0.8780212]
 [0.       ]
 [0.       ]]
[9 2 1]
```

Slika: Rezultati primene nad test podacima

4 Rezultati primene softmaks regresije u NN


```
[[5217   12  103  144   19    0  849    0   15    0]
 [   5 5379    1   35    6    0    6    0    3    0]
 [  42    2 4580   32  289    1  316    0    5    0]
 [  58   41   32 5004   91    0   78    0    8    0]
 [  14    8  582  222 4995    0  471    0   21    0]
 [   1    0    0    0    0 5437    0   50    6   16]
 [ 192    1  187   59  106    0 3763    0   18    0]
 [   0    0    0    0    0   61    0 5329    4  169]
 [  14    1   11    2    6    5   24    6 5429    3]
 [   0    0    0    1    0    3    0  103    1 5306]]
```

Slika: Matrica konfuzije

```
precision: [0.82041201 0.98969641 0.86956522 0.94201807 0.79122446 0.98675136
 0.86985668 0.95793637 0.98691147 0.98005172]
recall: [0.94118708 0.98806025 0.83333333 0.90998363 0.90620464 0.98728891
 0.68331215 0.9710277  0.98529946 0.96578085]
fscore: [0.87665938 0.98887765 0.85106383 0.9257238  0.8448203  0.98702006
 0.76538188 0.96443761 0.9861048  0.97286395]
```

Slika: Metrike

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
[9 2 1]
```

Slika: Rezultati primene nad test podacima