Pacmann

Logistic Regression with Regularization

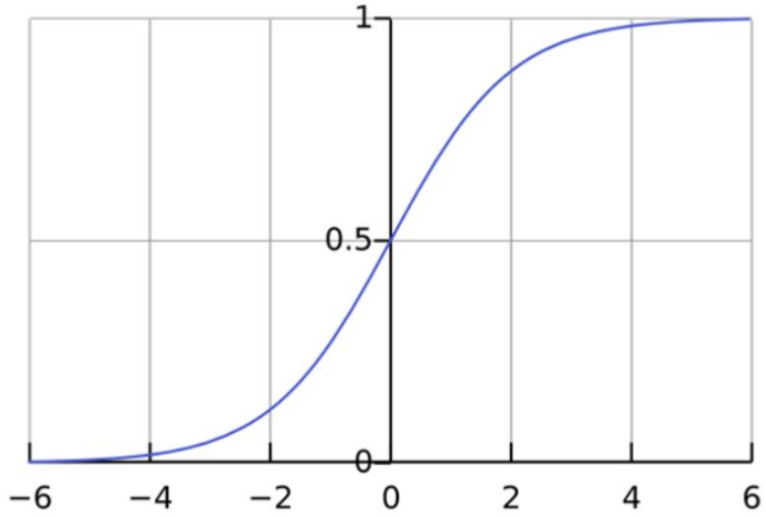
Advanced Machine Learning

Outline

- Introduction/Background
- Python Code
- Make conclusion and recommendation
- References

Introduction

Introduction

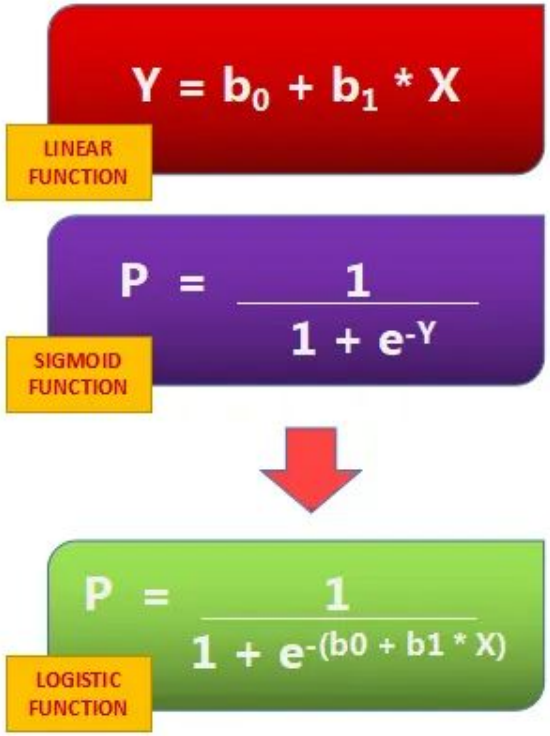


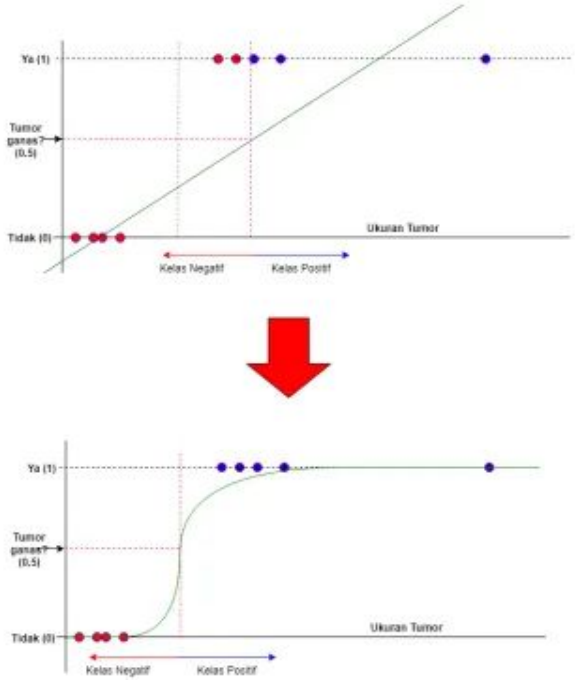
- Analisis regresi logistik merupakan suatu pendekatan untuk membuat model prediksi seperti halnya regresi linear atau yang biasa disebut dengan istilah Ordinary Least Squares (OLS) regression. Perbedaannya yaitu pada regresi logistik, peneliti memprediksi variabel terikat yang berskala dikotomi.

- Pada Analisis OLS mewajibkan syarat atau asumsi bahwa error varians (residual) terdistribusi secara normal. Sebaliknya, pada regresi logistik tidak mensyaratkan asumsi tersebut karena pada regresi logistik mengikuti distribusi logistik

Logistic Function

Regresi logistik adalah model statistik yang menggunakan fungsi logistik, atau fungsi logit, dalam matematika sebagai persamaan antara x dan y. Fungsi logit memetakan y sebagai fungsi sigmoid dari x

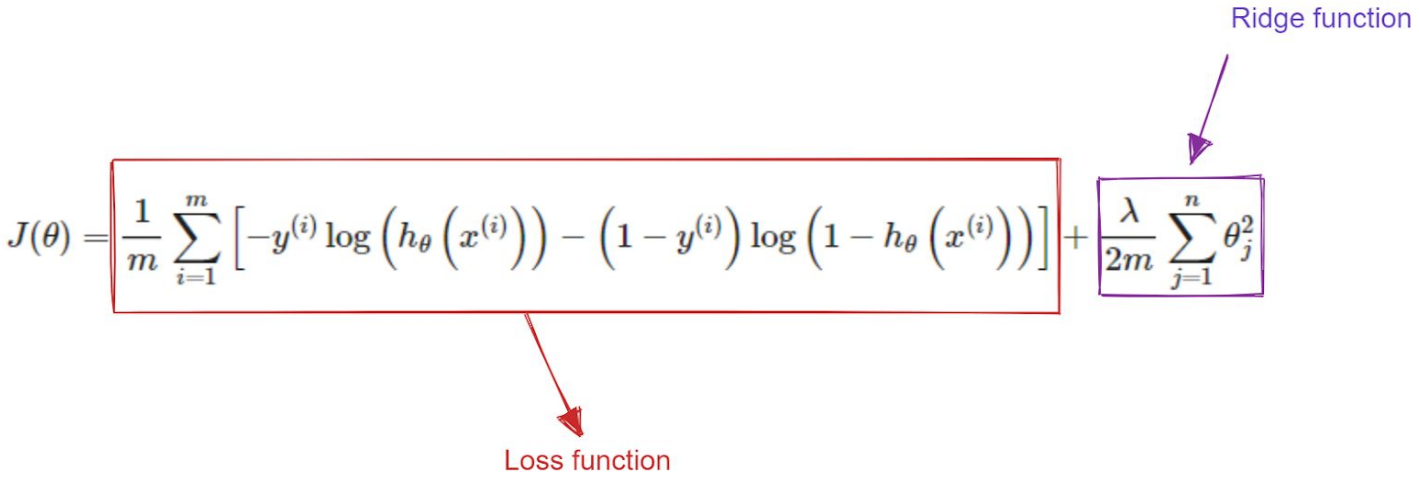




Loss Function

Loss function (fungsi biaya) pada regresi logistik digunakan untuk mengukur seberapa baik model regresi logistik memprediksi kelas aktual dari data.

Menambahkan regularization pada loss function dalam regresi logistik bertujuan untuk mencegah overfitting, meningkatkan generalisasi model pada data yang belum dilihat, dan dalam beberapa kasus, mengatasi masalah kolinearitas antar fitur



Gradient Descent

Gradient Descent adalah teknik optimasi yang digunakan untuk menemukan nilai parameter (koefisien) model yang meminimalkan fungsi kerugian (loss function). Dalam konteks regresi logistik, tujuannya adalah untuk menemukan set koefisien yang membuat model paling akurat dalam memprediksi label kelas.

Gradient Descent

Remember that the general form of gradient descent is:

$$\textit{Repeat } \{ \quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \}$$

We can work out the derivative part using calculus to get:

$$\textit{Repeat } \{ \quad \theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad \}$$

Python Code

Pseudo Code

```
class customlogisticregression():
    def __init__(self, learning_rate=0.01, iterations=1000, regularization_strength=0.01):
        initiate learning_rate
        initiate iterations
        initiate regularization_strength
        initiate Weights
        initiate bias

    def sigmoid(self, z):
        return sigmoid function

    def loss function(self, y_true, y_pred):
        # Avoid log(0) which is undefined
        determine epsilon

        # Compute binary cross entropy loss
        create loss function formula
        return loss
```

```
def fit(self, X, y):

    # Gradient Descent
    for _ in range(iterasi):
        # Compute predictions
        Create logistic model

        # Compute loss
        loss = loss function

        # Compute gradients
        compute gradient for Weights
        compute gradient for bias

        # Update parameters
        update Weights
        update bias

        # Print loss
        print iteration and loss output

def predict_proba(self, X):
    Calculate prediction probability from updated parameter
    return proba

def predict(self, X):
    transform predict probability to 0 and 1 values

def score(self, X, y=None):
    calculate accuracy score
    return accuracy
```

Code from Scratch

Langkah pertama yang dilakukan adalah dengan melakukan inisiasi pada learning rate, iteration dan regularization strength

```
def __init__(self, learning_rate=0.01, iterations=1000, regularization_strength=0.01):
    self.learning_rate = learning_rate
    self.iterations = iterations
    self.regularization_strength = regularization_strength
    self.weights = None
    self.bias = None
```

Langkah kedua adalah dengan membuat fungsi sigmoid, fungsi ini bertujuan untuk melakukan kalkulasi model logistik.

```
def binary_cross_entropy_loss(self, y_true, y_pred):
    # Avoid log(0) which is undefined
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)

    # Compute binary cross entropy loss
    loss = -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    return loss
```

Code from Scratch

Langkah ketiga menghitung loss function sebagai berikut:

- Menghitunng loss function dengan binary cross entropy loss
- menambahkan epsilon untuk menghindari hasil log(0)

```
def binary_cross_entropy_loss(self, y_true, y_pred):
    # Avoid log(0) which is undefined
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)

    # Compute binary cross entropy loss
    loss = -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    return loss
```

Code from Scratch

Langkah keempat adalah melakukan optimasi melalui gradient descent denga tahapan sebagai berikut:

- Menghitung model prediksi
- menghitung loss function
- menghitung gradient descent dengan menambahkan regularisasi.
- update parameter
- cetak output iterasi dan loss valuenya

```
def fit(self, X, y):
    n_samples, n_features = X.shape
    self.weights = np.zeros(n_features)
    self.bias = 0

    # Gradient Descent
    for _ in range(self.iterations):
        # Compute predictions
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self.sigmoid(linear_model)

        # Compute loss
        loss = self.binary_cross_entropy_loss(y, y_predicted)

        # Compute gradients
        dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y)) + (self.regularization_strength / n_s
        db = (1 / n_samples) * np.sum(y_predicted - y)

        # Update parameters
        self.weights -= self.learning_rate * dw
        self.bias -= self.learning_rate * db

        # Print loss
        if _ % 100 == 0:
            print(f"Iteration {_}, Loss: {loss}")
```

Code from Scratch

Langkah kelima adalah dengan melakukan perhitungan prediksi probability dan hotung skor akurasinya

```
def predict_proba(self, X):
    linear_model = np.dot(X, self.weights) + self.bias
    proba = self.sigmoid(linear_model)
    return proba

def predict(self, X):
    return (self.predict_proba(X)>0.5).astype("int")

def score(self, X, y=None):
    predictions = self.predict(X)
    accuracy = np.sum(predictions == y) / len(y)
    return accuracy
```

Pacmann

Implement with Dataset

Data set yang digunakan adalah Heart Disease , kita akan lakukan prediksi dengan model regresi logistik yang telah dibuat dan kita juga akan melakukan hyperparameter tuning dengan kode sebagai berikut:

```
## Hyperparameter TUning
param_grid = {
    'learning_rate': [0.001, 0.01],
    'iterations': [1000, 2000 ]
}

# GridSearchCV for hyperparameter tuning
grid_cv = GridSearchCV(model, param_grid=param_grid, cv=3)
grid_cv.fit(X_train, y_train)

print("Best hyperparameters (GridSearchCV):", grid_cv.best_params_)
print("Best score (GridSearchCV):", grid_cv.best_score_)

# Predict using the best model from GridSearchCV
best_lr_model = grid_cv.best_estimator_
predictions = best_lr_model.predict(X_test)

# Evaluate the model
accuracy = np.sum(predictions == y_test) / len(y_test)
print("Accuracy:", accuracy)
```

Data set yang digunakan adalah Heart Disease , kita akan lakukan prediksi dengan model regresi logistik yang telah dibuat dan kita juga akan melakukan hyperparameter tuning dengan kode sebagai berikut:

```
Best hyperparameters (GridSearchCV): {'iterations': 1000, 'learning_rate': 0.001}
Best score (GridSearchCV): 0.603343621399177
Accuracy: 0.5573770491803278
```

Conclusion

Conclusion & Recommendation

Konklusi

- Regresi logistik adalah mdoel untuk memprediksi variabel terikat yang berskala dikotomi.
- Asumsi klasik pada regresi linier tidak diperlukan paa regresi lopgistik.
- Regresi logistik dapat dilakukan optimalisasi dengan cara memodifikasi pada loss function dengan menambahkan epsilon untuk menghindari hasil log0, maupun dengan penambahan penalty berupa regularization dengn tujuan untuk mencegah overfitting, meningkatkan generalisasi model pada data yang belum dilihat, dan dalam beberapa kasus, mengatasi masalah kolinearitas antar fitur.
- Prediksi Heart Disease data dengan hyperparameter tuning menghasilkan nilai akurasi yang cukup rendah.

Rekomendasi

- Akurasi yang cukup rendah bisa disebabkas oleh beberapa alasan, termasuk pemilihan model maupu tuning hyper parameter tidak optimal. Oleh karena itu, diperlukan berbagai skenario optimasi parameter untuk dapat meningkatkan akurasi model.
- Selain itu data juga memiliki nilai pengaruh, Eksplorasi data yang mendalam juga diperlukan untuk mendapatkan hasil yang optimal.

Reference

- https://www.analyticsvidhya.com/blog/2022/02/implementing-logistic-regression-from-scratch-using-python/
- https://medium.com/@koushikkushal95/logistic-regression-from-scratch-dfb8527a4226
- https://python.plainenglish.io/logistic-regression-from-scratch-7b707662c8b9
- https://www.kaggle.com/datasets/dileep070/heart-disease-prediction-using-logistic-regression

Thank You