Machine Learning Package

Portfolio de algoritmos de Machine Learning

Sumário

- Os modelos de regressão representam relações entre as variáveis independentes *(features*) e uma variável dependente.
- Iremos implementar um modelo de regressão linear usando *Gradient Descent* e regularização *L2* – *RidgeRegression*
- Iremos implementar um modelo de regressão logística usando *Gradient Descent* e regularização *L2* – *LogisticRegression*

Datasets

- Os datasets estão disponíveis em:
  - https://www.dropbox.com/sh/oas4yru2r9n61hk/AADpRunbqES44W49gx9deRN5a?dl=0

- Obtém o módulo Python *ridge_regression.py* no e-learning ou em https://www.dropbox.com/sh/oas4yru2r9n61hk/AADpRunbqES44W49gx9deRN5a?dl=0

- Cria o sub-package *linear_model* e adiciona o módulo *ridge_regression*

- Obtém o módulo Python *mse.py* no e-learning ou em https://www.dropbox.com/sh/oas4yru2r9n61hk/AADpRunbqES44W49gx9deRN5a?dl=0

- Adiciona o módulo *mse* ao sub-package metrics

metrics sub-package

- No sub-package *metrics* adiciona o módulo chamado *mse.py.*
- *def mse*
  - assinatura/argumentos:
    - y_true – valores reais de Y
    - Y_pred – valores estimados de Y
  - ouput esperado:
    - O valor do erro entre *y_true* e *y_pred*
  - algoritmo:
    - Calcula o erro seguindo a formula da MSE (MQE em português):
    - sum((y_pred – y_true)**2) / (m*2)
    - m representa o número de amostras
    - $h(x^{(i)})$ representa os valores estimados
    - $Y^{(i)}$ representa os valores reais

$$J_\theta = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Objeto *RidgeRegression*

- No sub-package *linear_model*, tens o modulo *ridge_regression.py* que temo objeto *RidgeRegression.*

- *class RidgeRegression :*
  - Parâmetros:
    - l2_penalty – o coeficiente da regularização L2
    - alpha – a learning rate (taxa de aprendizagem)
    - max_iter – número máximo de iterações

  - Parâmetros estimados:
    - theta – os coeficientes/parâmetros do modelo para as variáveis de entrada (*features*)
    - theta_zero – o coeficiente/parâmetro zero. Também conhecido como interceção

  - Métodos:
    - fit – estima *theta* e *theta_zero* para o dataset de entrada
    - predict – estima a variável de saída (dependente) usando os *thetas* estimados
    - score – calcula o erro entre as previsões e os valores reais
    - cost – calcula a função de custo entre as previsões e os valores reais

Objeto *RidgeRegression*

- *RidgeRegression.fit*:

1. Estima os valores de Y $\left(h_\theta(x^{(i)}\right)$
2. Calcula o gradiente para o alfa $\alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
3. Calcula o termo de regularização L2 $\theta_j (1 - \alpha \frac{\lambda}{m})$
4. Atualiza o theta $\theta_j := \theta_j (1 - \alpha \frac{\lambda}{m}) - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
5. Atualiza o theta_zero $\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$

Objeto *RidgeRegression*

- *RidgeRegression.predict*:

  1. Estima os valores de Y usando o theta e theta_zero

$$h_q(x) = q^T x = q_0 x_0 + \ldots + q_n x_n$$

- *RidgeRegression.score*:

  1. Estima os valores de Y usando o *theta* e *theta*_zero
  2. Calcula a *mse* entre os valores reais e as previsões

Objeto *RidgeRegression*

- *RidgeRegression.cost*:

1. Obtém previsões $h_\theta(x^{(i)})$

2. Calcula o J entre os valores reais e as previsões

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

Objeto *LogisticRegression*

- Antes de implementarmos o objeto *LogisticRegression* temos de implementar a função *sigmoid*
- No sub-package *statistics*, adiciona o modulo *sigmoid_function.py* com a função *sigmoid_function.*
- *def sigmoid_function:*
  - assinatura/argumentos:
    - X – valores de entrada
  - ouput esperado:
    - A probabilidade dos valores serem iguais a 1 (função sigmoid)
  - algoritmo:
    - Calcula a função sigmoid para X seguindo a formula: $\boxed{\dfrac{1}{1+e^{-z}}}$
    - Em que Z corresponde a X

Objeto *LogisticRegression*

- Adiciona o modulo *logistic_regression.py* com o objeto *LogisticRegression* ao sub-package *linear_model*
- *class LogisticRegression:*
  - Parâmetros:
    - l2_penalty – o coeficiente da regularização L2
    - alpha – a learning rate (taxa de aprendizagem)
    - max_iter – número máximo de iterações
  - Parâmetros estimados:
    - theta – os coeficientes/parâmetros do modelo para as variáveis de entrada (*features*)
    - theta_zero – o coeficiente/parâmetro zero. Também conhecido como interceção
  - Métodos:
    - fit – estima *theta* e theta_zero para o dataset de entrada
    - predict – estima a variável de saída (dependente) usando os *thetas* estimados
    - score – calcula o erro entre as previsões e os valores reais
    - cost – calcula a função de custo entre as previsões e os valores reais

Objeto *LogisticRegression*

- *LogisticRegression.fit*:
  1. Estima os valores de Y usando a função *sigmoid_function*
  2. Calcula o gradiente para o alfa
  3. Calcula o termo de regularização L2
  4. Atualiza o *theta*
  5. Atualiza o *theta*_zero
  6. Repete os passos anteriores até atingir o número máximo de iterações (*max_iter*)

- *LogisticRegression.predict*:
  1. Estima os valores de Y usando o *theta, theta_zero* e a função *sigmoid_function*
  2. Converte os valores estimados em 0 ou 1 (binário). Valores iguais ou superiores a 0.5 tomam o valor de 1. Valores inferiores a 0.5 tomam o valor de 0.

Objeto *LogisticRegression*

- *LogisticRegression.score*:
  1. Obtém previsões usando o método *predict*
  2. Calcula a accuracy entre os valores reais e as previsões

- *LogisticRegression.cost*:
  1. Estima os valores de Y usando o *theta, theta_zero* e a função *sigmoid_function*
  2. Calcula o custo (J) entre os valores reais e as previsões usando a seguinte formula:

$$J(\theta) = \left[-\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta(x^{(i)}) + (1 - y^{(i)}) \log 1 - h_\theta(x^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Onde m corresponde ao número de exemplos *samples* e n corresponde ao número de *features*

Teste *RidgeRegression* e *LogisticRegression*

- *RidgeRegression*:
  1. Usa o dataset *cpu.csv*
  2. Usa o *sklearn.preprocessing.StandardScaler* para standardizar os dataset. cpu_dataset.X = StandardScaler().fit_transform(cpu_dataset.X)
  3. Divide o dataset em treino e teste
  4. Treina o modelo. Qual o score? E o custo?

- *LogisticRegression*:
  1. Usa o dataset *breast-bin.csv*
  2. Usa o *sklearn.preprocessing.StandardScaler* para standardizar os dataset. breast_dataset.X = StandardScaler().fit_transform(breast_dataset.X)
  3. Divide o dataset em treino e teste
  4. Treina o modelo. Qual o score? E o custo?

Avaliação

- Exercício 6: Completa as implementações dos modelos *RidgeRegression* e *LogisticRegression*

  - 6.1) Adiciona aos modelos anteriores o atributo (parâmetro estimado) *cost_history.*
    - O *cost_history* deve ser um dicionário.
    - Durante as iterações do *Gradient Descent*, computa a função de custo (*self.cost(dataset))* e armazena o resultado no dicionário *cost_history.*
    - A chave deve ser o número da iteração e o valor deve ser o custo nessa iteração.

  - 6.2) Realiza um gráfico (line plot) que permita visualizar o comportamento do custo em função do número de iterações.
    - O eixo Y deve conter o valor de custo enquanto o eixo X deve conter as iterações. Podes usar o dicionário *cost_history*.
    - Usa o dataset *cpu.csv,* o package *matplotlib* e um jupyter notebook para visualizares o comportamento da função de custo (J) no modelo *RidgeRegression*.
    - Usa o dataset *breast-bin.csv,* o package *matplotlib* e um jupyter notebook para visualizares o comportamento da função de custo (J) no modelo *LogisticRegression*.

  - NOTA: Deves usar o *sklearn.preprocessing.StandardScaler* para standardizar os dois datasets!
  - NOTA: Deves usar um número máximo de iterações superior a 1000 (max_iter=2000)

Avaliação

- Exercício 6: Completa as implementações dos modelos *RidgeRegression* e *LogisticRegression*

  - 6.3) Altera agora o algoritmo de *Gradient Descent.* Este algoritmo deve parar quando o valor da função de custo (*J/self.cost)* não se altera.
    - Quando a diferença entre o custo da iteração anterior e o custo da iteração atual for inferior a um determinado valor deves parar o *Gradient Descent.*
    - No caso do *RidgeRegression,* o critério de paragem deve ser uma diferença inferior a 1.
    - No caso do *LogisticRegression*, o critério de paragem deve ser uma diferença inferior a 0.0001.
    - Deves usar o dicionário *cost_history* para obteres o custo da iteração anterior e calcular a diferença da seguinte forma: cost_history(i-1) – cost_history(i).
    - Usa o dataset *cpu.csv,* o package *matplotlib* e um jupyter notebook para visualizares o comportamento da função de custo (J) no modelo *RidgeRegression*
    - Usa o dataset *breast-bin.csv,* o package *matplotlib* e um jupyter notebook para visualizares o comportamento da função de custo (J) no modelo *LogisticRegression*
  - NOTA: Deves usar o *sklearn.preprocessing.StandardScaler* para standardizar os dois datasets!
  - NOTA: Deves usar um número máximo de iterações superior a 1000 (max_iter=2000)

Avaliação

- Exercício 6: Completa as implementações dos modelos *RidgeRegression* e *LogisticRegression*
  - 6.4) (OPCIONAL) Adiciona uma segunda versão do algoritmo *Gradient Descent.* Este algoritmo deve diminuir o valor de alfa quando a função de custo (*J/self.cost)* não se altera.
    - Quando a diferença entre o custo da iteração anterior e o custo da iteração atual for inferior a um determinado valor deves diminuir o alfa
    - No caso do *RidgeRegression,* o critério para alterar o alfa deve ser uma diferença inferior a 1.
    - No caso do *LogisticRegression*, o critério para alterar o alfa deve ser uma diferença inferior a 0.0001.
    - Deves diminuir o valor do alfa usando a seguinte sugestão: *self.alfa = self.alfa/2*
    - Deves usar o dicionário *cost_history* para obteres o custo da iteração anterior e calcular a diferença da seguinte forma: cost_history(i-1) – cost_history(i).
    - Usa o dataset *cpu.csv,* o package *matplotlib* e um jupyter notebook para visualizares o comportamento da função de custo (J) no modelo *RidgeRegression* com o novo *Gradient Descent*
    - Usa o dataset *breast-bin.csv,* o package *matplotlib* e um jupyter notebook para visualizares o comportamento da função de custo (J) no modelo *LogisticRegression* com o novo *Gradient Descent*
  - NOTA: Proposta de implementação no slide seguinte
  - NOTA: Deves usar o *sklearn.preprocessing.StandardScaler* para standardizar os dois datasets!
  - NOTA: Deves usar um número máximo de iterações superior a 1000 (max_iter=2000)
  - NOTA: Esta alinha do exercício 6 é opcional. Implementações alternativas também serão consideradas

Avaliação

- Exercício 6: Completa as implementações dos modelos *RidgeRegression* e *LogisticRegression*

  - 6.4) (OPCIONAL) Adiciona uma segunda versão do algoritmo *Gradient Descent.* Este algoritmo deve diminuir o valor de alfa quando a função de custo (*J/self.cost)* não se altera.

  - Proposta de implementação:
    - Move o algoritmo do *Gradient Descent* atual (implementado no *fit*) para um método chamado *_regular_fit*
    - Adiciona um parâmetro ao modelo chamado *use_adaptive_alpha*
    - Cria um método alternativo chamado *_adaptive_fit*. Este método é semelhante ao método *fit* mas deve conter o novo algoritmo *Gradient Descent*.
    - Altera o método *fit* da seguinte forma:
      - Se *self.use_adaptive_alpha* é verdadeiro usa o método *_adaptive_fit*
      - Se *self.use_adaptive_alpha* é falso usa o método *_regular_fit*