wineInst

September 19, 2023

1 Primeiro Trabalho Computacional. Regressão Linear na qualidade de vinho.

1.1 1. Introdução

Está fornecido um arquivo csv (valores separados por vírgula) contendo características químicas de amostra de vinho, e sua qualidade avaliada por especialistas, em uma escala de 1 a 7. Seu objetivo é avaliar um sistema de regressão linear sobre este problema.

Os módulos, classes e definições contidas aqui devem ser vistas como sugestão. Desde que você use Python, apresente seu trabalho como um notebook e responda às perguntas, os requisitos estão satisfeitos. Eu também não tenho objeções ao uso de código encontrado na internet ou via chatGPT, mas use-os por sua conta e risco.

1.2 2. Classes e Métodos

Com estes módulos, classes e métodos que se seguem, o laço principal de treinamento fica muito simples. Lembre-se de ativar o d2l antes de invocar o notebook. Estou usando também o módulo sklearn para a divisão do arquivo em treinamento e validação. É preciso instalá-lo com pip install sklearn no terminal.

[1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split
from d2l import tensorflow as d2l
import random
```

[2]:
```
class LinearRegression(d2l.Module):  #@save
    """The linear regression model implemented with high-level APIs."""
    """Do d2l"""
    def __init__(self, lr):
        super().__init__()
        self.save_hyperparameters()
        initializer = tf.initializers.RandomNormal(stddev=0.01)
        self.net = tf.keras.layers.Dense(1, kernel_initializer=initializer)
```

[3]:
```
class LinearRegressionData(d2l.DataModule):  #@save
    """Data for linear regression."""
```

```
    def __init__(self, nInputs,data,num_train=5197, num_val=1300,
                 batch_size=32):
        super().__init__()
        self.save_hyperparameters()
        self.X=data[:,0:nInputs]
        self.y=data[:,nInputs]
```

[4]:
```
@d2l.add_to_class(d2l.DataModule)  #@save
def get_tensorloader(self, tensors, train, indices=slice(0, None)):
    tensors = tuple(a[indices] for a in tensors)
    shuffle_buffer = tensors[0].shape[0] if train else 1
    return tf.data.Dataset.from_tensor_slices(tensors).shuffle(
        buffer_size=shuffle_buffer).batch(self.batch_size)

@d2l.add_to_class(LinearRegressionData)  #@save
def get_dataloader(self, train):
    i = slice(0, self.num_train) if train else slice(self.num_train, None)
    return self.get_tensorloader((self.X, self.y), train, i)
```

1.3 3. Leitura do Arquivo e Pré-processamento

Existem 12 variáveis de entrada e 1 alvo. A primeira coluna é uma literal “white” ou “red” . Com a criação da variável dummy, teremos 13 entradas. Há 6497 exemplos, e eles não são equilibrados (há mais brancos que tintos, e mais vinhos de boa qualidade que de má qualidade). **Há dados faltantes.**

white | red | fixed acidity | volatile acidity | citric acid \

0 True False | 7.0 | 0.270 | 0.36
1 True False 6.3 0.300 0.34
2 Tru e False 8 8.1 0.280 0.40
3 Tr ue False 7.2 0.230 0.32
4 T rul False 7.2 0.230 0.32
… … … … … …
6492 False True 6.2 0.600 0.08
6493 False True 5.9 0.550 0.10
6494 False True 6.3 0.51 0.13
6495 False True 5.9 0.65 0.12
6496 False True 6.0 0.10 0.47

residual sugar chlorides free sulfur dioxide total sulfur dioxide \

0 20.7 0.045 45.0 170.0
1 1.6 0.049 14.0 132.0
2 6.9 0.050 30.0 97.0
3 8.5 0.058 47.0 186.0
4 8.5 0.058 47.0 186.0
… … … … …
6492 2.0 0.090 32.0 44.0
6493 2.2 0.062 39.0 51.0

6494 2.3 0.076 29.0 40.0
6495 2.0 0.075 32.0 44.0
6496 3.6 0.067 18.0 42.0

	density	pH	sulphates	alcohol	quality
0	1.00100	3.00	0.450000	8.8	6
1	0.99400	3.30	0.490000	9.5	6
2	0.99510	3.26	0.440000	10.1	6
3	0.99560	3.19	0.400000	9.9	6
4	0.99560	3.19	0.400000	9.9	6
…	…	…	…	…	…
6492	0.99490	3.45	0.580000	10.5	5
6493	0.99512	3.52	0.531215	11.2	6
6494	0.99574	3.42	0.750000	11.0	6
6495	0.99547	3.57	0.710000	10.2	5
6496	0.99549	3.39	0.660000	11.0	6

[6497 rows x 14 columns]

Você deve ainda usar a função train_test_split (ou outra) para separar estes arquivos em treinamento e validação. Sugiro 20% para a validação.

[6]:
```
train_data, val_data = train_test_split(data, test_size=0.2)
train_data.shape, val_data.shape
```

[6]:
```
((5197, 14), (1300, 14))
```

Finalmente, lembre-se que a função pd.read_csv cria um dataframe, não uma matriz. Verifique como transformar em uma matriz, que deve conter todos os exemplos de treinamento e em seguida todos os exemplos de teste. Sua dimensão final será 6497 por 14.

Concatenar 1s Tentar aplicar Wopt= inv (XT X) XT Y

[8]:
```
Xb = tf.concat((dataMatrix[:,0:13], tf.ones((6497, 1),dtype=tf.double)),axis=1)
y=dataMatrix[:,13]
#quad=tf.matmul(tf.transpose(Xb),Xb)
#quad
W=tf.linalg.matvec(tf.linalg.matmul(tf.linalg.inv(tf.matmul(tf.
 ↪transpose(Xb),Xb)),tf.transpose(Xb)),y)
#yhat=tf.linalg.matvec(Xb,W)
#yhat[0:5]
W
```

[8]:
```
<tf.Tensor: shape=(14,), dtype=float64, numpy=
array([-1.32314648e+03, -1.27173047e+03,  8.48257125e-02, -1.48894237e+00,
       -6.07625080e-02,  6.24849078e-02, -7.69513034e-01,  4.93139669e-03,
       -1.41086914e-03, -1.04005678e+02,  4.95887420e-01,  7.24766700e-01,
        2.22394642e-01,  1.42598633e+03])>
```

2 Métodos para treinamento

Estes métodos definem como se dá o passo à frente (processamento da rede), qual a função de custo e otimizador.

[9]:
```
@d2l.add_to_class(LinearRegression)  #@save
def forward(self, X):
    return self.net(X)
```

[10]:
```
@d2l.add_to_class(LinearRegression)  #@save
def loss(self, y_hat, y):
    fn = tf.keras.losses.MeanSquaredError()
    return fn(y, y_hat)
```

[11]:
```
@d2l.add_to_class(LinearRegression)  #@save
def configure_optimizers(self):
    return tf.keras.optimizers.SGD(self.lr)
```

2.1 4. Treinamento

Com esses passos, o laço de treinamento é bem simples. Veja o exemplo visto em sala. ATENÇÂO: Os dados têm variância (escala) desigual e relativamente grande. Como nada limita a saída da rede, uma taxa de aprendizado alta leva facilmente à divergência. Se o gráfico do erro fica em branco ou o treinamento leva tempo demais, este é provavelmente o caso. Diminua a taxa para algo em torno de 10^-5 .

2.2 5. Avaliações e análise

Terminado o treinamento, houve (espera-se) uma minimização do erro quadrático médio. Mas outras medidas de desempenho podem ser usadas.

Para avliar o modelo, pode ser útil a seguinte função:

`prediction=model.call(inputs)`

Supondo que você instanciou seu modelo como "model". Para fazer operações (cálculo de médias e erros), pode ser necessário mudar (reshape) algum tensor. Lembre-se de usar os dados de VALIDAÇÂO. Uma média de vários treinamentos com condiçoes iniciais (e partições da base dados) diferentes nos dá um resultado mais confiável.

2.2.1 5.1. Erro absoluto médio e Preditor Trivial

Qual o erro absoluto médio do seu preditor? Como a categoria é dada por inteiros de 1 a 7, esperamos, no mínimo, um erro menor do que 1, ou o sistema frequentemente erraria por mais de uma categoria.

Por vezes queremos saber se, depois de todo este trabalho temos algo superior a um preditor trivial ou ingênuo. O preditor trivial atribui como saída a média da função que se quer prever.

Compare os erros absolutos médios. O seu regressor é melhor do que uma simples média da saída?

Atenção: Este problema é fortemente não-linear, e fizemos pouco pré-processamento com os dados. Não se espera um bom desempenho com este primeiro modelo simples. Ele deve ser melhor que o preditor trivial, mas não por muito.

2.2.2 5.2. Classificação por arredondamento

Como a classificação é um índice inteiro, podemos pensar neste caso como um problema de classificação em 7 classes, e não de regressão. Veremos ferramentas mais indicadas para esta classe de problemas, mas uma solução trivial seria arredondar a saída do preditor e verificar quantas classes acertamos.

Compare a taxa de acerto (vamos chamar isso acurácia mais à frente) contra o preditor-classificador trivial (a média das classificações está próxima a 6). Um preditor aleatório (que atribui um número de 1 a 7 aleatoriamente) deveria ter uma taxa de acerto de 14%.

Atenção , de novo, você deve estar acima do preditor trivial, mas não por muito, e bem acima dos 14%.

2.2.3 5.3. Correlação

Outra medida interessante para saber o quanto nosso preditor absorveu a informação contida no arquivo de dados é a **correlação** entre saída desejada e obtida. Este é um número de -1 a 1, -1 indicando uma relação determinística negativa (quando a saída do preditor é baixa, o especialista deu nota alta e vice_versa), 0 é ausência de correlação (a saída de um não nos permite dizer nada sobre o outro) e +1 significa que o preditor substitui o especialista sem qualquer erro. Gostaríamos, é claro, de um valor próximo a 1. Observe que o preditor aleatório e também o trivial têm correlaçao zero com a saída desejada.

2.2.4 5.4. Melhoramentos (extra)

Não se espera mesmo um bom desempenho neste caso, mas algumas coisas podem ajudar. Não são requisitos para o trabalho, mas serão considerados pontos extra.

- Variações no número de épocas, tamanho do "minibatch", taxas de aprendizado.
- Normalização das entradas (para que tenham média zero e desvio-padrão 1).
- (mais difícil) Uma análise estatística que elimine variáveis pouco informativas, ou proponha uma combinação linear mais informativa de algumas variáveis.

2.2.5 5.5. Uma provocação

É verdade que este sistema, baseado apenas em regressão linear, não tem um desempenho excelente. Mas, já podemos fazer uma reflexão. Trabalhos publicados com esta base de dados têm acurácia de 85% na classificação. Um sistema como este nível de desempenho, treinado com modelos mais complexos e as técnicas de aprendizado não muito diferentes da que usamos aqui, é "realmente inteligente"? Ele "realmente aprendeu" a classificar vinhos?

[ ]: