eysmjrqb4

April 9, 2025

0.1

[1]:
```
import pandas as pd
import torch
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import seaborn as sns
from scipy import stats
import numpy as np
import torch.nn as nn

df = pd.read_csv('winequality-red.csv', sep=';')
tensor_data = torch.tensor(df.values, dtype=torch.float32)
print(tensor_data)
```

```
tensor([[ 7.4000,  0.7000,  0.0000,  …,  0.5600,  9.4000,  5.0000],
        [ 7.8000,  0.8800,  0.0000,  …,  0.6800,  9.8000,  5.0000],
        [ 7.8000,  0.7600,  0.0400,  …,  0.6500,  9.8000,  5.0000],
        …,
        [ 6.3000,  0.5100,  0.1300,  …,  0.7500, 11.0000,  6.0000],
        [ 5.9000,  0.6450,  0.1200,  …,  0.7100, 10.2000,  5.0000],
        [ 6.0000,  0.3100,  0.4700,  …,  0.6600, 11.0000,  6.0000]])
```

0.2 +

[2]:
```
list_tensor = []
for i in tensor_data:
    l = []
    if i[0] > 7.0 and i[0] < 10.5:
        l.append(1)
    else:
        l.append(0)
    if i[1] <= 0.6:
        l.append(1)
    else:
        l.append(0)
    if i[2] >= 0.2:
        l.append(1)
    else:
        l.append(0)
    if i[3] > 1.5 and i[3] < 3.5:
        l.append(1)
    else:
        l.append(0)
    if i[4] <= 0.08:
        l.append(1)
    else:
        l.append(0)
    if i[5] > 10.0 and i[5] < 35.0:
        l.append(1)
    else:
        l.append(0)
    if i[6] > 20.0 and i[6] < 100.0:
        l.append(1)
    else:
        l.append(0)
    if i[7] > 0.995 and i[7] < 0.998:
        l.append(1)
    else:
        l.append(0)
    if i[8] > 3.1 and i[8] < 3.5:
        l.append(1)
    else:
        l.append(0)
    if i[9] >= 0.6:
        l.append(1)
    else:
        l.append(0)
    if i[10] > 0.1:
        l.append(1)
    else:
        l.append(0)
    if i[11] > 5.0:
        l.append(1)
    else:
        l.append(0)
    list_tensor.append(l)
list_tensor = torch.tensor(list_tensor)
print(list_tensor)
```

```
tensor([[1, 0, 0,  …, 0, 1, 0],
        [1, 0, 0,  …, 1, 1, 0],
        [1, 0, 0,  …, 1, 1, 0],
        …,
        [0, 1, 0,  …, 1, 1, 1],
        [0, 0, 0,  …, 1, 1, 0],
        [0, 1, 1,  …, 1, 1, 1]])
```

0.3

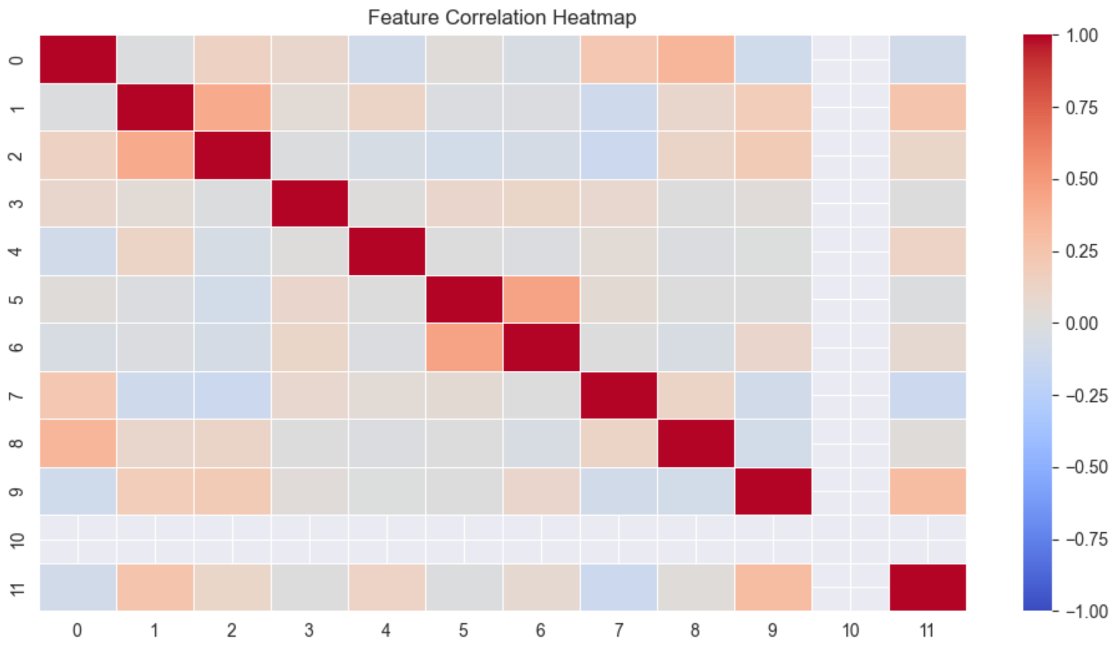
```
[3]: list_tensor = list_tensor.clone().detach().to(dtype=torch.float32)
     #
     X = list_tensor[:, :-1]   #
     y = list_tensor[:, -1]   #

     #
     X_std = X.std(dim=0)
     y_std = y.std()

     print("    :", X_std)
     print("    :", y_std)

     #       NaN   Inf
     print("      NaN:", torch.isnan(X).any() or torch.isnan(y).any())
     print("      Inf:", torch.isinf(X).any() or torch.isinf(y).any())


     #
     def pearson_correlation(X, y):
         #
         X_mean = X.mean(dim=0, keepdim=True)
         y_mean = y.mean()

         #
         X_centered = X - X_mean
         y_centered = y - y_mean

         #
         X_std = X_centered.pow(2).sum(dim=0).sqrt()
         y_std = y_centered.pow(2).sum().sqrt()

         #
         valid_features = X_std != 0
         X_centered = X_centered[:, valid_features]
         X_std = X_std[valid_features]

         #
         covariance = (X_centered * y_centered.unsqueeze(1)).sum(dim=0)

         #
         correlation = torch.zeros(X.shape[1])  #       0
         correlation[valid_features] = covariance / (X_std * y_std)
```

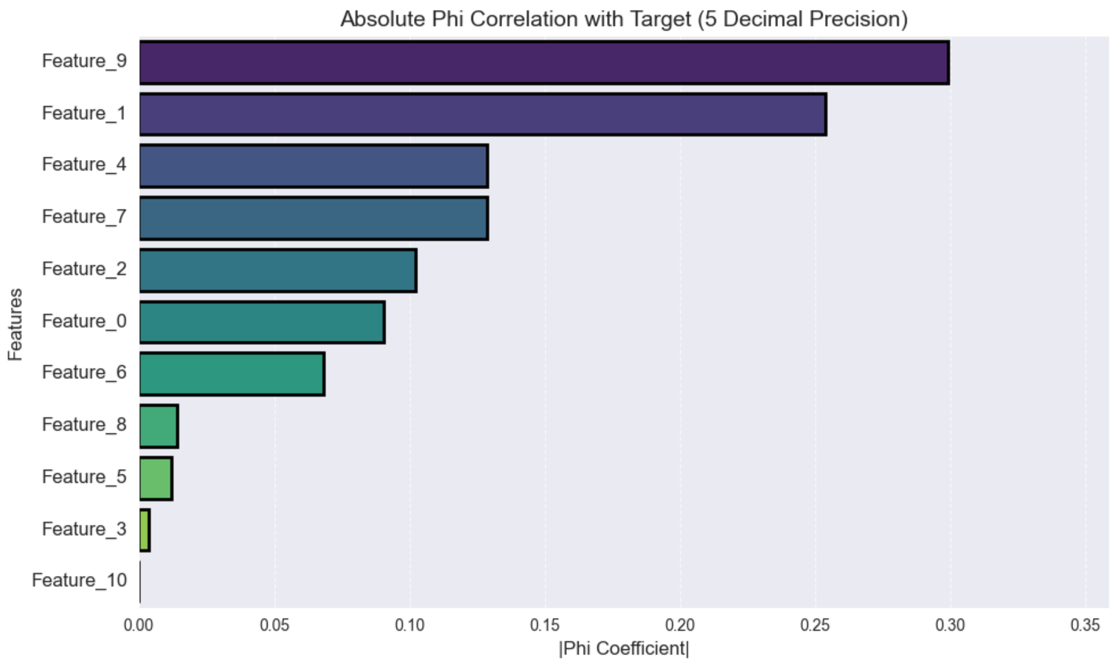
```
    return correlation


#
correlations = pearson_correlation(X, y)

#
best_feature_index = correlations.abs().argmax().item()

print(f"    : {best_feature_index}")
print(f"  : {correlations[best_feature_index].item()}")
```

```
   : tensor([0.4760, 0.4647, 0.4853, 0.3609, 0.4980, 0.4960, 0.4619, 0.4874,
0.3983,
        0.4918, 0.0000])
   : tensor(0.4989)
     NaN: tensor(False)
     Inf: tensor(False)
   : 9
  : 0.30015796422958374
```

[4]:
```
#     ---------------------------------------------
plt.figure(figsize=(12, 6))

#
corr_matrix = pd.DataFrame(list_tensor[:, :]).corr()
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm', vmin=-1.0, vmax=1.0,␣
 ↪fmt=".2f", linewidths=.5)
#sns.heatmap(corr_matrix, annot=False, cmap='coolwarm',fmt=".2f",linewidths=.5)
plt.title('Feature Correlation Heatmap')
plt.savefig('heatmap.png')  #
plt.show()

#
data = pd.DataFrame(list_tensor[:, :-1])
data['target'] = list_tensor[:, -1]  #


#       Phi
def phi_coefficient(col1, col2):
    confusion_matrix = pd.crosstab(col1, col2)
    n = confusion_matrix.sum().sum()
    chi2 = stats.chi2_contingency(confusion_matrix)[0]
    return np.sqrt(chi2 / n)


#        Phi
```

```python
phi_values = {}
for col in data.columns[:-1]:
    phi = phi_coefficient(data[col], data['target'])
    phi_values[col] = abs(phi)  #

#    DataFrame
phi_df = (
    pd.DataFrame({
        'Feature': [f"Feature_{i}" for i in phi_values.keys()],  #
        'Abs_Phi_Correlation': np.round(list(phi_values.values()), 5)  #  5
    })
    .sort_values('Abs_Phi_Correlation', ascending=False)
    .reset_index(drop=True)
)

#
plt.figure(figsize=(10, 6))
sns.barplot(
    x='Abs_Phi_Correlation',
    y='Feature',
    hue='Feature',
    data=phi_df,
    palette='viridis',
    legend=False,
    dodge=False,  #
    linewidth=2,  #
    edgecolor='black',  #
    saturation=0.8  #
)

#
plt.gca().set(yticklabels=[])  #   y
plt.yticks(ticks=range(len(phi_df)),
           labels=phi_df['Feature'],
           fontsize=12,
           va='center')  #

#
plt.xlim(0, phi_df['Abs_Phi_Correlation'].max() * 1.2)  #
plt.gcf().subplots_adjust(left=0.3)  #

#
plt.title('Absolute Phi Correlation with Target (5 Decimal Precision)',␣
 ↪fontsize=14)
plt.xlabel('|Phi Coefficient|', fontsize=12)
plt.ylabel('Features', fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

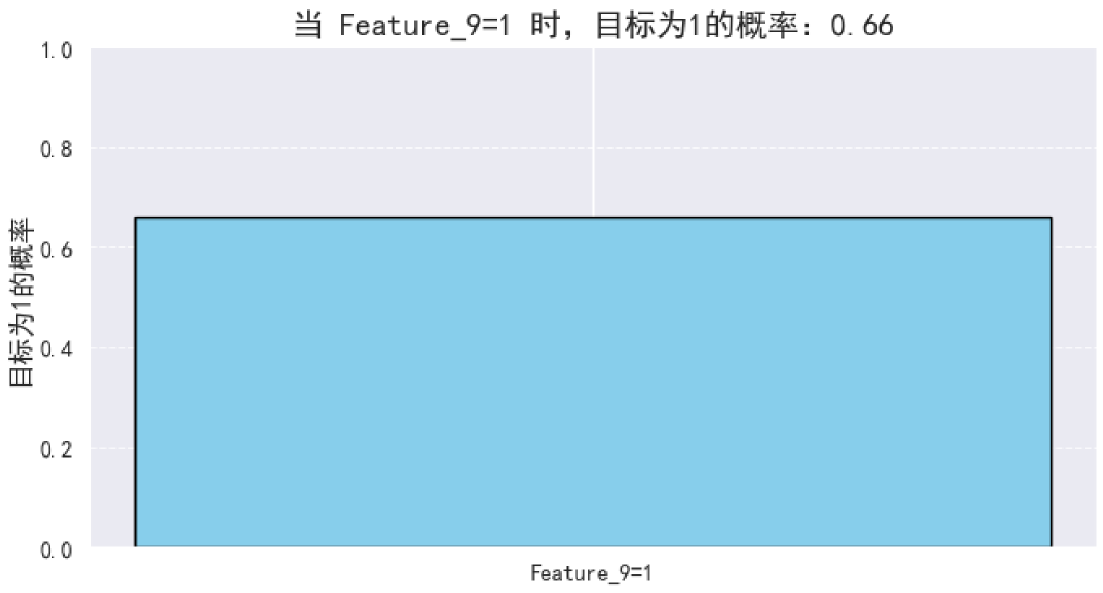




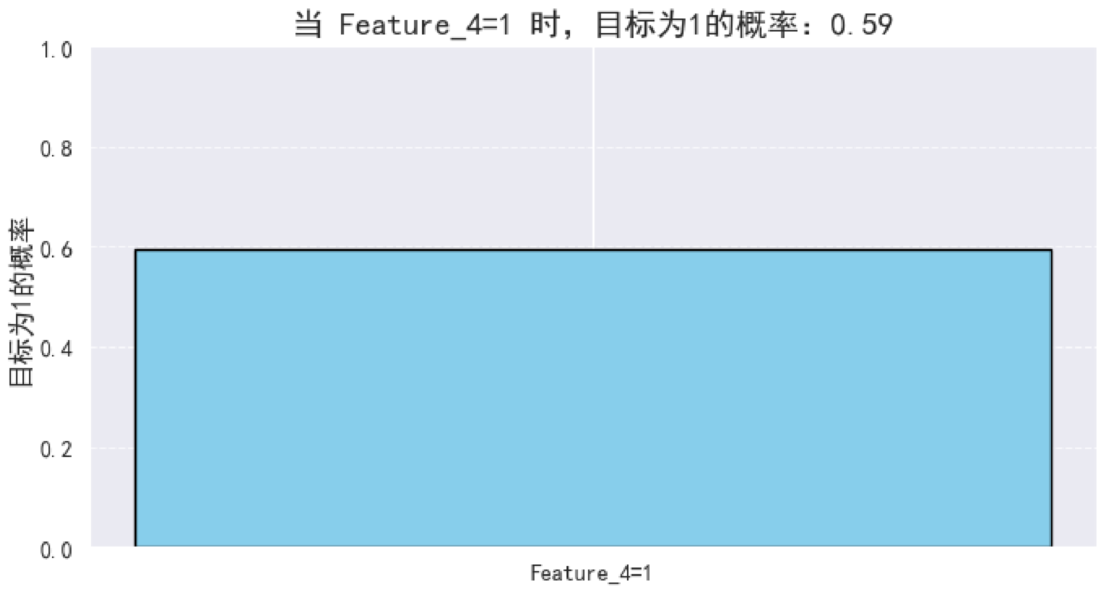
0.4 9 4 7

[5]:
```
#
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False


def plot_feature_probability(feature_column, target_column, feature_name):
    #        numpy
    feature = list_tensor[:, feature_column].numpy().astype(int)
    target = list_tensor[:, target_column].numpy().astype(int)

    #    1
    mask = (feature == 1)
    if mask.sum() == 0:  #
        print(f"    {feature_name}    1   ")
        return

    #    1
    prob = target[mask].mean()

    #
    plt.figure(figsize=(8, 4))
    plt.bar([f"{feature_name}=1"], [prob], color='skyblue', edgecolor='black',␣
  ↪width=0.5)
    plt.ylim(0, 1)
    plt.ylabel("  1  ", fontsize=12)
    plt.title(f"  {feature_name}=1    1  {prob:.2f}", fontsize=14)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
```

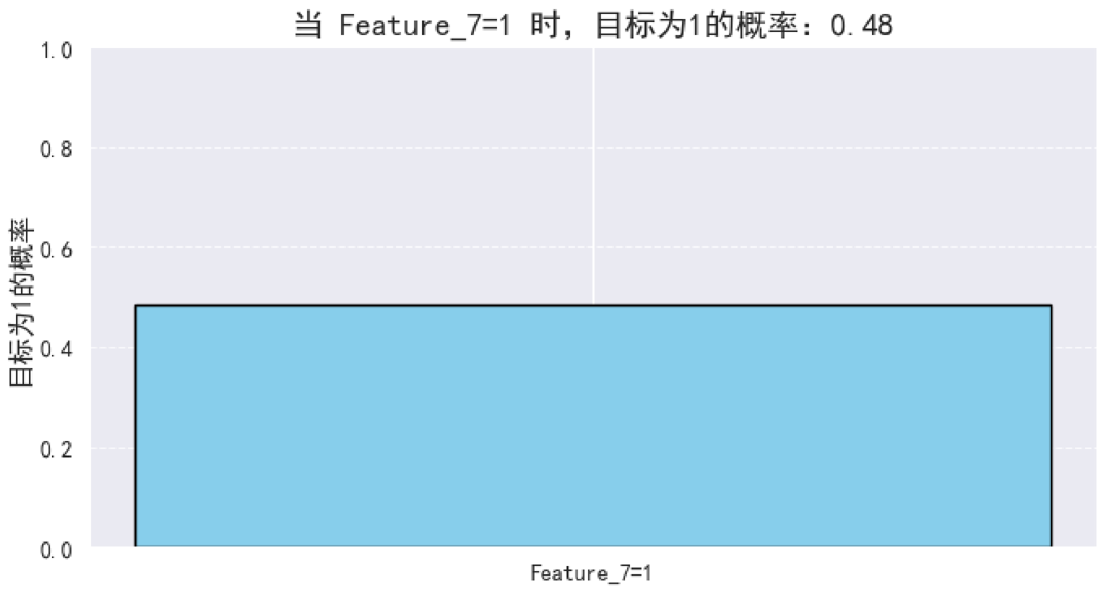
[6]:
```
plot_feature_probability(feature_column=9, target_column=-1,␣
  ↪feature_name="Feature_9")
```



[7]:
```
plot_feature_probability(feature_column=4, target_column=-1,␣
 ↪feature_name="Feature_4")
```



[8]:
```
plot_feature_probability(feature_column=7, target_column=-1,␣
 ↪feature_name="Feature_7")
```



0.5 + SGD

```
[15]: def my_sgd_loss(l, target, lr, batch_size=40, num_epochs=40):
          w = torch.ones(l.shape[1])
          w = torch.nn.init.normal_(w, mean=-1.0, std=1.5)
          b = 0.0
          loss = 0
          for j in range(num_epochs):
              for epoch in range(num_epochs):
                  if epoch != j:
                      i = epoch * batch_size
                      batch_X = l[i:i + batch_size]
                      batch_y = target[i:i + batch_size]
                      z = torch.matmul(batch_X, w) + b
                      sig = 1 / (1 + torch.exp(-z))
                      delta_w = torch.matmul(batch_X.T, (sig - batch_y)) / batch_size
                      delta_b = torch.sum((sig - batch_y)) / batch_size
                      w -= delta_w * lr
                      b -= delta_b * lr
              loss += my_loss(j, w, b, l, target, batch_size)
          return loss / num_epochs


      def my_dataloader(list_tensor):
          list_tensor = list(list_tensor)
```

```
    l = []
    target = []
    for i in list_tensor:
        #
        features = [i[4], i[7], i[9]]
        l.append(features)
        #
        target.append(i[-1])
    #    target
    return torch.tensor(l), torch.tensor(target).squeeze()


def my_loss(flag, w, b, l, target, batch_size):
    i = flag * batch_size
    test_data = l[i:i + batch_size]
    #      .T
    p = torch.matmul(test_data, w) + b  #
    p = torch.where(p >= 0.5, 1.0, 0.0)  #
    correct = (p == target[i:i + batch_size]).sum().item()
    loss = correct / len(test_data)
    return loss


l, target = my_dataloader(list_tensor)
loss = my_sgd_loss(l, target, 0.01)
print(f'     {loss}')
```

```
    0.48977564102564103
```

0.6 Pytorch “ ”

[11]:
```
net = nn.Sequential(
    nn.Flatten(),
    nn.Linear(in_features=3, out_features=8),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(in_features=8, out_features=1),
)


def init_weights(m):
    if type(m) == nn.Linear:
        nn.init.normal_(m.weight, mean=-1.0, std=0.05)


net.apply(init_weights)
batch_size, lr, num_epochs = 40, 0.01, 40
```

```python
optimizer = torch.optim.Adam(net.parameters(), lr=lr, weight_decay=0.01)
loss = nn.BCEWithLogitsLoss()


def train_evaluate(l, batch_size=40, num_epochs=40):
    val_loss = 0
    correct = 0
    total = 0
    val_accuracy = 0
    for j in range(num_epochs):
        net.train()  #
        for i in range(num_epochs):
            if j != i:
                batch_x = l[i * batch_size:(i + 1) * batch_size]
                batch_y = target[i * batch_size:(i + 1) * batch_size]
                optimizer.zero_grad()
                my_loss = loss(net(batch_x).squeeze(), batch_y)
                my_loss.backward()
                optimizer.step()
                #
        #  epoch         epoch  batch
        net.eval()  #
        with torch.no_grad():
            batch_x = l[j * batch_size:(j + 1) * batch_size]
            batch_y = target[j * batch_size:(j + 1) * batch_size]
            #      batch
            eval_batch_x = l[j * batch_size:(j + 1) * batch_size]
            eval_logits = net(eval_batch_x).squeeze()  #
            #
            val_loss += loss(eval_logits, batch_y).item() * len(batch_x)

            probabilities = torch.sigmoid(eval_logits)  #
            predictions = (probabilities >= 0.5).float()
            correct += (predictions == batch_y).sum().item()
            total += len(batch_y)

            # ACC
            val_accuracy += correct / total
    avg_val_accuracy = val_accuracy / num_epochs
    print(f'    {avg_val_accuracy:.5f}')
    avg_loss = val_loss / len(l)
    print(f'   {avg_loss:.5f}')


#
l, target = my_dataloader(list_tensor)
init_weights(net)
```

```
train_evaluate(l, batch_size=40, num_epochs=40)
```

```
 0.56422
0.65475
```