مبانی هوش محاسباتی، تمرین دوم

گزارش:

بخش اول تمرین شامل مراحل زیر است:

- خواندن دیتاست
- پیدا کردن مقادیر null در دیتاست و جایگزینی آن‌ها با وسیله تابع mean
- پیدا کردن فیچرهایی که مقدار آن‌ها غیر عدد است(object) و جایگزینی آن به وسیله labelencoder
- نرمال سازی داده‌ها (برای اینکه بین یک رنج خاصی قرار بگیرند و داده‌ها به سمت فیچر خاصی بایاس نشوند)

پس از آن می‌توان با استفاده از لایبرری sklearn ، پرسپترون ساده (linear) را روی دیتا اجرا کرد. دقت حاصل به صورت زیر است:

simple Perceptron

```
[152] from sklearn.linear_model import Perceptron

      model = Perceptron()
      model.fit(X_train, y_train)

      preds = model.predict(X_test)
      preds
```

```
array([1, 1, 0, ..., 0, 1, 0])
```

accuracy

```
from sklearn.metrics import accuracy_score

acc = accuracy_score(y_test, preds) * 100
acc
```

```
81.50215583615645
```

بخش دوم:

از آن جایی که پرسپترون ساده برای دیتاهای غیرخطی خوب کار نمی‌کند از کرنل استفاده می کنیم. برای این کار کافی است کلاسی به نام KernelPreceptron ساخته شود که در تابع fit آن برای محاسبات وزن از کرنل استفاده می شود.

kernel perceptron

```
[ ]  import numpy as np

     class KernelPerceptron(object):

       def polynomial_kernel(x, y, p=3):
         return (1 + np.dot(x, y)) ** p

       def __init__(self, eta=0.01, n_iteration=1, kernel=polynomial_kernel):
         self.eta = eta
         self.n_iteration = n_iteration
         self.kernel = kernel

       # a function to calculate (1+X[i].X[j])^P and build kernel matrix
       def kernelCalculator(self, samples, X_train):
         KernelMatrix = np.zeros((samples, samples))
         for i in range(samples):
           for j in range(samples):
             KernelMatrix[i,j] = self.kernel(X_train[i], X_train[j])

         return KernelMatrix

       def fit(self, X, y):
         # total samples=103904 , features=22
         n_samples = len(X)
         n_features = len(X[0])

         self.alpha = np.zeros(n_samples, dtype=np.float64)

         K = self.kernelCalculator(n_samples, X)
         for _ in range(self.n_iteration):
           for i in range(n_samples):
             if np.sign(np.sum(self.alpha * y * K[:,i])) != y[i]:    # if np.sign(np.sum(self.alpha * y * K[:,i])) == -1
               self.alpha[i] += 1.0

       #def predict(self, X):
         #TODO
```

کانستراکتور این کلاس شامل تعداد iteration ها، لرنینگ ریت(که البته استفاده نمی شود) و نوع کرنل است. ما از کرنل polynomial استفاده می کنیم.

- Polynomial of power p: $K(x_i,x_j) = (1+ x_i^T x_j)^p$

برای تابع فیت هم یک آرایه به نام آلفا به اندازه تعداد سمپل ها در نظر میگیریم. این بار، به جای ضرب داخلی فیچرها باید کرنلی از فیچر ها را درنظر بگیریم که مقادیرش در ماتریس K ذخیره می‌شود (و طبق فرمول بالا محاسبه می شود). در دو حلقه بعدی، به تعداد سمپل ها، باید بررسی کنیم که چه تعداد ارور وجود دارد. با پیدا شدن هر خطا، به مقدار الفا اضافه میکنیم. درواقع الفا تعداد دفعاتی است که x به اشتباه classified شده (این موارد با استفاده از الگوریتم مربوطه نتیجه‌گیری شده است.)

Initialize $\boldsymbol{\alpha}$ to an all-zeros vector of length n, the number of training samples.
For some fixed number of iterations, or until some stopping criterion is met:

For each training example $\mathbf{x}_j$, y_j:

Let $\hat{y} = \operatorname{sgn} \sum_{i}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j)$

If $\hat{y} \neq y_j$, perform an update by incrementing the mistake counter:

$a_j \leftarrow a_j + 1$

برای ترین کردن داده‌ها از الگوریتم بالا استفاده می شود. برای پردیکت هم از الگوریتم زیر استفاده می کنیم:

Inputs: a sequence of training data $D_{\text{test}} = \{(\underline{x}_1, y_1) \cdots (\underline{x}_M, y_M)\}$ where $\underline{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, 1\}$

Initialise: get saved alphas from training, $\underline{\alpha}^\star$

For each test point:

- **Predict:** $\hat{y}_i = \operatorname{sign} \sum_{j=1}^{m} w_j^\star K(\underline{x}_i, \underline{x}_j)$

Return: predicted $\hat{y}$ vector

تابع پردیکت هم کامل نشد.

برای ترین کردن دیتا هم با توجه به اینکه رم به ما اجازه ترین کردن ۱۰۳۹۰۴ سمپل را نمی‌دهد می‌توان از بخش کوچکی از سمپل استفاده کرد.(مثلا ۲۰۰۰ تا)

محیا احسانی مهر