Pattern Recognition and Machine Learning: Homework 6

Qingru Hu   2020012996

April 10, 2023

Problem 1

I use `hmmlearn` module to build HMM models.

(1)

I use the `CategoricalHMM` model in `hmmlearn` to train the dataset, and I obtain from fitting the initial , the transition and emission probabilities, shown respectively in , Fig.2 and Fig.3.

Dice Type	Dice 1	Dice 2
Initial Prob	0.618	0.382

Table 1: The initial probabilities

Dice Type	Dice 1	Dice 2
Dice 1	0.888	0.112
Dice 2	0.156	0.844

Table 2: The transition probabilities

Dice/Point	1	2	3	4	5	6
Dice 1	0.158	0.164	0.184	0.171	0.191	0.132
Dice 2	0.120	0.098	0.096	0.108	0.088	0.491

Table 3: The emission probabilities

The code is shown as below.

```
import numpy as np
from hmmlearn import hmm

data = np.load('sequences.npy')
X = data.reshape(200*30, 1)
lens = np.ones(data.shape[0])*30
lens = lens.astype(int)
model = hmm.CategoricalHMM(n_components=2, random_state=10)
model.fit(X, lens)
model.score(X)
```

```
# -10434.902086730863
```

(2)

Forward Algorithm

The probability of observing sequence 6 6 6 6 using forward algorithm is $p = 0.015$.

```
  iprob = model.startprob_
  tprob = model.transmat_
  eprob = model.emissionprob_

  for t in range(4):
      if t==0:
          a0 = eprob[0, 6]*iprob[0]
          a1 = eprob[1, 6]*iprob[1]
      else:
          a0 = eprob[0, 6]*(a0*tprob[0, 0] + a1*tprob[1, 0])
          a1 = eprob[1, 6]*(a0*tprob[0, 1] + a1*tprob[1, 1])
  p = a0 + a1
  # p = 0.014626307201743518
```

Backward Algorithm

The probability of observing sequence 6 6 6 6 using backward algorithm is $p = 0.015$.

```
  iprob = model.startprob_
tprob = model.transmat_
eprob = model.emissionprob_

for t in [3, 2, 1, 0]:
    # as = np.zeros([2, 2])
    if t==3:
        b0 = 1
        b1 = 1
    else:
        b0 = tprob[0, 0]*eprob[0, 6]*b0 + tprob[0, 1]*eprob[1, 6]*b1
        b1 = tprob[1, 0]*eprob[0, 6]*b0 + tprob[1, 1]*eprob[1, 6]*b1
p = a0 + a1
# p = 0.014626307201743518
```

(3)

This player is cheating and he switched his dice on his 12th roll.

```
seq = np.array([3, 2, 1, 3, 4, 5, 6, 3, 1, 4, 1, 6, 6, 2, 6])
seq = seq.reshape(1, -1)
model.decode(seq)
```

```
# log_prob = -28.45720629383466,
# state_sequence = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
```

Problem 2

In this problem, the datasets 'iris', 'breast_cancer' and 'usps' are represented as I, B and U. I select hyper-parameter configuration that has the largest average validation accuracy as the optimal for each model.

2.1

```
models = {
    "Decision Tree": DecisionTreeClassifier,  # criterion, max_depth
    "Random Forest": RandomForestClassifier, # criterion, n_estimators
    "Bagging": BaggingClassifier, # n_estimators
    "Gradient Boosting": GradientBoostingClassifier,
    # n_estimators, learning_rate
    # "XGBoost":
    "Naive Bayes": GaussianNB, # var_smoothing
    "Perceptron": Perceptron, # penalty, alpha
    "Logistic Regression": LogisticRegression, # penalty, C
    "LDA": LinearDiscriminantAnalysis,
    "SVM": SVC, # kernel, C
}
```

2.2

```
model_hparams_best = {
    "Decision Tree": dict(criterion='gini', max_depth=5),
    "Random Forest": dict(n_estimators=50, criterion='entropy'),
    "Bagging": dict(n_estimators=20),
    "Gradient Boosting": dict(learning_rate=0.1, n_estimators=100),
    # "XGBoost":
    "Naive Bayes":dict(var_smoothing=1e-9),
    "Perceptron":dict(alpha=0.001, penalty='l1'),
    "Logistic Regression": dict(C=1, penalty='l2'),
    "LDA":dict(),
    "SVM":dict(kernel='rbf', C=1)
}
```

2.3

Decision Tree

The best configuration of hyperparameters is `criterion='gini'` and `max_depth=5`.

criterion	entropy	entropy	entropy	gini	gini	gini	log_loss	log_loss	log_loss
max_depth	I	B	U	I	B	U	I	B	U
3	0.933	0.895	0.923	0.933	0.895	0.944	0.933	0.912	0.937
5	0.933	0.912	0.909	0.933	0.947	0.923	0.933	0.912	0.93
10	0.933	0.912	0.923	0.933	0.912	0.937	0.933	0.912	0.93
12	0.933	0.895	0.944	0.933	0.912	0.937	0.933	0.912	0.937

Random Forest

criterion	entropy	entropy	entropy	gini	gini	gini	log_loss	log_loss	log_loss
n_estimators	I	B	U	I	B	U	I	B	U
20	1.0	0.93	0.972	1.0	0.947	0.986	0.933	0.912	0.972
50	1.0	0.877	0.965	1.0	0.877	0.979	0.933	0.877	0.979
100	1.0	0.93	0.965	0.933	0.912	0.972	0.933	0.912	0.972
200	1.0	0.947	0.986	0.933	0.912	0.972	0.933	0.912	0.979

The best configuration of hyperparameters is `criterion='entropy'` and `n_estimators=50`.

Bagging

n_estimators/dataset	I	B	U
2	0.867	0.982	0.951
5	0.867	1.0	0.944
10	0.933	1.0	0.951
20	0.933	0.982	0.986

The best configuration of hyperparameters is `n_estimators=20`.

Gradient Boosting

The best configuration of hyperparameters is `learning_rate=0.1` and `n_estimators=100`.

Naive Bayes

var_smoothing/dataset	I	B	U
1e-09	1.0	0.982	0.93
1e-07	1.0	0.982	0.93
0.001	1.0	0.982	0.93
0.1	1.0	0.965	0.937

The best configuration of hyperparameters is `var_smoothing=1e-9`.

learning_rate	1e-3	1e-3	1e-3	1e-2	1e-2	1e-2	1e-1	1e-1	1e-1
n_estimators	I	B	U	I	B	U	I	B	U
10	1.0	0.632	0.909	0.933	0.632	0.909	1.0	0.632	0.93
20	0.933	0.632	0.93	0.933	0.632	0.909	0.933	0.965	0.909
50	0.933	0.947	0.937	0.933	0.965	0.923	1.0	0.947	0.951
100	0.933	0.632	0.909	1.0	0.632	0.93	1.0	0.632	0.93
200	0.933	0.632	0.909	0.933	0.965	0.909	1.0	0.965	0.937

Perceptron

penalty	1e-3	1e-3	1e-3	1e-2	1e-2	1e-2	1e-1	1e-1	1e-1
alpha	I	B	U	I	B	U	I	B	U
1e-05	0.867	0.965	0.993	0.867	0.965	0.993	0.867	0.965	0.993
0.0001	0.867	0.947	0.993	0.867	0.912	0.986	0.867	0.982	0.986
0.001	0.867	0.947	0.993	0.867	0.965	0.986	0.867	0.947	0.937
0.01	0.867	0.965	0.993	0.867	0.965	0.993	0.867	0.965	0.993
0.1	0.867	0.912	0.986	0.867	0.982	0.986	0.8	0.93	0.937

The best configuration of hyperparameters is `alpha=0.001` and `penalty='l1'`.

Logistic Regression

The model solver 'lbfgs' can not handle 'l1' penalty, so I just include 'None' and 'l2' in penalty options.

penalty	None	None	None	l2	l2	l2
C	I	B	U	I	B	U
0.1	1.0	0.982	0.965	1.0	0.982	0.986
1	0.867	0.982	0.986	1.0	0.982	0.965
10	1.0	0.982	0.965	1.0	0.982	0.979

The best configuration of hyperparameters is `C=1` and `penalty='l2'`.

LDA

dataset	I	B	U
validation accuracy	1.0	0.930	0.993

The average of the validation accuracy over the three datasets is 0.978.

SVM

kernel	linear	linear	linear	poly	poly	poly	rbf	rbf	rbf	sigmoid	sigmoid	sigmoid
C	I	B	U	I	B	U	I	B	U	I	B	U
0.001	0.933	0.947	0.993	0.933	0.632	0.503	0.933	0.632	0.895	0.933	0.877	0.944
0.01	0.533	0.684	0.503	0.933	0.965	0.993	0.933	0.842	0.965	0.933	0.965	0.979
0.1	0.933	0.632	0.503	0.6	0.772	0.671	1.0	0.965	1.0	0.933	0.965	0.986
1	0.933	0.632	0.503	0.933	0.632	0.895	0.933	0.877	0.944	1.0	0.982	0.986
10	0.933	0.965	0.993	0.933	0.842	0.965	0.933	0.965	0.979	0.933	0.895	1.0
100	0.6	0.772	0.671	1.0	0.965	1.0	0.933	0.965	0.986	1.0	0.982	1.0
1000	0.933	0.632	0.895	0.933	0.877	0.944	1.0	0.982	0.986	0.933	0.965	0.951

The best configuration of hyperparameters is `kernel='rbf'` and `C=1`.

2.4

The optimal hyperparameter configuration is shown in 2.2, and the accuracy on the test set for all datasets is shown in Fig.1.

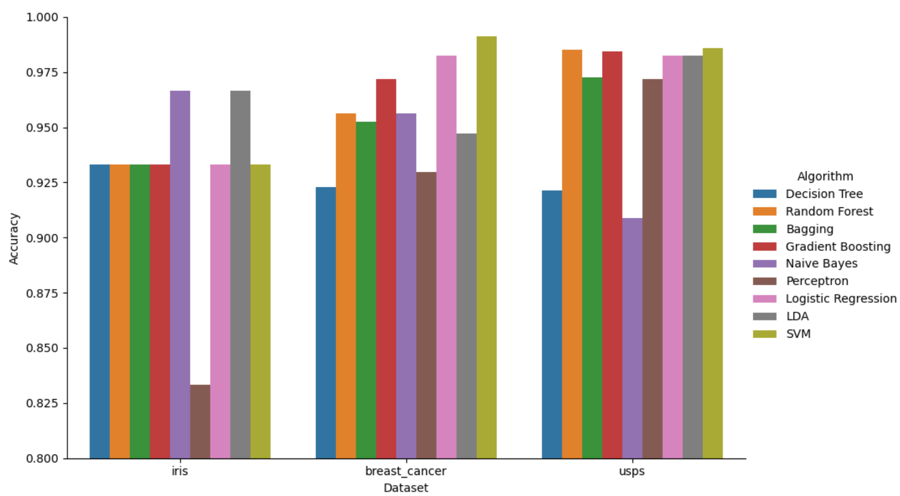

Figure 1: The accuracy on the test dataset for each model