02-750

Assignment 3

Team:
Zeyuan Zuo (zeyuanz)
Tianqin Li (tianqinl)

HW3q1

April 6, 2021

[1]:
```
import numpy as np
import random
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.utils import shuffle

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.gaussian_process.kernels import Matern,RBF,WhiteKernel,DotProduct
from sklearn.metrics import r2_score
from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning

from modAL.uncertainty import uncertainty_sampling
from modAL.models import BayesianOptimizer, ActiveLearner, CommitteeRegressor
from modAL.acquisition import max_EI
from modAL.disagreement import max_std_sampling, max_disagreement_sampling

import seqlogo

import copy

### Set random seed
seed = 5
random.seed(seed)
np.random.seed(seed)
```

[2]:
```
data = np.loadtxt('hw3_data.csv', dtype = str, delimiter = ',')[1:]
peptide = data[:,2]

def create_ohe_dictionary(peptide):
    ohe_dict = {}
    encoding = 0
    for i in range(len(peptide)):
        for j in range(len(peptide[i])):
```

```
            if peptide[i][j] not in ohe_dict.keys():
                ohe_dict[peptide[i][j]] = encoding
                encoding += 1
    return ohe_dict

ohe_dict = create_ohe_dictionary(peptide)

def ohe_row(peptide_string, ohe_dict):
    idx = 0
    row = np.zeros(shape=9*len(ohe_dict))
    for aa in peptide_string:
        row[idx + ohe_dict[aa]] = 1
        idx += len(ohe_dict)
    return row

def one_hot_encoding(peptide, ohe_dict):
    ohe_encoding_peptide = np.zeros(shape=(len(peptide), 9 * len(ohe_dict)))
    for i in range(len(peptide)):
        ohe_encoding_peptide[i] = ohe_row(peptide[i], ohe_dict)
    return ohe_encoding_peptide

X = one_hot_encoding(peptide, ohe_dict)
y = data[:,3].astype('float64')
```

[6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

[7]:
```
# RandomForestRegressor with random query
X_train_random = copy.deepcopy(X_train)
y_train_random = copy.deepcopy(y_train)

regressor = ActiveLearner(
    estimator=RandomForestRegressor(),
    query_strategy=uncertainty_sampling,
    X_training=X_train_random[0].reshape(1, -1),
    y_training=y_train_random[np.array(0)].reshape(1),
)
n_queries = 2000
for idx in range(n_queries):
    query_idx = np.random.randint(len(X_train_random))
    regressor.teach(X_train_random[query_idx].reshape(1, -1), y_train_random[np.
 ↪array(query_idx)].reshape(1))
    X_train_random, y_train_random = (np.delete(X_train_random, query_idx,␣
 ↪axis=0), np.delete(y_train_random, query_idx))

    if idx % 100 == 0:
        y_pred_final = regressor.predict(X_test)
        y_train_pred = regressor.predict(X_train)
```

```
        r2_test = r2_score(y_test, y_pred_final)
        r2_train = r2_score(y_train, y_train_pred)
        print("[%d/%d]\tR2 train:\t%.4f\tR2 test:\t%.4f" %(idx, n_queries,␣
 ↪r2_train, r2_test))

y_pred_final = regressor.predict(X_test)
y_train_pred = regressor.predict(X_train)
r2_test = r2_score(y_test,y_pred_final)
r2_train = r2_score(y_train,y_train_pred)

# print(y_test,y_pred_final,y_train_pred)
print("R2 train:\t %.4f" %(r2_train))
print("R2 test:\t %.4f" %(r2_test))
```

```
[0/2000]        R2 train:       -0.8543 R2 test:        -0.8299
[100/2000]      R2 train:       0.2774  R2 test:        0.2764
[200/2000]      R2 train:       0.4214  R2 test:        0.4013
[300/2000]      R2 train:       0.4382  R2 test:        0.4211
[400/2000]      R2 train:       0.4588  R2 test:        0.4219
[500/2000]      R2 train:       0.4977  R2 test:        0.4487
[600/2000]      R2 train:       0.5131  R2 test:        0.4607
[700/2000]      R2 train:       0.5428  R2 test:        0.4846
[800/2000]      R2 train:       0.5514  R2 test:        0.4857
[900/2000]      R2 train:       0.5723  R2 test:        0.5055
[1000/2000]     R2 train:       0.5999  R2 test:        0.5242
[1100/2000]     R2 train:       0.6053  R2 test:        0.5190
[1200/2000]     R2 train:       0.6048  R2 test:        0.5210
[1300/2000]     R2 train:       0.6174  R2 test:        0.5301
[1400/2000]     R2 train:       0.6215  R2 test:        0.5293
[1500/2000]     R2 train:       0.6275  R2 test:        0.5224
[1600/2000]     R2 train:       0.6378  R2 test:        0.5215
[1700/2000]     R2 train:       0.6502  R2 test:        0.5275
[1800/2000]     R2 train:       0.6631  R2 test:        0.5336
[1900/2000]     R2 train:       0.6703  R2 test:        0.5337
R2 train:        0.6786
R2 test:         0.5397
```

[8]:
```
simplefilter("ignore", category=ConvergenceWarning)
# activeGaussianProcessRegressor with active sampling
X_train_active = X_train.copy()
y_train_active = y_train.copy()

kernel = DotProduct(sigma_0=1.5)+WhiteKernel()

learners = [
    ActiveLearner(
        estimator=GaussianProcessRegressor(kernel = kernel),
```

```
        X_training=X_train_active[idx].reshape(1, -1),
        y_training=y_train_active[np.array(idx)].reshape(1),
    ) for idx in range(4)
]
committee = CommitteeRegressor(
    learner_list = learners,
    query_strategy = max_std_sampling,
)

n_queries = 2000
for idx in range(n_queries):
    query_idx,_ = committee.query(X_train_active)
    committee.teach(X_train_active[query_idx].reshape(1, -1), y_train_active[np.
 ↪array(query_idx)].reshape(1))
    X_train_active, y_train_active = (np.delete(X_train_active, query_idx,␣
 ↪axis=0), np.delete(y_train_active, query_idx))

    if idx % 100 == 0:
        y_pred_final = committee.predict(X_test, return_std = False)
        y_train_pred = committee.predict(X_train, return_std = False)
        r2_test = r2_score(y_test, y_pred_final)
        r2_train = r2_score(y_train, y_train_pred)
        print("[%d/%d]\tR2 train:\t%.4f\tR2 test:\t%.4f" %(idx, n_queries,␣
 ↪r2_train, r2_test))

y_pred_final = committee.predict(X_test, return_std = False)
y_train_pred = committee.predict(X_train, return_std = False)
r2_test = r2_score(y_test, y_pred_final)
r2_train = r2_score(y_train, y_train_pred)

# print(y_test,y_pred_final,y_train_pred)
print("R2 train:\t %.4f" %(r2_train))
print("R2 test:\t %.4f" %(r2_test))
```

```
[0/2000]        R2 train:       -1.6069 R2 test:        -1.5795
[100/2000]      R2 train:       0.1704  R2 test:        0.1232
[200/2000]      R2 train:       0.1088  R2 test:        0.0704
[300/2000]      R2 train:       0.3947  R2 test:        0.3574
[400/2000]      R2 train:       0.4907  R2 test:        0.4531
[500/2000]      R2 train:       0.5065  R2 test:        0.4732
[600/2000]      R2 train:       0.5356  R2 test:        0.5005
[700/2000]      R2 train:       0.5474  R2 test:        0.5137
[800/2000]      R2 train:       0.5790  R2 test:        0.5485
[900/2000]      R2 train:       0.5865  R2 test:        0.5541
[1000/2000]     R2 train:       0.5934  R2 test:        0.5636
[1100/2000]     R2 train:       0.6058  R2 test:        0.5766
[1200/2000]     R2 train:       0.6131  R2 test:        0.5811
```

```
[1300/2000]     R2 train:       0.6164  R2 test:        0.5841
[1400/2000]     R2 train:       0.6243  R2 test:        0.5906
[1500/2000]     R2 train:       0.6280  R2 test:        0.5961
[1600/2000]     R2 train:       0.6325  R2 test:        0.6007
[1700/2000]     R2 train:       0.6329  R2 test:        0.5994
[1800/2000]     R2 train:       0.6367  R2 test:        0.6031
[1900/2000]     R2 train:       0.6390  R2 test:        0.6038
R2 train:        0.6404
R2 test:         0.6048
```

0.1 Questions:

1. Done (see code)
2. Done (see code)
3. We use committee regressor. It contains three guassian process regressor using a combination of DotProduct and WhiteKernel. Each regressor intializes with different training samples.
4. Active Learner: $R^2 = 0.6048$. Number of n_queries = 2000
5. Random Forest regressor with random query: $R^2 = 0.5337$. Number of n_queries = 2000
6. Comparison: With same amount of training data (2000 queries), committee regressor achieves higher R^2 score than random forest regressor with random query. Random forest regressor has a higher R^2 score than committee regressor. However, when number of queries increase, committee regressor outperforms random forest regressor.

Within the first 1000 queries, random forest regressor achieves $R^2 = 0.5999$ in training set and $R^2 = 0.5242$ in test set. However, the final $R^2 = 0.6703$ in training set and $R^2 = 0.5337$ in test set. It implies that in 1000-2000 queries, random forest regressor begins the overfit the training data and there is no large improvement of R^2 in test set.

Within the first 1000 queries, committee regressor achieves $R^2 = 0.5934$ in training set and $R^2 = 0.5636$ in test set. It behaves better than random forest regressor in test set. It finally achieves is $R^2 = 0.6390$ in training set and $R^2 = 0.6038$ in test set. We notice that compared to random forest regressor, committee regressor has a higher R^2 in test set and a lower R^2 in training set. During queries 1000-2000, committee regressor continues to achieves higher R^2 in test set which means that it does learn something.

Therefore, from comparison above, committee regressor with active learning achieves higher R^2 in the same amount of queries than random forest regressor with random query. It also reduces overfitting of the model.

[ ]:

HW3q2

April 6, 2021

1 Question 2 (50 points)

In this question, you will simulate a peptide design experiment, trying to find peptides with high binding affinity to MHC class I using a bayesian optimization approach. Notice the goal here is not trying to find a peptide sequence that maximize the binding affinity to MHC, Since a sizable proportion of the sequence data we are using contains maximum binding affinity out of the data (9.0). Using the same feature encoding as question 1, we will examine several techniques to maximize the percentage of sequence with affinity of 9.0 for stringent querying.

[1]:
```
import numpy as np
import random
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.utils import shuffle

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import Matern
from sklearn.ensemble import RandomForestRegressor

from modAL.models import BayesianOptimizer
from modAL.acquisition import max_EI

import seqlogo

import copy

### Set randome seed
seed = 5
random.seed(seed)
np.random.seed(seed)
```

2 Reading and Processing the Data¶

[2]:
```
data = np.loadtxt('hw3_data.csv', dtype = str, delimiter = ',')[1:]
### TO DO
peptide = data[:,2]
encode_order = 'ACDEFGHIKLMNPQRSTVWY'
def create_ohe_dictionary(encode_order):
    ohe_dict = {}
    encoding = 0
    for i in range(len(encode_order)):
        ohe_dict[encode_order[i]] = encoding
        encoding += 1
    return ohe_dict

ohe_dict = create_ohe_dictionary(encode_order)

def ohe_row(peptide_string, ohe_dict):
    idx = 0
    row = np.zeros(shape=9*len(ohe_dict))
    for aa in peptide_string:
        row[idx + ohe_dict[aa]] = 1
        idx += len(ohe_dict)
    return row

def one_hot_encoding(peptide, ohe_dict):
    ohe_encoding_peptide = np.zeros(shape=(len(peptide), 9 * len(ohe_dict)))
    for i in range(len(peptide)):
        ohe_encoding_peptide[i] = ohe_row(peptide[i], ohe_dict)
    return ohe_encoding_peptide

X = one_hot_encoding(peptide, ohe_dict)
y = data[:,3].astype('float64')
```

3 2.1: Random Sampling (5 pts. total)

Create a random query strategy for randomly selecting a sample to query from the data. If the data selected is a new sequence with binding affnity of 9.0, append it to a list. After each query selection, measure the percentage of sequence with binding affnity 9.0 found by the strategy. Do this for 200 sampling steps. This will serve as the baseline to compare with optimizator performance in section 2.2 and 2.3.

[3]:
```
X_cp = copy.deepcopy(X)
y_cp = copy.deepcopy(y)
optimal_idx_rand = []
history_rand = []
cnt = 0
### TO DO
```

```
n_query = 200
for i in range(n_query):
    idx = np.random.randint(0, len(y_cp))
    if y_cp[idx] == 9.0:
        cnt += 1
        optimal_idx_rand.append(X_cp[idx])
    history_rand.append(cnt / (i+1))
    X_cp, y_cp = np.delete(X_cp, idx, axis=0), np.delete(y_cp, idx)
```

4 2.2: Baysian Optimization with Gaussian Process (15 pts. total)

Create a Baysian optimizer with Gaussian process as regressor and Max Expected improvement as the queuing strategy. If the data selected is a new sequence with binding affinity of 9.0, append it to a list. After each query selection, measure the percentage of sequence with binding affinity 9.0 found by the strategy. Do this for 200 sampling steps.

Hint: Check the modAL documentation for how to set up a Baysian optimizer.

[4]:
```
X_cp = copy.deepcopy(X)
y_cp = copy.deepcopy(y)

optimal_idx_gp = []
history_gp = []
arr = np.arange(len(X_cp))
np.random.shuffle(arr)
### TO DO
# initializing the optimizer
optimizer = BayesianOptimizer(
    estimator=GaussianProcessRegressor(),
    X_training=X_cp[arr[:10]],
    y_training=y_cp[arr[:10]],
    query_strategy=max_EI
)
# Bayesian optimization
cnt = 0

for i in range(n_query):
    query_idx, query_inst = optimizer.query(X_cp)
    optimizer.teach(X_cp[query_idx].reshape(1, -1), y_cp[np.array(query_idx)].
 ↪reshape(1))
    if y_cp[query_idx] == 9.0:
        cnt += 1
        optimal_idx_gp.append(X_cp[query_idx])
    history_gp.append(cnt / (i+1))
    X_cp, y_cp = np.delete(X_cp, query_idx, axis=0), np.delete(y_cp, query_idx)
```

5 2.3: Bayesian Optimizer with Random Forest (10 pts. total)

Although Baysian optimization often uses the Gaussian process, Baysian optimizer in ModAL can take any other regressor that has a predict function with a return_std input parameter. If return_std is set to True, the function returns the predicted values and standard deviation in the prediction. Create a Baysian optimizer with random forest regressor and Max Expected improvement as the queuing strategy. If the data selected is a new sequence with binding affinity of 9.0, append it to a list. After each query selection, measure the percentage of sequences with binding affinity 9.0 found by the strategy. Do this for 200 sampling steps.

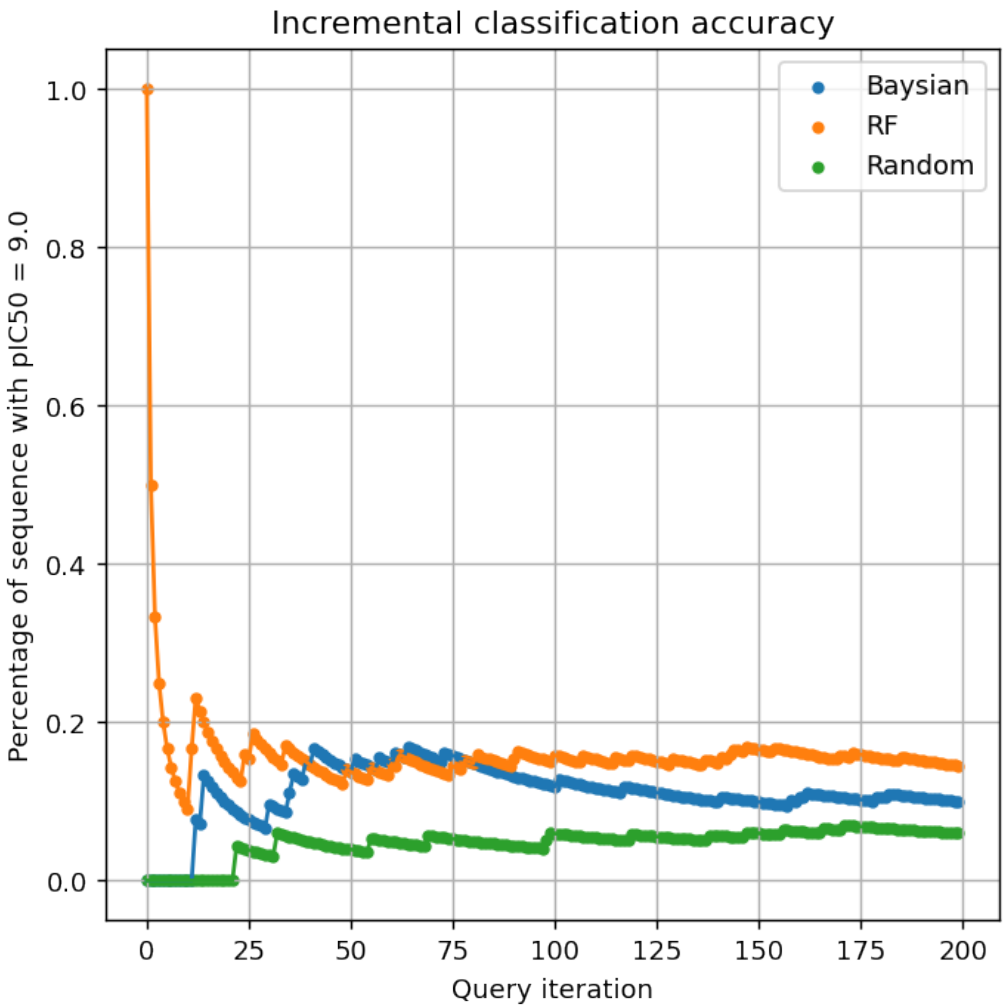
Hint: You might find the following class wrapper for random forest helpful.

[5]:
```
class rfwapper(RandomForestRegressor):
    def predict(self, X, return_std = False):
        if return_std:
            ys = np.array([e.predict(X) for e in self.estimators_])
            return np.mean(ys, axis = 0).ravel(), (np.std(ys, axis = 0).ravel()␣
 ↪+ 1e-6)
        return super().predict(X).ravel()
```

[6]:
```
X_cp = copy.deepcopy(X)
y_cp = copy.deepcopy(y)

optimal_idx_rf = []
history_rf = []
arr = np.arange(len(X_cp))
np.random.shuffle(arr)

### TO DO
# initializing the optimizer
optimizer = BayesianOptimizer(
    estimator=rfwapper(),
    X_training=X_cp[arr[:10]],
    y_training=y_cp[arr[:10]],
    query_strategy=max_EI
)
# Bayesian optimization
cnt = 0
n_query = 200
for i in range(n_query):
    query_idx, query_inst = optimizer.query(X_cp)
    optimizer.teach(X_cp[query_idx].reshape(1, -1), y_cp[np.array(query_idx)].
 ↪reshape(1))
    if y_cp[query_idx] == 9.0:
        cnt += 1
        optimal_idx_rf.append(X_cp[query_idx])
    history_rf.append(cnt / (i+1))
    X_cp, y_cp = np.delete(X_cp, query_idx, axis=0), np.delete(y_cp, query_idx)
```

6 2.4: Plot Percentage of sequence with maximum binding affinity with respect to number of sequence queried (10 pts. total)

[7]:
```
# Plot our performance over time.
fig, ax = plt.subplots(figsize=(6, 6), dpi=130)

ax.plot(history_gp)
ax.scatter(range(len(history_gp)), history_gp, s=13, label = 'Baysian')


ax.plot(history_rf)
ax.scatter(range(len(history_rf)), history_rf, s=13, label = 'RF')


ax.plot(history_rand)
ax.scatter(range(len(history_rand)), history_rand, s=13, label = 'Random')

ax.grid(True)

ax.set_title('Incremental classification accuracy')
ax.set_xlabel('Query iteration')
ax.set_ylabel('Percentage of sequence with pIC50 = 9.0')
ax.legend()
plt.show()
```



7 2.5: Create sequence logo based on sequences found with each querying strategies (5 pts. total)

A sequence logo is a graphical representation of the sequence conservation of amino acids in protein sequences), as amino acids that are important for functions are likely to be conserved. Hence, a sequence logo is a way to visualize such an importance. Convert the each sets of sequences obtained by one of your optimization strategies to a sequence logo. Below is an example using all of the sequence of affinity 9.0.

Important: We are using seqlogo to create sequence logo from our set of sequences. You can install seqlogo by entering the command

conda install -c bioconda seqlogo

in your conda terminal

[10]:
```
X_rf = optimal_idx_rf
ppm = np.sum(X_rf, axis = 0).reshape(20,9)
ppm /= np.sum(ppm, axis = 0)
ppm = seqlogo.Ppm(ppm, alphabet_type="AA")
seqlogo.seqlogo(ppm, ic_scale = False, format = 'jpeg', size = 'medium')
```

[10]: