INF367A Exercise 7

Naphat Amundsen

February 29, 2020

Introduction

This exercise is about doing Bayesian model selection.

1 Task

We want to use linear models. But the data are not always linear and thus we may need to use basis functions to handle non-linearity. However, it is not clear which basis functions are good for any particular data. Furthermore, selection of the noise precision β is crucial for to get good performance.

In this task, we model (again) bicycling in New York. You should download the data set new_york_bicycles3.csv from MittUiB. The goal is to predict the highest daily temperature given the other variables. In this exercise, we select a set of basis functions and the value of β. To this end, you should try different basis functions and different values of β. In other words, a model is specified by the basis functions and beta, i.e., $M_i = (\phi_i, \beta_i)$. Note: If the selected β is either smallest or the largest value that you tried, then you should probably try more values. Use the following model selection strategies:

1. Use the posterior ratio to select a model. To do this, you need to compute marginal likelihoods. (Note: you want to compare $P(D|M_i)P(M_i)$ values of different models.)
2. Use BIC to select a model. (Note: you want to maximize BIC.)
3. Use AIC to select a model. (Note: you want to minimize AIC.)
4. Use cross-validation to select a model. (Note: you want to minimize the validation loss.)

Hints: Typically, one projects data to higher dimensions. However, it may be useful to consider projecting data also to a lower dimensional space (i.e. not to use all features).

Remember that when you compute marginal likelihood, you should use the predictive distribution $P(y|\hat{f}_\theta(x)) = N(y|m^T\phi(x), 1/\beta + \phi(x)^T S\phi(x))$. In case of AIC, BIC, and the cross-validation loss, you should use the the usual likelihood $P(y|w, x, \beta) = N(y|w^T\phi(x), 1/\beta)$.

Remember that the MAP estimate of a Gaussian distribution is its mean.

Scipy's Gaussian distribution, norm has a scale parameter that takes in a standard deviation, i.e., p $\sqrt{1/precision}$.

If you get totally insensible results, first sanity check is to check whether you are minimizing when you should be maximizing or vice versa.

Methods:

Disclaimer: I am just writing what I think here in case I forget in the future, scroll down to find actual answers.

Posterior ratio and BIC

Posterior ratio in this context is simply comparing the posteriors to all the models, i.e $P(D|M_i)P(M_i)/P(D)$, but is since $P(D)$ is a common factor for all, we can ignore it, thus needing only to compare $P(D|M_i)P(M_i)$. The models have parameters θ (in this case coefficients of the basis functions and the intercept), so we have to marginalize away θ in order to obtain $P(D|M_i)$:

$$P(D|M_i) = \int P(D|\theta, M_i)P(\theta|M_i)d\theta \tag{1}$$

when $n \to \infty$.
The integral is not in a nice form, so it is approximated using Laplace approximation, which results in:

$$\log P(D|M_i) \approx \sum_{i=1}^{n} \log P(D_i|\theta, M_i) + \log P(\theta|M_i) + \frac{d}{2}\log 2\pi - \frac{d}{2}\log n - \frac{1}{2}|F(\hat{\theta})| \tag{2}$$

Then for large values of n (which we already assume), several terms converge to constants, resulting with

$$\log P(D|\hat{\theta}, M) - \frac{d}{2}\log n \tag{3}$$

where $\hat{\theta} = \arg\max_\theta P(D|\theta, M)P(\theta|M)$.
All of which we will define as $BIC(M) = \log P(D|\hat{\theta}, M) - \frac{d}{2}\log n$

In order to find $\hat{\theta}$ we can simply use the conjugate prior linear regression weights assuming that the prior weights are normally distributed as $\theta \sim \mathcal{N}(0, \alpha^{-1}I)$:

$$S = A^{-1} = \left(\alpha I + \beta \sum_{i=1}^{n} x_i x_i^T\right)^{-1} \tag{4}$$

$$m = A^{-1}b = \beta S \sum_{i=1}^{n} y_i x_i \tag{5}$$

$$\Rightarrow P(w|y, x, \alpha, \beta) = \mathcal{N}(w|m, S) \tag{6}$$

where β is the error precision ($\epsilon_i \sim \mathcal{N}(0, \beta^{-1})$). Since the normal distribution peaks at its mean, $\hat{\theta}$ is simply the mean m.

AIC

Pure instrumentally, AIC is very similar to BIC (although the fundamental ideas are different). AIC is defined as

$$AIC = -\frac{2}{n} \log P(y|\hat{\theta}, X) + 2 \cdot \frac{d}{n} \tag{7}$$

Note that likelihood here is using a different notation than BIC.

Cross Validation

Just regular cross validation. The cross validation performed in this task is with 5 folds. The hyperparameter in interest is the degree of the polynomial basis function.

2 Actual answers

I assume that $P(M_i) \sim$ Uniform. In the case of posterior ratio, the assumption effectively removes the prior ratio as they will always be 1:

$$\frac{P(M_i|D)}{P(M_j|D)} = \frac{P(D|M_i)}{P(D|M_j)} \times \frac{P(M_i)}{P(M_j)} \tag{8}$$

$$\Rightarrow \frac{P(M_i|D)}{P(M_j|D)} = \frac{P(D|M_i)}{P(D|M_j)} \tag{9}$$

Effectively, we are comparing which posteriors have the highest values, so I will just do that instead of finding the ratios between all combinations of things. Why make it difficult you know?

The candidate models are polynomial models of degree 1 to 8. The results are as follows:

Table 1: Model selection for polynomial fit. AIC and BIC uses $\hat{\theta}$ that was calculated the Bayesian way (conjugate prior). The Cross Validation calculates $\hat{\theta}$ using whatever Sklearn uses. The performance metric for CV is the negative mean absolute error (same error as AIC and BIC, but negative). Posterior: higher is better, BIC: higher is better, AIC: lower is better, CV higher is better. Best values with respect to method is bolded. **Note:** Since we assumed uniform prior for models, comparing posterior ratios is equivalent to comparing BIC values

degree	LogLhoods	BIC	AIC	CV_val	CV_train
1	-3560.76	-3576.86	33.33	**-8.40**	-6.89
2	-3499.99	-3556.34	32.91	-8.75	-6.13
3	-3135.87	-3286.11	29.83	-16.13	-5.00
4	**-2048.55**	**-2386.60**	**20.32**	-148.72	-6.28
5	-5848.31	-6524.42	57.01	-5583.39	-3.47
6	-5913.03	-7152.57	59.58	-26028.28	-0.23
7	-5913.04	-8037.97	62.66	-161911.32	**-0.02**
8	-5889.99	-9343.00	67.07	-1182049.76	-0.06

Posterior, BIC, AIC seem to agree that a polynomial kernel of degree 4 is the best. Cross validation says that we should have degree 1, and for some weird reason the mean training error is best at 7 degrees and not 8.

Code

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from scipy.stats import norm
from common import texmatrix
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model.base import BaseEstimator
import sklearn.metrics as metrics

def get_data():
    pd.set_option('display.expand_frame_repr', False)
    df = pd.read_csv('new_york_bicycles3.csv')

    X = df.values[:,1:]
    y = df.values[:,0]
    return X, y

def poly_basis(degree: int=2):
    def kernelized(X):
        return PolynomialFeatures(degree, include_bias=True).fit_transform
                                                    (X)
```

```
    return kernelized

def get_thetas(Xs):
    '''
    Calculate thetas (using conjugate prior)
    '''
    thetas = []
    for X_ in Xs:
        d = X_.shape[1]

        A = alpha*np.eye(d) + beta * X_.T@X_
        S = np.linalg.inv(A)
        theta = m = S@(beta * (X_ * y_)).sum(axis=0) # This is theta hat
        thetas.append(theta)
    return thetas

def get_loglikelihoods(Xs, thetas, beta=1):
    '''
    Bayesian way
    '''
    lhoods = []
    for theta, X_ in zip(thetas, Xs):
        preds = theta@X_.T
        logL = np.log(norm.pdf(preds-y, 0, np.sqrt(1/beta))+1e-12).sum()
        lhoods.append(logL)
    return lhoods

class LinReg(BaseEstimator):
    '''
    Sklearn compitable Linear Regression that
    fits parameters using bayesian way (not used)
    '''
    def __init__(self, beta=1):
        self.beta = beta

    def fit(self, X, y, *args, **kwargs):
        d = X.shape[1]

        A = alpha*np.eye(d) + self.beta * X.T@X
        S = np.linalg.inv(A)
        theta = m = S@(self.beta * (X * y.reshape(-1,1))).sum(axis=0) #
                                                    This is theta hat
        self.theta = theta

    def predict(self, X):
        return self.theta@X.T

if __name__ == '__main__':
    X, y = get_data()
    y_ = y.reshape(-1,1)

    alpha = 1
    beta = 1
```

```
n = X.shape[0]

phis = [poly_basis(i) for i in range(1, 9)]
Xs = [phi(X) for phi in phis]
thetas = get_thetas(Xs)
loglikelihoods = get_loglikelihoods(Xs, thetas, beta=beta)

def get_bics(Xs, thetas, lhoods):
    BICs = []
    for X_, theta, logL in zip(Xs, thetas, lhoods):
        d = X_.shape[1]
        other_term = d/2 * np.log(n)
        BICs.append(logL - other_term)
    return BICs

def get_aics(Xs, thetas, lhoods):
    AICs = []
    for X_, theta, logL in zip(Xs, thetas, lhoods):
        AICs.append(-2/n * logL + 2*X_.shape[1]/n)
    return AICs

def get_cvs(Xs, thetas, lhoods):
    mean_val_scores = []
    mean_train_scores = []
    for X_, theta in zip(Xs,thetas):
        gsc = GridSearchCV(LinearRegression(fit_intercept=False),
                                            param_grid={}, scoring='
                                            neg_mean_absolute_error',
                                             cv=5, iid=True,
                                            return_train_score=True)
        gsc.fit(X_, y)
        mean_val_scores.append(gsc.cv_results_['mean_test_score'][0])
        mean_train_scores.append(gsc.cv_results_['mean_train_score'][0
                                            ])
    return mean_val_scores, mean_train_scores

bics = get_bics(Xs, thetas, loglikelihoods)
aics = get_aics(Xs, thetas, loglikelihoods)
cv_vals, cv_trains = get_cvs(Xs, thetas, loglikelihoods)

df = pd.DataFrame(dict(
    degree=np.arange(1,9),
    LogLhoods=loglikelihoods,
    BIC=get_bics(Xs, thetas, loglikelihoods),
    AIC=get_aics(Xs, thetas, loglikelihoods),
    CV_val=cv_vals,
    CV_train=cv_trains
))

pd.options.display.float_format = '{:.2f}'.format
print(df.to_latex(index=False))
```