3F8: Inference
Full Technical Report

Danut-Gabriel Buica

March 27, 2020

Abstract

Dealing with high dimensions, we can easily linearly separate our data. This may lead to overfitting, therefore, we want to further improve our Logistic Classifier by applying Laplace approximation on our weigths. This will prevent the classification to be made with too high certainty while still minimizing the misclassification. we will also investigate the performance of selecting hyper-parameter values by optimizing the model evidence. By experimated with different parameters, we determined that using a trade-off between minimizing the data misfit and model complexity, we can achive far better prediction rates.

1 Introduction

1. During our coursework we implemented a Logistic Classifier. The problem is that in higher dimmension our data can be liniarly separated by many set of weights and in some cases this may lead to overfitting.
2. To solve this problem we will implement a Bayesian binary classifier that applies the Laplace approximation to our Logistic Classification model. Furthermore, we will investigate the performance of selecting hyper parameters values by optmizing over the model evidence.

2 Exercise a)

The goal of the Laplace approximation is to fit a Gaussian approximation to a probability density defined over a set of continuous variables. First, we will consider the case of a single continuous variable z with the probability distribution $p(z)$ so that:

$$p(z) = \frac{1}{Z} f(z) \tag{1}$$

where $Z = \int f(z)dz$ is the unknown normalization constant. Let $q(z) = \mathcal{N}(z|m, v)$ denote our Gaussian approximation. We want our Gaussian to be centered on a mode of $p(z)$. Therefore, m is the MAP solution of $p(z)$:

$$\left.\frac{df(z)}{dz}\right|_{z=m} = 0 \tag{2}$$

As for finding the value of v, the property of a Gaussian distribution that its logarithm is a quadratic function of the variables proves to be very useful. By considering the Taylor expansion of $lnf(z)$ around m, we get:

$$\ln f(z) \approx \ln f(m) - \frac{1}{2} A(z-m)^2 \tag{3}$$

where

$$A = -\left.\frac{d^2}{dz^2} \ln f(z)\right|_{z=m} \tag{4}$$

Notice that the first order of the Taylor expnasion is 0 since m is the local maximum of the distribution. By taking the exponential of eq. 3 we obtain:

$$f(z) \approx f(m) \exp\left(-\frac{A}{2}(z-m)^2\right) = f(m)\left(\frac{A}{2\pi}\right)^{-1/2} \mathcal{N}(z|m, A^{-1}) \tag{5}$$

Thus, we get the following forms of $q(z) = \mathcal{N}(z|m, A^{-1})$ and $Z = f(m)\left(\frac{A}{2\pi}\right)^{-1/2}$. We should note that this is well defined when $A > 0$, in other words m represents a local maximum.

Furthermore, we can extend the Laplace approximation to a distribution $p(\mathbf{z}) = \frac{1}{Z} f(\mathbf{z})$ defined over an M-dimensional space $\mathbf{z}$. Our choice for the mean of the Gaussian will again the MAP solution $\mathbf{z}_{MAP}$ with the property that:

$$\nabla f(\mathbf{z})|_{\mathbf{z}=\mathbf{z}_{MAP}} = 0 \tag{6}$$

Hence, expanding $\ln f(\mathbf{z})$ around $\mathbf{z}_{MAP}$ gives:

$$\ln f(\mathbf{z}) \approx \ln f(\mathbf{z}_{MAP}) - \frac{1}{2}(\mathbf{z}-\mathbf{z}_{MAP})^T \mathbf{A} (\mathbf{z}-\mathbf{z}_{MAP}) \tag{7}$$

where $\mathbf{A}$ is an $M \times M$ Hessian matrix defined as:

$$\mathbf{A} = -\nabla\nabla \ln f(\mathbf{z})|_{\mathbf{z}=\mathbf{z}_{MAP}} \tag{8}$$

By taking the exponential of eq. 7 we obtain:

$$f(\mathbf{z}) \approx f(\mathbf{z}_{MAP}) \exp\left(-\frac{1}{2}(\mathbf{z}-\mathbf{z}_{MAP})^T \mathbf{A} (\mathbf{z}-\mathbf{z}_{MAP})\right) = f(\mathbf{z}_{MAP}) \frac{(2\pi)^{M/2}}{|\mathbf{A}|^{1/2}} \mathcal{N}(\mathbf{z}|\mathbf{z}_{MAP}, \mathbf{A}^{-1}) \tag{9}$$

This gives us $q(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{z}_{MAP}, \mathbf{A}^{-1})$ and

$$Z = f(\mathbf{z}_{MAP}) \frac{(2\pi)^{M/2}}{|\mathbf{A}|^{1/2}} \tag{10}$$

Note that $\mathbf{A}$ must be positive definite for a local maximum at $\mathbf{z}_{MAP}$.

We go on with how to apply the theory on Laplace approximation to the Logistic classifier. We firstly assume the prior on our weights to be of the following form:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_0, \mathbf{S}_0) \tag{11}$$

Hence for a dataset $\mathcal{D}(\mathbf{y}, \tilde{\mathbf{X}})$ the posterior distribution over $\mathbf{w}$ will be:

$$p(\mathbf{w}|\mathcal{D}) \propto p(\mathbf{w}) p(\mathcal{D}|\mathbf{w}) \tag{12}$$

From the central limit theorem, we know that the posterior distribution is expected to become better approximated by a Gaussian as the number of data points observed increases. Therefore, we will use Laplace approximation to find $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{MAP}, \mathbf{S}_N) \simeq p(\mathbf{w}|\mathcal{D})$. In this case, $f(\mathbf{w}) = p(\mathbf{w})p(\mathcal{D}|\mathbf{w})$.

By taking the logarithm of $f(\mathbf{w})$ and substituting the prior and the log-likelihood, we obtain:

$$f(\mathbf{w}) = -\frac{1}{2}(\mathbf{w}-\mathbf{m}_0)^T \mathbf{S}_0^{-1} (\mathbf{w}-\mathbf{m}_0) + \sum_{n=1}^{N} [y_n \ln(\sigma(\mathbf{w}^T \tilde{X}_n)) - (1-y_n)\ln(1-\sigma(\mathbf{w}^T \tilde{X}_n)] + const \tag{13}$$

We then proceed on finding $\mathbf{w}_{MAP}$ by maximizing $f(\mathbf{w})$. The covariance matrix is given by the inverse of the second derivative of the log-likelihood as follows:

$$\mathbf{S}_N^{-1} = -\nabla\nabla \ln f(\mathbf{w})|_{\mathbf{w}=\mathbf{w}_{MAP}} = \mathbf{S}_0^{-1} + \sum_{n=1}^{N} \sigma(\mathbf{w}^T \tilde{X}_n)(1-\sigma(\mathbf{w}^T \tilde{X}_n)) \tilde{X}_n \tilde{X}_n^T \tag{14}$$

The predictive distibution for a $\mathcal{C}_1$ for a new feature vector ϕ is obatained by marginalizing with respect to the posterior distribution $p(\mathbf{w}|\mathcal{D})$:

$$p(\mathcal{C}_1|\phi,\mathcal{D}) = \int p(\mathcal{C}_1|\phi,\mathbf{w})p(\mathbf{w}|\mathcal{D})d\mathbf{w} \simeq \int \sigma(\mathbf{w}^T\phi)q(\mathbf{w})d\mathbf{w} \tag{15}$$

Note that $p(\mathcal{C}_2|\phi,\mathcal{D}) = 1 - p(\mathcal{C}_1|\phi,\mathcal{D})$. Given that $\sigma(\mathbf{w}^T\phi)$ depends only on the projection of $\mathbf{w}$ onto ϕ, let $a = \mathbf{w}^T\phi$. This gives:

$$\sigma(\mathbf{w}^T\phi) = \int \delta(a - \mathbf{w}^T\phi)\sigma(a)da \tag{16}$$

Substituting 16 in 15 and denoting $p(a) = \int \delta(a-\mathbf{w}^T\phi)q(\mathbf{w})d\mathbf{w}$, we can rewrite the predictive distribution as:

$$p(\mathcal{C}_1|\phi,\mathcal{D}) = \int \sigma(a)p(a)da \tag{17}$$

We can view, the delta function as a distribution $g(a|\mathbf{w}) = \mathcal{N}(a,0)$. Thus, $p(a)$ is a marginal distribution from a joint distribution $q(\mathbf{w})$, therefore also a Gaussian for which we can calculate the mean μ_a and variance σ_a^2 using moments.

$$\mu_a = \mathbb{E}[a] = \int p(a)ada = \int q(\mathbf{w})\mathbf{w}^T\phi d(\mathbf{w}) = \mathbf{w}_{MAP}^T\phi \tag{18}$$

$$\sigma_a^2 = \mathbb{V}[a] = \int p(a)(a^2 - \mathbb{E}[a]^2)da = \int q(\mathbf{w})((\mathbf{w}^T\phi)^2 - (\mathbf{w}_{MAP}^T\phi)^2)d\mathbf{w} = \phi^T\mathbf{S}_N\phi \tag{19}$$

Hence,

$$p(\mathcal{C}_1|\mathcal{D}) = \int \sigma(a)\mathcal{N}(a|\mu_a,\sigma_a^2)da \tag{20}$$

By making use of the similarity between the logistic sigmoid and the probit funtion $\sigma(a) \simeq \Phi(\lambda a)$ with $\lambda^2 = \pi/8$, we can rewrite eq. 20 as follows:

$$p(\mathcal{C}_1|\mathcal{D}) = \int \Phi(\lambda a)\mathcal{N}(a|\mu_a,\sigma_a^2)da = \Phi\left(\frac{\mu_a}{\sqrt{\lambda^{-1/2} + \sigma_a^2}}\right) \tag{21}$$

Moreover, by defining $\kappa(\sigma^2) = \sqrt{(1 - \pi\sigma^2/8)^{-1}}$, the above equation becomes:

$$p(\mathcal{C}_1|\mathcal{D}) = \sigma(\kappa(\sigma_a^2)\mu_a) \tag{22}$$

We can see that when $\mu_a = 0$ the decision boundary corresponds to $p(\mathcal{C}_1|\mathcal{D}) = 0.5$ and it is the same boundary as the one obtained by using $\mathbf{w}_{MAP}$. The decision boundary of Laplace approximation is the same as the decision boundary of the MAP estimator as both minimize the number of misclassifications. Therefore, we expect the same values for the confusion matrices, but different log-likelihoods. We expect the Laplace log-lokelihoods to be smaller as it tries to avoid overfitting.

We continue by computing the model evidence of the Bayesian model ($Z = p(\mathcal{D})$). We have our dataset $\mathcal{D}$, a set of models $\mathcal{M}$ having parameters $\{\mathbf{w}_i\}$:

$$Z = p(\mathcal{D}) = \int f(\mathbf{w})d\mathbf{w} = \int p(\mathcal{D}|\mathbf{w})p(\mathbf{w})d\mathbf{w} \tag{23}$$

By applying 10 we obtain the following:

$$\begin{aligned} \ln p(\mathcal{D}) &\simeq \ln p(\mathcal{D}|w_{MAP}) + \ln p(\mathbf{w}_{MAP}) + \frac{M}{2}\ln(2\pi) + \frac{1}{2}\ln|\mathbf{S}_N| \\ &\simeq \ln p(\mathcal{D}|w_{MAP}) - \frac{1}{2}(\mathbf{w}_{MAP} - \mathbf{m}_0)^T\mathbf{S}_0^{-1}(\mathbf{w}_{MAP} - \mathbf{m}_0) + \frac{1}{2}\ln|\mathbf{S}_N| + \frac{1}{2}\ln|\mathbf{S}_0| \end{aligned} \tag{24}$$

The first term in the first line represents the log-likelihood evaluated at the MAP solution and the other three terms represent the "Occam factor" which penalises the model complexity. The model complexity depends on the number of the parameters and also on the prior that we assigned. Te best evidence is achieved by a trade-off between data misfit minimization and model complexity minimization. For the prior stated in the exercise we set $\mathbf{m}_0 = 0$ and $\mathbf{S}_0 = \sigma_0^2\mathbf{I}$.

3 Exercise b)

To find the MAP solution, we performed gradient based minimization using the python funcion `scipy.optimize.fmin_l_bfgs_b`. For this we give it as an argument for the gradient the Jacobian:

$$\frac{dp(\mathbf{w}|\mathcal{D})}{d\mathbf{w}} = \mathbf{w}^T\mathbf{S}_0^{-1} + \mathbf{X}^T[\mathbf{y} - \sigma(\mathbf{w}^T\mathbf{X})] \tag{25}$$

The code to train and fit the weights on a Gaussian with mean $\mathbf{w}_{MAP}$ and covariance $\mathbf{S}_N$ and to compute the model evidence based on the above theory (eq. 24) can be seen below. Note the use of Cholesky factorization to optimise the calculation of the determinants.

```
def logf(params, *args):
    #compute the negative log-likelihood
    w = params
    X_tilde_train = args[0]
    y_train = args[1]
    S_0_inv = args[2]
    sigmoid = predict(X_tilde_train,w)
    answer = np.dot(y_train, np.log(sigmoid)) + np.dot((1-y_train),np.log(1-sigmoid))
    answer = answer - 0.5*np.dot(np.matmul(w, S_0_inv), w)
    return (-answer)

def logfprime(params, *args):
    #compute Jacobian
    w = params
    X_tilde_train = args[0]
    y_train = args[1]
    cov_0_inv = args[2]
    sigmoid_value = predict(X_tilde_train,w)
    answer = np.dot(cov_0, w) - np.dot(np.transpose(X_tilde_train), (y_train - sigmoid_value))
    return answer

def compute_hessian(X_tilde, w, var):
    sigmoid = predict(X_tilde, w)
    cov_inv = (1/var) * np.identity(X_tilde.shape[ 1 ], dtype = float)
    for i in range (X_tilde.shape[0]):
    cov_inv = cov_inv + sigmoid[i]*(1 - sigmoid[i]) * np.outer(X_tilde[i], X_tilde[i])

    return cov_inv

def kappa(variance):
    return(1.0/ np.sqrt(1+np.pi * variance/8))

def predictive_distribution(feature, wmap, Sn):
    mean_a = np.dot(wmap, np.transpose(feature))
    variance_a = np.dot(feature, np.dot(Sn, feature))
```

```
    return logistic(kappa(variance_a)*mean_a)

def prediction(X_tilde, wmap, Sn):
    p = np.array([])
    for feature in X_tilde:
    p = np.append(p, predictive_distribution(feature, wmap, Sn))
return p

def plot_predictive_distribution2(X, y, w, Sn, map_inputs = lambda x : x):
    xx, yy = plot_data_internal(X, y)
    ax = plt.gca()
    X_tilde = get_x_tilde(map_inputs(np.concatenate((xx.ravel().reshape((-1, 1)),
                                yy.ravel().reshape((-1, 1))), 1)))
    Z = prediction(X_tilde, w, Sn)
    Z = Z.reshape(xx.shape)
    cs2 = ax.contour(xx, yy, Z, cmap = 'RdBu', linewidths = 2)
    plt.clabel(cs2, fmt = '%2.1f', colors = 'k', fontsize = 14)
    plt.show()

def compute_average_laplace(X_tilde, y, w, Sn):
    output_prob = prediction(X_tilde, w, Sn)
    return np.mean(y * np.log(output_prob) + (1 - y) * np.log(1.0 - output_prob))

def make_confussion_matrix_lap(X_test, y_test, w, Sn):
    confussion_matrix = np.array([[0.0, 0.0],
                                  [0.0, 0.0]])
    predicted_y = prediction(X_test, w, Sn) > 0.5
    NoO = np.sum(y_test)
    NoZ = np.size(y_test) - NoO
    for i in range (np.size(predicted_y)):
        if(y_test[i] == 0):
            if(predicted_y[i] == 0):
                confussion_matrix[0][0] += 1
            else:
                confussion_matrix[0][1] += 1
        else:
            if(predicted_y[i] == 0):
                confussion_matrix[1][0] += 1
            else:
                confussion_matrix[1][1] += 1
    confussion_matrix[0] = confussion_matrix[0] / NoZ
    confussion_matrix[1] = confussion_matrix[1] / NoO
    print(confussion_matrix)

def log_model_evidence(log_fmap, Sn_inv, S_0):
    S_Cholensky = np.linalg.cholesky(Sn_inv)
    S0_Cholensky = np.linalg.cholesky(S_0)
    sum = - np.sum(np.log(np.diag(S_Cholensky))) - np.sum(np.log(np.diag(S0_Cholensky)))  - log_fmap
    return sum

def train_wmap(var, X_tilde):
    beta = np.random.randn(X_tilde.shape[ 1 ])
```

```
    cov_inv = 1/var * np.identity(X_tilde_train.shape[ 1 ], dtype = float)
    wmap, fmap, d = scipy.optimize.fmin_l_bfgs_b(logf, x0=beta, args=(X_tilde_train, y_train, cov_inv),
    return wmap, fmap

#train and fit using the parameters given in c)

var = 1.
l = 0.1

X_tilde_train = get_x_tilde(evaluate_basis_functions(l, X_train, X_train))
X_tilde_test = get_x_tilde(evaluate_basis_functions(l, X_test, X_train))
wmap, fmap = train_wmap(var, X_tilde_train)
print(wmap, fmap)
```

4 Exercise c)

We then train the classifier using this code. We fixed the widths to be $l = 0.1$ and variance $\sigma_0^2 = 1.0$. Figure 1 displays the visualisation of the contours of the resulting class predictive probabilities on top of the data for Laplace approximation model and the MAP solution. As in the coursework for this model. Setts its classification boundaries tight around one class (specificlly class 1), it classified any outlier as class 2, so it is biased towards predicting class 2. However, the Laplace approximation model shows slightly expanded bounderies. We can notice that in the bottom-left part the bounderies for 0.3 and 0.2 expended. Although, this gives less certainty for outsiders to be classified as class 2, the decision boundery of 0.5 remains unchaged.

Figure 1: Plots showing data and contour lines for the predictive distribution generated by the Laplace approximation (left) and the MAP solution (right).

5 Exercise d)

The final average training and test log-likelihoods are shown in Tables 1 and 2 . These results indicate that the models tend to overfit the data as explained in the previous section. Although, they would work well for data with low variance, they will fail for highly varied data. Moreover, we can see that the Laplace log-likelihoods are smaller than the MAP solution, showing that although the model still overfits the data, it is less certain about its predictions than the MAP model.
The 2x2 confusion matrices on the train and test set is shown in Tables 3 and 4. By analising these tables, we conclude that, indeed, they are the same as predicted in Section 2. Both models compute their decision bounderies my minimizing the number of missclasifications.

Avg. Train ll	Avg. Test ll
-0.223	-0.284

Table 1: Log-likelihoods for MAP solution.

Avg. Train ll	Avg. Test ll
-0.262	-0.313

Table 2: Log-likelihoods for Laplace approximation.

		$\hat{y}$	
		0	1
y	0	0.95	0.05
	1	0.14	0.86

Table 3: Conf. matrix for for MAP solution.

		$\hat{y}$	
		0	1
y	0	0.95	0.05
	1	0.14	0.86

Table 4: Conf. matrix for Laplace approximation.

6 Exercise e)

In this exercise we aim to tune our parameters σ_0^2 and l by using a grid search. We construct a 10×10 matrix representin all combination of σ_0^2 and l. From coursework, we expect that the best value for l is in the interval of $(0.1, 1)$ and we choose σ_0^2 in the interval of $(0.1, 100)$. Moreover, we will choose logarithmically spaced values as we expect the model complexity to increase with the varience and therfore penalise our model evidence. The entries of the matrix will consist of the approximation of the model evidence and we will make use of a heat map to visualise our findings (See Figure 2). The best model evidence corresponds to $\sigma_0^2 = 1$ and $l = 0.6$. It is worth noticing that the second best is very close and gives the values of $\sigma_0^2 = 0.5$ and $l = 0.5$. The code used can be seen below.

```
l = np.geomspace(0.1, 1, num = 10)# XXX Width of the Gaussian basis funcction.
var = np.geomspace(0.1, 100, num = 10)

heat_map = np.ones((len(l), len(var)))

for m in range (len(l)):
    for n in range (len(var)):
        X_tilde_train = get_x_tilde(evaluate_basis_functions(l[m], X_train, X_train))
         X_tilde_test = get_x_tilde(evaluate_basis_functions(l[m], X_test, X_train))

        wmap, fmap = train_wmap(var[n], X_tilde_train)
        S_0 = var[n] * np.identity(X_tilde_train.shape[ 1 ], dtype = float)
        Sn_inv = compute_hessian(X_tilde_train, wmap, var[n])

        heat_map[m][n] = log_model_evidence(fmap, Sn_inv, S_0)
        if(minim > heat_map[m][n]):
            minim = heat_map[m][n]
            m_prim = m
            n_prim = n

print("Optimized l, m: ({}, {})".format(l[m_prim], var(n_prim)))
ax = sns.heatmap(heat_map)
```

Finally, from Figure 2 we can see how the model evidence is affected by our choice on the prior distibution. For very large variances the model complexity increases and penalises the model evidence.

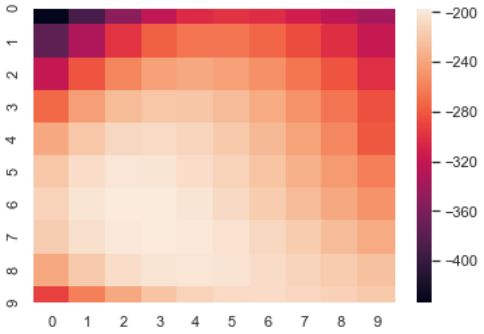

Figure 2: Heat map plot of the the approximation of the model evidence obtained in the grid search.

7 Exercise f)

The training and test log-likelihoods per datapoint and confiusion matrix obtained for $\sigma_0^2 = 1$ and $l = 0.6$ can be seen in Tables 5 and 6. Figure 3 displays the visualisation of the contours of the resulting class predictive probabilities on top of the data after tuning the parameters. From these, we can clearly see the data separet by the expended bouderies. Moreover, the log-likelihoods show that this model fits and generalizes very well(the best we have seen so far including coursework). Furthermore, by looking at the confussion matrix we can see that this model predicts the data with higher accuracy than the previuos ones.

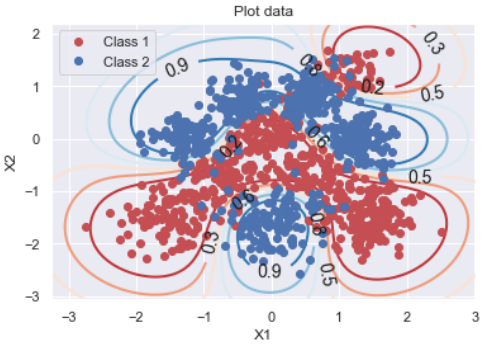

Figure 3: Visualisation of the contours of the class predictive probabilities for Laplace approximation after hyper-parameter tuning by maximising the model evidence.

Avg. Train ll	**Avg. Test ll**
-0.198	-0.176

Table 5: Average training and test log-likelihoods for Laplace approximation after hyper-parameter tuning by maximising the model evidence.

		$\hat{y}$	
		0	1
y	0	0.96	0.04
	1	0.09	0.91

Table 6: Confusion matrix for Laplace approximation after hyper-parameter tuning by maximising the model evidence.

8 Conclusions

1. The Bayesian binary classifier performs better the simple RBF Logistic Classifier. Although, it requires more tuning on the choice of the prior and width. It can also computationally expansive as it requires the computation of determinants of high dimensional matrices. This can be improved by using optimized algorithms.

2. We do not need to know the normalization constant Z of the true distribution to apply the Laplace approximation.

3. For multimodal data we will need different Laplace approximations for each mode.

4. The Laplace approximation is that, since it is based on a Gaussian distribution, it is only directly applicable to real variables.

5. The Laplace approximationis based purely on the aspects of the true distribution at a specific value of the variable, and so can fail to capture important global properties.

References

Patern recognition and machine learning Christopher M. Bishop *Patern recognition and machine learning* Spinger p.213-p.218