Homework 5: Conditional Probability Models

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython), though if you need to you may scan handwritten work. You may find the minted package convenient for including source code in your LaTeX document. If you are using LyX, then the listings package tends to work better.

1 Introduction

In this homework we'll be investigating conditional probability models, with a focus on various interpretations of logistic regression, with and without regularization. Along the way we'll discuss the calibration of probability predictions, both in the limit of infinite training data and in a more bare-hands way. On the Bayesian side, we'll recreate from scratch the Bayesian linear gaussian regression example we discussed in lecture. We'll also have several optional problems that work through many basic concepts in Bayesian statistics via one of the simplest problems there is: estimating the probability of heads in a coin flip. Later we'll extend this to the probability of estimating click-through rates in mobile advertising. Along the way we'll encounter empirical Bayes and hierarchical models.

2 From Scores to Conditional Probabilities[1]

Let's consider the classification setting, in which $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \{-1, 1\}$ are sampled i.i.d. from some unknown distribution. For a prediction function $f : \mathcal{X} \to \mathbf{R}$, we define the **margin** on an example (x, y) to be $m = yf(x)$. Since our class predictions are given by $\text{sign}(f(x))$, we see that a prediction is correct iff $m(x) > 0$. It's tempting to interpret the magnitude of the score $|f(x)|$ as a measure of confidence. However, it's hard to interpret the magnitudes beyond saying one prediction score is more or less confident than another, and without any scale to this "confidence score", it's hard to know what to do with it. In this problem, we investigate how we can translate the score into a probability, which is much easier to interpret. In other words, we are looking for a way to convert score function $f(x) \in \mathbf{R}$ into a conditional probability distribution $x \mapsto p(y = 1 \mid x)$.

In this problem we will consider **margin-based losses**, which are loss functions of the form $(y, f(x)) \mapsto \ell(yf(x))$, where $m = yf(x)$ is called the **margin**. We are interested in how we can go from an empirical risk minimizer for a margin-based loss, $\hat{f} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \ell(y_i f(x_i))$, to a conditional probability estimator $\hat{\pi}(x) \approx p(y = 1 \mid x)$. Our approach will be to try to find a

[1] This problem is based on Section 7.5.3 of Schapire and Freund's book *Boosting: Foundations and Algorithms*.

way to use the Bayes[2] prediction function[3] $f^* = \arg\min_f \mathbb{E}_{x,y}[\ell(yf(x)]$ to get the true conditional probability $\pi(x) = p(y = 1 \mid x)$, and then apply the same mapping to the empirical risk minimizer. While there is plenty that can go wrong with this "plug-in" approach (primarily, the empirical risk minimizer from a [limited] hypothesis space $\mathcal{F}$ may be a poor estimate for the Bayes prediction function), it is at least well-motivated, and it can work well in practice. And **please note** that we can do better than just hoping for success: if you have enough validation data, you can directly assess how well "calibrated" the predicted probabilities are. This blog post has some discussion of calibration plots: https://jmetzen.github.io/2015-04-14/calibration.html.

It turns out it is straightforward to find the Bayes prediction function f^* for margin losses, at least in terms of the data-generating distribution: For any given $x \in \mathcal{X}$, we'll find the best possible prediction $\hat{y}$. This will be the $\hat{y}$ that minimizes

$$\mathbb{E}_y \left[\ell\left(y\hat{y}\right) \mid x\right].$$

If we can calculate this $\hat{y}$ for all $x \in \mathcal{X}$, then we will have determined $f^*(x)$. We will simply take

$$f^*(x) = \arg\min_{\hat{y}} \mathbb{E}_y \left[\ell\left(y\hat{y}\right) \mid x\right].$$

Below we'll calculate f^* for several loss functions. It will be convenient to let $\pi(x) = p\left(y = 1 \mid x\right)$ in the work below.

1. Write $\mathbb{E}_y \left[\ell\left(yf(x)\right) \mid x\right]$ in terms of $\pi(x)$, $\ell(-f(x))$, and $\ell\left(f(x)\right)$. [Hint: Use the fact that $y \in \{-1, 1\}$.]

$$\begin{aligned}\mathbb{E}_y \left[\ell\left(yf(x)\right) \mid x\right] &= \mathbb{E}_y \left[\ell\left(f(x)\right) \mid x\right] p(y = 1|x) + \mathbb{E}_y \left[\ell\left(-f(x)\right) \mid x\right] p(y = -1|x) \\ &= \ell(f(x))\pi(x) + \ell(-f(x))(1 - \pi(x))\end{aligned}$$

2. Show that the Bayes prediction function $f^*(x)$ for the exponential loss function $\ell\left(y, f(x)\right) = e^{-yf(x)}$ is given by

$$f^*(x) = \frac{1}{2} \ln\left(\frac{\pi(x)}{1 - \pi(x)}\right),$$

where we've assumed $\pi(x) \in (0, 1)$. Also, show that given the Bayes prediction function f^*, we can recover the conditional probabilities by

$$\pi(x) = \frac{1}{1 + e^{-2f^*(x)}}.$$

[Hint: Differentiate the expression in the previous problem with respect to $f(x)$. To make things a little less confusing, and also to write less, you may find it useful to change variables a bit: Fix an $x \in \mathcal{X}$. Then write $p = \pi(x)$ and $\hat{y} = f(x)$. After substituting these into the expression you had for the previous problem, you'll want to find $\hat{y}$ that minimizes the expression. Use differential calculus. Once you've done it for a single x, it's easy to write the

[2] Don't be confused – it's Bayes as in "Bayes optimal", as we discussed at the beginning of the course, not Bayesian as we've discussed more recently.

[3] In this context, the Bayes prediction function is often referred to as the "population minimizer." In our case, "population" refers to the fact that we are minimizing with respect to the true distribution, rather than a sample. The term "population" arises from the context where we are using a sample to approximate some statistic of an entire population (e.g. a population of people or trees).

solution as a function of x.]
For an fixed $x \in \mathcal{X}$ write $p = \pi(x)$ and $\hat{y} = f(x)$, we get:

$$\frac{\partial E_y}{\partial \hat{y}} = -pe^{-\hat{y}} + (1-p)e^{\hat{y}} = 0$$

$$e^{2\hat{y}} = \frac{p}{1-p}$$

$$f^*(x) = \frac{1}{2}\ln\frac{\pi(x)}{1-\pi x}$$

Transform the equation above, we can get:

$$\pi(x) = \frac{1}{1+e^{-2f^*(x)}}$$

3. Show that the Bayes prediction function $f^*(x)$ for the logistic loss function $\ell(y, f(x)) = \ln\left(1+e^{-yf(x)}\right)$ is given by

$$f^*(x) = \ln\left(\frac{\pi(x)}{1-\pi(x)}\right)$$

and the conditional probabilities are given by

$$\pi(x) = \frac{1}{1+e^{-f^*(x)}}.$$

Again, we may assume that $\pi(x) \in (0,1)$.

For an fixed $x \in \mathcal{X}$ write $P = \pi(x)$ and $\hat{y} = f(x)$, we get:

$$\frac{\partial E_y}{\partial \hat{y}} = -p\frac{1}{1+e^{\hat{y}}} + (1-p)\frac{e^{\hat{y}}}{1+e^{\hat{y}}} = 0$$

$$e^{\hat{y}} = \frac{p}{1-p}$$

$$f^*(x) = \ln\frac{\pi(x)}{1-\pi(x)}$$

Transform the equation above, we can get:

$$\pi(x) = \pi(x) = \frac{1}{1+e^{-f^*(x)}}$$

4. [Optional] Show that the Bayes prediction function $f^*(x)$ for the hinge loss function $\ell(y, f(x)) = \max(0, 1-yf(x))$ is given by

$$f^*(x) = \text{sign}\left(\pi(x) - \frac{1}{2}\right).$$

Note that it is impossible to recover $\pi(x)$ from $f^*(x)$ in this scenario. However, in practice we work with an empirical risk minimizer, from which we may still be able to recover a reasonable estimate for $\pi(x)$. An early approach to this problem is known as "Platt scaling": https://en.wikipedia.org/wiki/Platt_scaling.

3 Logistic Regression

3.1 Equivalence of ERM and probabilistic approaches

In lecture we discussed two different ways to end up with logistic regression.

ERM approach: Consider the classification setting with input space $\mathcal{X} = \mathbf{R}^d$, outcome space $\mathcal{Y}_{\pm} = \{-1, 1\}$, and action space $\mathcal{A}_{\mathbf{R}} = \mathbf{R}$, with the hypothesis space of linear score functions $\mathcal{F}_{\text{score}} = \left\{x \mapsto x^T w \mid w \in \mathbf{R}^d\right\}$. Consider the margin-based loss function $\ell_{\text{logistic}}(m) = \log\left(1 + e^{-m}\right)$ and the training data $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$. Then the empirical risk objective function for hypothesis space $\mathcal{F}_{\text{score}}$ and the logistic loss over $\mathcal{D}$ is given by

$$\begin{aligned}\hat{R}_n(w) &= \frac{1}{n}\sum_{i=1}^{n} \ell_{\text{logistic}}(y_i w^T x_i) \\ &= \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i w^T x_i\right)\right).\end{aligned}$$

Bernoulli regression with logistic transfer function: Consider the conditional probability modeling setting with input space $\mathcal{X} = \mathbf{R}^d$, outcome space $\mathcal{Y}_{0/1} = \{0, 1\}$, and action space $\mathcal{A}_{[0,1]} = [0, 1]$, where an action corresponds to the predicted probability that an outcome is 1. Define the **standard logistic function** as $\phi(\eta) = 1/\left(1 + e^{-\eta}\right)$ and the hypothesis space $\mathcal{F}_{\text{prob}} = \left\{x \mapsto \phi(w^T x) \mid w \in \mathbf{R}^d\right\}$. Suppose for every y_i in the dataset $\mathcal{D}$ above, we define $y_i' = \begin{cases} 1 & y_i = 1 \\ 0 & y_i = -1 \end{cases}$, and let $\mathcal{D}'$ be the resulting collection of (x_i, y_i') pairs. Then the negative log-likelihood (NLL) objective function for $\mathcal{F}_{\text{prob}}$ and $\mathcal{D}'$ is give by

$$\begin{aligned}\text{NLL}(w) &= -\sum_{i=1}^{n} y_i' \log \phi(w^T x_i) + (1 - y_i') \log\left(1 - \phi(w^T x_i)\right) \\ &= \sum_{i=1}^{n} \left[-y_i' \log \phi(w^T x_i)\right] + (y_i' - 1) \log\left(1 - \phi(w^T x_i)\right)\end{aligned}$$

If $\hat{w}_{\text{prob}}$ minimizes $\text{NLL}(w)$, then $x \mapsto \phi(x^T \hat{w}_{\text{prob}})$ is a maximum likelihood prediction function over the hypothesis space $\mathcal{F}_{\text{prob}}$ for the dataset $\mathcal{D}'$.

Show that $n\hat{R}_n(w) = \text{NLL}(w)$ for all $w \in \mathbf{R}^d$. And thus the two approaches are equivalent, in that they produce the same prediction functions.

$$\text{NLL}(w) = \sum_{i=1}^{n} \left[-y_i' \log \phi(w^T x_i)\right] + (y_i' - 1) \log\left(1 - \phi(w^T x_i)\right)$$

If $y_i = 1$, then $y_i' = 1 \implies \text{NLL}(w) = \sum_{i=1}^{n} \left[-\log \phi(w^T x_i)\right] = \sum_{i=1}^{n} \log(1 + exp(-y_i w^T x_i)) = n\hat{R}_n(w)$

If $y_i = -1$, then $y_i' = 0 \implies \text{NLL}(w) = \sum_{i=1}^{n} \left[-\log(1 - \phi(w^T x_i))\right] = \sum_{i=1}^{n} \log(1 + exp(-y_i w^T x_i)) = n\hat{R}_n(w)$

The two approaches are equivalent.

3.2 Numerical Overflow and the log-sum-exp trick

Suppose we want to calculate $\log(\exp(\eta))$ for $\eta = 1000.42$. If we compute this literally in Python, we will get an overflow (try it!), since numpy gets infinity for $e^{1000.42}$, and log of infinity is still infinity. On the other hand, we can help out with some math: obviously $\log(\exp(\eta)) = \eta$, and there's no issue.

It turns out, $\log(\exp(\eta))$ and the problem with its calculation is a special case of the LogSumExp function that shows up frequently in machine learning. We define

$$\text{LogSumExp}(x_1, \ldots, x_n) = \log\left(e^{x_1} + \cdots + e^{x_n}\right).$$

Note that this will overflow if any of the x_i's are large (more than 709). To compute this on a computer, we can use the "**log-sum-exp trick**". We let $x^* = \max(x_1, \ldots, x_n)$ and compute LogSumExp as

$$\text{LogSumExp}(x_1, \ldots, x_n) = x^* + \log\left[e^{x_1 - x^*} + \cdots + e^{x_n - x^*}\right].$$

1. Show that the new expression for LogSumExp is valid.

$$\begin{aligned} x^* = \log[e^{x^*}] \implies \text{LogSumExp} &= \log[e^{x^*}] + \log\left[e^{x_1 - x^*} + \cdots + e^{x_n - x^*}\right] \\ &= \log\left[e^{x^*}(e^{x_1 - x^*} + \cdots + e^{x_n - x^*})\right] \\ &= \log\left(e^{x_1} + \cdots + e^{x_n}\right) \end{aligned}$$

2. Show that $\exp(x_i - x^*) \in (0, 1]$ for any i, and thus the exp calculations will not overflow.

$$x^* = \max(x_1, \ldots, x_n) \implies x_i - x^* \leq 0 \implies \exp(x_i - x^*) \in (0, 1]$$

3. Above we've only spoken about the exp overflowing. However, the log part can also have problems by becoming negative infinity for arguments very close to 0. Explain why the log term in our expression $\log\left[e^{x_1 - x^*} + \cdots + e^{x_n - x^*}\right]$ will never be "-inf".
   $\log[x]$ will be "-inf" if x is very close to 0, but we have at least $e^{x^* - x^*} = 1$. Hence $\log\left[e^{x_1 - x^*} + \cdots + e^{x_n - x^*}\right]$ will never be "-inf".

4. In the objective functions for logistic regression, there are expressions of the form $\log(1 + e^{-s})$ for some s. Note that a naive implementation gives 0 for $s > 36$ and inf for $s < -709$. Show how to use the numpy function *logaddexp* to correctly compute $\log(1 + e^{-s})$.
   Use numpy.logaddexp(0,-s)

3.3 Regularized Logistic Regression

For a dataset $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$ drawn from $\mathbf{R}^d \times \{-1, 1\}$, the regularized logistic regression objective function can be defined as

$$\begin{aligned} J_{\text{logistic}}(w) &= \hat{R}_n(w) + \lambda \|w\|^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i w^T x_i\right)\right) + \lambda \|w\|^2. \end{aligned}$$

1. Prove that the objective function $J_{\text{logistic}}(w)$ is convex. You may use any facts mentioned in the convex optimization notes.

   In the notes, we have proven that $\lambda\|\omega\|^2$ and Log_Sum_Exp are a convex functions. Hence the objective function is convex.

2. Complete the `f_objective` function in the skeleton code, which computes the objective function for $J_{\text{logistic}}(w)$. Make sure to use the log-sum-exp trick to get accurate calculations and to prevent overflow.

```python
import numpy as np
def f_objective(theta, X, y, l2_param):
    '''
    Args:
        theta: 1D numpy array of size num_features
        X: 2D numpy array of size (num_instances, num_features)
        y: 1D numpy array of size num_instances
        l2_param: regularization parameter

    Returns:
        objective: scalar value of objective function
    '''
    num_ftrs=X.shape[1]
    num_ins=X.shape[0]
    loss=0
    for i in range(num_ins):
        loss+=np.logaddexp(0,-y[i]*np.dot(theta,X[i]))
    loss=loss/num_ins
    reg=l2_param*(np.dot(theta,theta))
    return(loss+reg)
```

3. Complete the `fit_logistic_regression_function` in the skeleton code using the `minimize` function from `scipy.optimize`. `ridge_regression.py` from Homework 2 gives an example of how to use the `minimize` function. Use this function to train a model on the provided data. Make sure to take the appropriate preprocessing steps, such as standardizing the data and adding a column for the bias term.

```
from scipy.optimize import minimize
from functools import partial
def fit_logistic_reg(X, y, objective_function, l2_param):
    '''
    Args:
        X: 2D numpy array of size (num_instances, num_features)
        y: 1D numpy array of size num_instances
        objective_function: function returning the value of the objective
        l2_param: regularization parameter

    Returns:
        optimal_theta: 1D numpy array of size num_features
    '''
    num_ftrs= X.shape[1]
    obj_func = partial(objective_function, X = X, y=y, l2_param = l2_param)
    w_0 = np.ones(num_ftrs)
    w_ = minimize(obj_func, w_0).x
    return w_
```

```
with open("X_train.txt") as textFile:
    x_train = [line.split() for line in textFile]
x_train=[[float(v) for v in r[0].split(',')] for r in x_train]
with open("X_val.txt") as textFile:
    x_val = [line.split() for line in textFile]
x_val=[[float(v) for v in r[0].split(',')] for r in x_val]
with open("y_train.txt") as textFile:
    y_train = [line.split() for line in textFile]
y_train=[[float(v) for v in r[0].split(',')] for r in y_train]
with open("y_val.txt") as textFile:
    y_val = [line.split() for line in textFile]
y_val=[[float(v) for v in r[0].split(',')] for r in y_val]
x_train, x_val, y_train, y_val = np.array(x_train),np.array(x_val),np.array(y_train),np.array(y_v
```

```
from sklearn.preprocessing import StandardScaler
SS = StandardScaler()
X_train = SS.fit_transform(x_train)
X_val = SS.fit_transform(x_val)
X_train = np.hstack((X_train, np.ones((X_train.shape[0], 1))))
X_val = np.hstack((X_val, np.ones((X_val.shape[0], 1))))
y_train[y_train == 0] = -1
y_val[y_val == 0] = -1
```

```
theta=fit_logistic_reg(X_train, y_train, f_objective, l2_param=1)
```

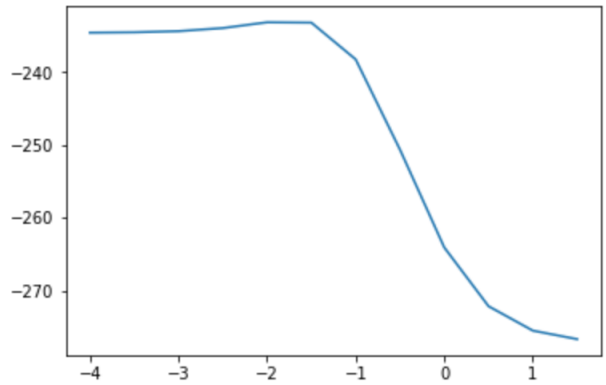
4. Find the ℓ_2 regularization parameter that minimizes the log-likelihood on the validation set. Plot the log-likelihood for different values of the regularization parameter.

```
def log_likelihood(theta,X,y):
    num_ftrs=X.shape[1]
    num_ins=X.shape[0]
    loss=0
    for i in range(num_ins):
        loss += np.logaddexp(0,-y[i]*np.dot(theta, X[i]))
    return(-loss)
L2=list([10.0**i for i in np.arange(-4,2,0.5)])
l=[]
for l2reg in L2:
    theta = fit_logistic_reg(X_train, y_train, f_objective, l2_param=l2reg)
    result = log_likelihood(theta,X_val,y_val)
    l.append(result)
```

```
l
```

```
[array([-234.58149407]),
 array([-234.52777074]),
 array([-234.371965]),
 array([-233.94866293]),
 array([-233.15498229]),
 array([-233.19850266]),
 array([-238.25171206]),
 array([-250.64983843]),
 array([-264.09589474]),
 array([-272.1777477]),
 array([-275.530377]),
 array([-276.69882865])]
```

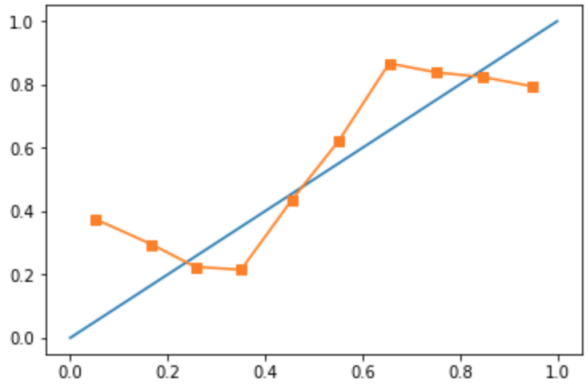
```
import matplotlib.pyplot as plt
log_L2 = [np.log10(i) for i in L2]
plt.plot(log_L2,l)
plt.show()
```



When $\lambda = 0.01$, the log likelihood function on the validation set reaches maximum, which is -233.15498229.

5. Based on the Bernoulli regression development of logistic regression, it seems reasonable to interpret the prediction $f(x) = \phi(w^T x) = 1/\left(1 + e^{-w^T x}\right)$ as the probability that $y = 1$, for a randomly drawn pair (x, y). Since we only have a finite sample (and we are regularizing, which will bias things a bit) there is a question of how well "calibrated" our predicted probabilities are. Roughly speaking, we say $f(x)$ is well calibrated if we look at all examples (x, y) for which $f(x) \approx 0.7$ and we find that close to 70% of those examples have $y = 1$, as predicted... and then we repeat that for all predicted probabilities in $(0, 1)$. To see how well-calibrated our predicted probabilities are, break the predictions on the validation set into groups based on the predicted probability (you can play with the size of the groups to get a result you think is informative). For each group, examine the percentage of positive labels. You can make a table or graph. Summarize the results. You may get some ideas and references from scikit-learn's discussion.

```
from sklearn import datasets
from sklearn.calibration import calibration_curve

plt.plot([0, 1], [0, 1])
n_ins, num_ftrs = X_val.shape
theta_opt=fit_logistic_reg(X_train, y_train, f_objective, l2_param=10**(-2))
prob_pos = np.dot(X_val,theta_opt)
for i in range(n_ins):
    prob_pos[i] = 1/(1+np.exp(-prob_pos[i]))
prob_pos = (prob_pos - prob_pos.min()) / (prob_pos.max() - prob_pos.min())
fraction_of_positives, mean_predicted_value = calibration_curve(y_val, prob_pos, n_bins=10)
plt.plot(mean_predicted_value, fraction_of_positives, "s-",label="%s" % ('Logistic'))
plt.show()
```




```
fraction_of_positives
```

```
array([0.375     , 0.29545455, 0.22413793, 0.21538462, 0.43396226,
       0.62162162, 0.86666667, 0.83783784, 0.82352941, 0.79411765])
```

I break the validation predictions into 10 groups. The prediction is not well calibrated.

6. [Optional] If you can, create a dataset for which the log-sum-exp trick is actually necessary for your implementation of regularized logistic regression. If you don't think such a dataset

exists, explain why. If you like, you may consider the case of SGD optimization. [This problem is intentionally open-ended. You're meant to think, explore, and experiment. Points assigned for interesting insights.]

4 Bayesian Logistic Regression with Gaussian Priors

Let's return to the setup described in Section 3.1 and, in particular, to the Bernoulli regression setting with logistic transfer function. We had the following hypothesis space of conditional probability functions:

$$\mathcal{F}_{\text{prob}} = \left\{ x \mapsto \phi(w^T x) \mid w \in \mathbf{R}^d \right\}.$$

Now let's consider the Bayesian setting, where we induce a prior on $\mathcal{F}_{\text{prob}}$ by taking a prior $p(w)$ on the parameter $w \in \mathbf{R}^d$.

1. For the dataset $\mathcal{D}'$ described in Section 3.1, give an expression for the posterior density $p(w \mid \mathcal{D}')$ in terms of the negative log-likelihood function

$$\text{NLL}_{\mathcal{D}'}(w) \quad = \quad -\sum_{i=1}^{n} y_i' \log \phi(w^T x_i) + (1 - y_i') \log \left(1 - \phi(w^T x_i)\right)$$

and a prior density $p(w)$ (up to a proportionality constant is fine).

$$\mathbb{P}(w|D') = \frac{\mathbb{P}(D'|w)\mathbb{P}(w)}{\mathbb{P}(D')} = \frac{exp(-\text{NLL}_{\mathcal{D}'}(w))\mathbb{P}(w)}{\mathbb{P}(D')}$$

2. Suppose we take a prior on w of the form $w \sim \mathcal{N}(0, \Sigma)$. Find a covariance matrix Σ such that MAP estimate for w after observing data $\mathcal{D}'$ is the same as the minimizer of the regularized logistic regression function defined in Section 3.3 (and prove it). [Hint: Consider minimizing the negative log posterior of w. Also, remember you can drop any terms from the objective function that don't depend on w. Also, you may freely use results of previous problems.]

$-log(P(w|D')) = \text{NLL} - log(P(w)) + log(P(D))$

We have proven that $\text{NLL} = n\hat{R}_n$, and $log(\mathbb{P}(D))$ is a constant.

Hence, $-log(P(w)) = -\frac{1}{2}log(|2\pi\Sigma|) + \frac{1}{2}w^T\Sigma^{-1}w$ corresponds to the regularization term $\frac{1}{2}w^T\Sigma^{-1}w = n\lambda w^T w$

$\Sigma = \frac{1}{2n\lambda}I$

Proof: when $w \sim \mathcal{N}(0, \frac{1}{2n\lambda}I), \hat{w}_{MAP} = argmin[-\log p(w|D)] = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i w^T x_i\right)\right) + \lambda\|w\|^2$

3. In the Bayesian approach, the prior should reflect your beliefs about the parameters before seeing the data and, in particular, should be independent on the eventual size of your dataset. Following this, you choose a prior distribution $w \sim \mathcal{N}(0, I)$. For a dataset $\mathcal{D}$ of size n, how should you choose λ in our regularized logistic regression objective function so that the

minimizer is equal to the mode of the posterior distribution of w (i.e. is equal to the MAP estimator).
For a $\Sigma = I$, to make LR equivalent to the prior distribution of w, we can make $\frac{1}{2n\lambda} = 1$, that is $\lambda = \frac{1}{2n}$.

5 Bayesian Linear Regression - Implementation

In this problem, we will implement Bayesian Gaussian linear regression, essentially reproducing the example from lecture, which in turn is based on the example in Figure 3.7 of Bishop's *Pattern Recognition and Machine Learning* (page 155). We've provided plotting functionality in "support_code.py". Your task is to complete "problem.py". The implementation uses np.matrix objects, and you are welcome to use[4] the np.matrix.getI method.

1. Implement likelihood_func.

```
import matplotlib.pyplot as plt
import numpy.matlib as matlib
from scipy.stats import multivariate_normal
import numpy as np
import support_code

def likelihood_func(w, X, y_train, likelihood_var):
    '''
    Implement likelihood_func. This function returns the data likelihood
    given f(y_train | X; w) ~ Normal(Xw, likelihood_var).

    Args:
        w: Weights
        X: Training design matrix with first col all ones (np.matrix)
        y_train: Training response vector (np.matrix)
        likelihood_var: likelihood variance

    Returns:
        likelihood: Data likelihood (float)
    '''

    #TO DO
    num_ins=X.shape[0]
    likelihood=1.0
    for i in range(num_ins):
        ml=multivariate_normal.pdf(y_train[i], np.dot(X[i],w), likelihood_var)
        likelihood*=ml
    return likelihood
```

2. Implement get_posterior_params.

[4]However, in practice we are usually interested in computing the product of a matrix inverse and a vector, i.e. $X^{-1}b$. In this case, it's usually faster and more accurate to use a library's algorithms for solving a system of linear equations. Note that $y = X^{-1}b$ is just the solution to the linear system $Xy = b$. See for example John Cook's blog post for discussion.

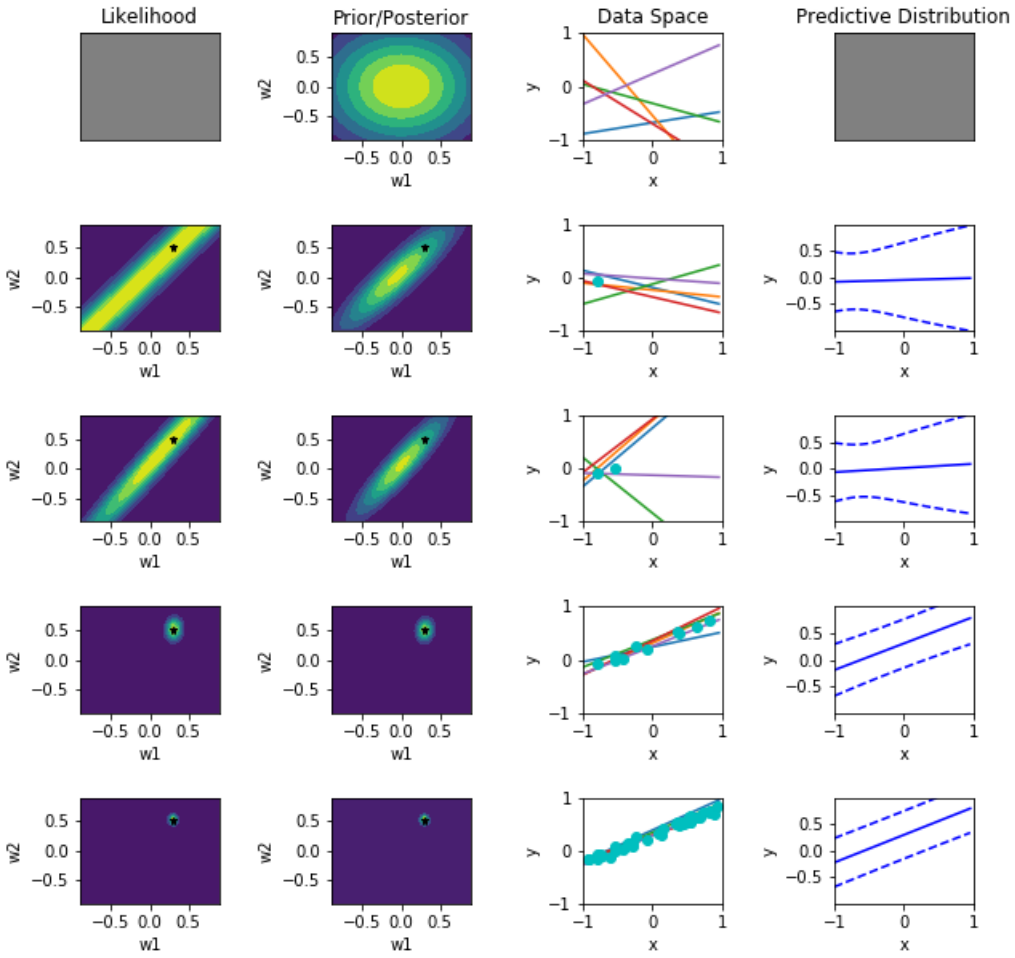
```python
def get_posterior_params(X, y_train, prior, likelihood_var = 0.2**2):
    '''
    Implement get_posterior_params. This function returns the posterior
    mean vector \mu_p and posterior covariance matrix \Sigma_p for
    Bayesian regression (normal likelihood and prior).

    Note support_code.make_plots takes this completed function as an argument.

    Args:
        X: Training design matrix with first col all ones (np.matrix)
        y_train: Training response vector (np.matrix)
        prior: Prior parameters; dict with 'mean' (prior mean np.matrix)
               and 'var' (prior covariance np.matrix)
        likelihood_var: likelihood variance- default (0.2**2) per the lecture slides

    Returns:
        post_mean: Posterior mean (np.matrix)
        post_var: Posterior mean (np.matrix)
    '''

    # TO DO
    mean_MP=np.linalg.inv(np.dot(X.T,X)+likelihood_var*np.linalg.inv(prior['var']))
    post_mean=np.dot(np.dot(mean_MP,X.T),y_train)
    post_var=np.linalg.inv(1/likelihood_var*np.dot(X.T,X)+np.linalg.inv(prior['var']))
    return post_mean, post_var
```

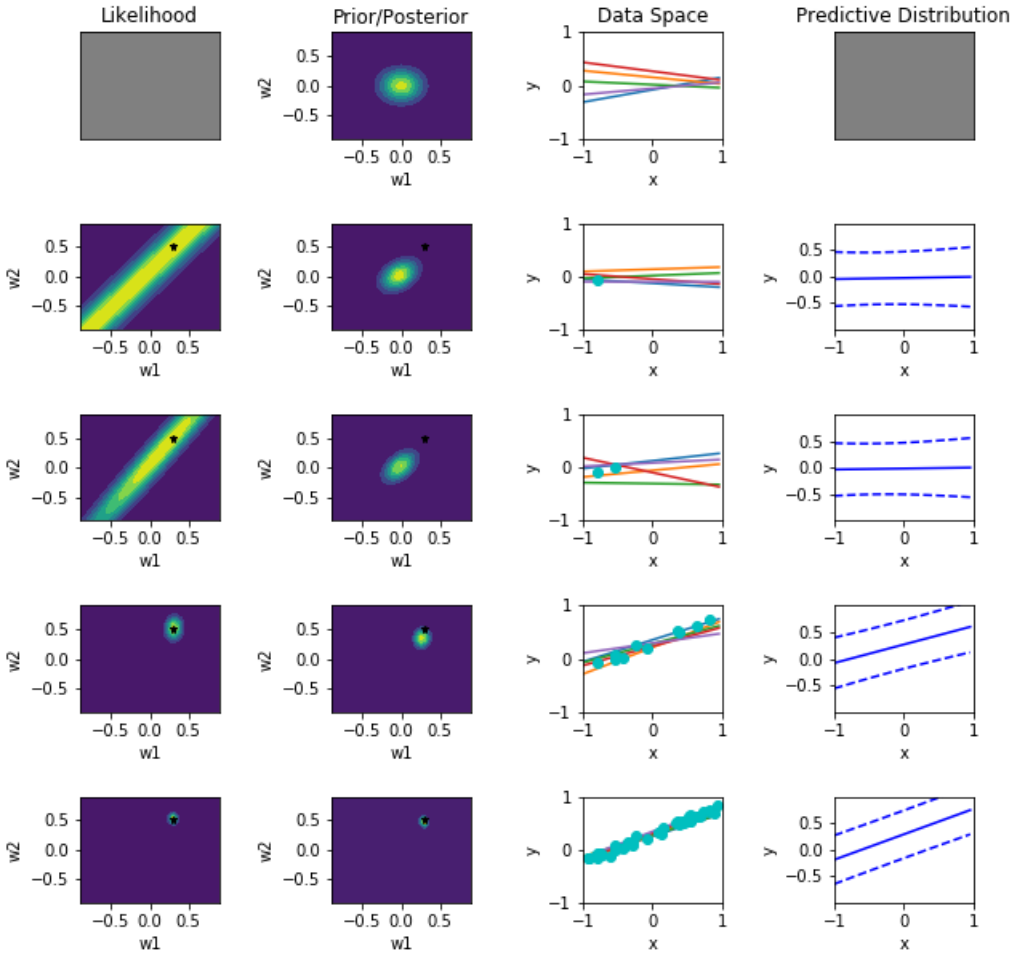
3. Implement get_predictive_params.

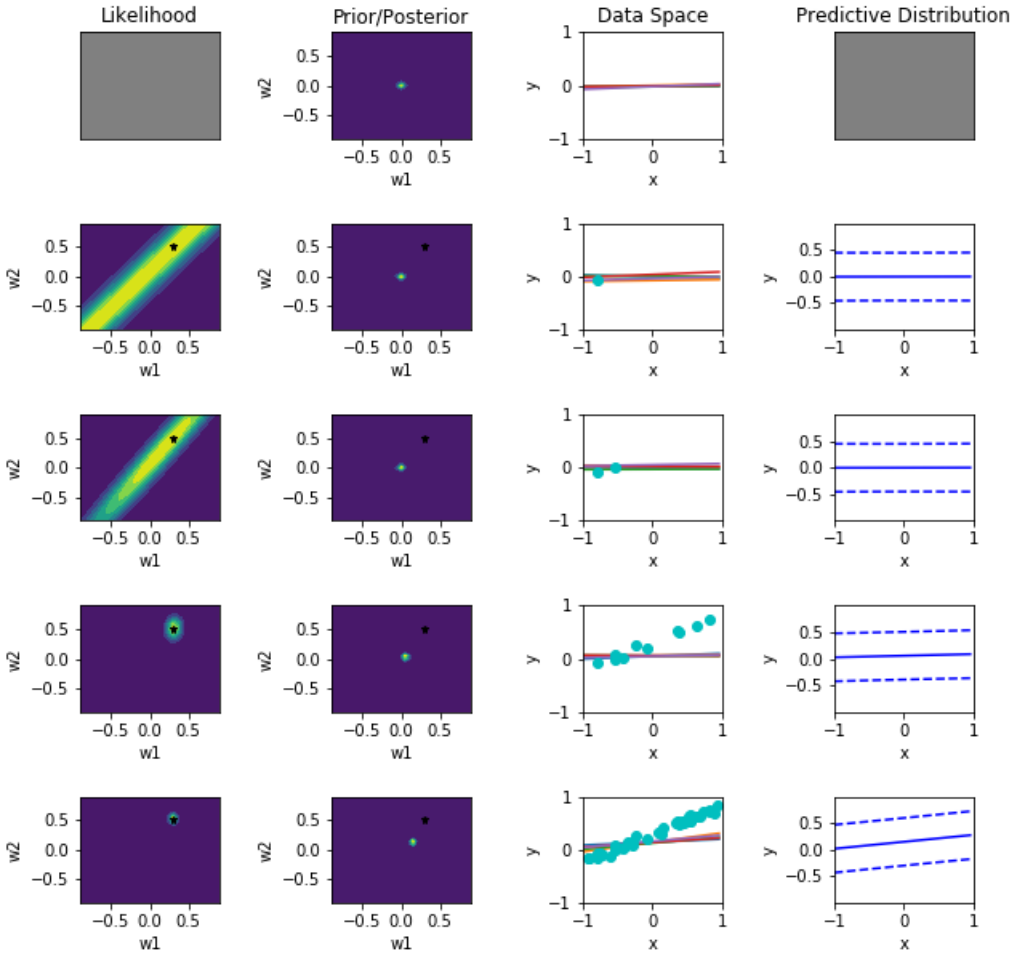
```python
def get_predictive_params(X_new, post_mean, post_var, likelihood_var = 0.2**2):
    '''
    Implement get_predictive_params. This function returns the predictive
    distribution parameters (mean and variance) given the posterior mean
    and covariance matrix (returned from get_posterior_params) and the
    likelihood variance (default value from lecture).

    Args:
        X_new: New observation (np.matrix object)
        post_mean, post_var: Returned from get_posterior_params
        likelihood_var: likelihood variance (0.2**2) per the lecture slides

    Returns:
        - pred_mean: Mean of predictive distribution
        - pred_var: Variance of predictive distribution
    '''

    # TO DO
    pred_mean=np.dot(post_mean.T,X_new)
    pred_var=np.sqrt(np.dot(np.dot(X_new.T,post_var),X_new)+likelihood_var)
    return pred_mean, pred_var
```

4. Run "python problem.py" from inside the Bayesian Regression directory to do the regression and generate the plots. This runs through the regression with three different settings for the prior covariance. You may want to change the default behavior in support_code.make_plots

from plt.show, to saving the plots for inclusion in your homework submission.




Likelihood
Prior/Posterior
Data Space
Predictive Distribution
w1
w2
x
y



5. Comment on your results. In particular, discuss how each of the following change with sample size and with the strength of the prior: (i) the likelihood function, (ii) the posterior distribution, and (iii) the posterior predictive distribution.

   (i): The strength of the prior does not change the likelihood function. Large sample size will limit the distribution of the likelihood function, restricting it in a smaller range.
   (ii): Stronger prior will limit the posterior in a smaller range, making the posterior harder to converge to the true distribution. Larger sample size helps more accurate predictions on the posterior distribution.

(iii): The stronger the prior is,less variance the predictive distribution has. Stronger prior will make it harder for the predictive distribution to converge. Larger sample size helps the predictive distribution to better converge to the actual distribution.

6. Our work above was very much "full Bayes", in that rather than coming up with a single prediction function, we have a whole distribution over posterior prediction functions. However, sometimes we want a single prediction function, and a common approach is to use the MAP estimate – that is, choose the prediction function that has the highest posterior likelihood. As we discussed in class, for this setting, we can get the MAP estimate using ridge regression. Use ridge regression to get the MAP prediction function corresponding to the first prior covariance ($\Sigma = \frac{1}{2}I$, per the support code). What value did you use for the regularization coefficient? Why?
In the get_posterior_params function, the default variance of the likelihood function were set to be $0.2^2 \implies \frac{0.2^2}{\lambda} = \frac{1}{2} \implies \lambda = 0.08$.

```
from sklearn.linear_model import Ridge
```

```
alpha = 0.08 # Change to the correct value
ridge = Ridge(alpha=alpha,
              fit_intercept=False,
              solver='cholesky')

ridge.fit(xtrain, ytrain)
```

```
Ridge(alpha=0.08, copy_X=True, fit_intercept=False, max_iter=None,
   normalize=False, random_state=None, solver='cholesky', tol=0.001)
```

If alpha is set correctly, ridge.coef_ will equal the prior mean/MAP estimate returned by the next two cells.

```
ridge.coef_
```

```
array([[0.30052135, 0.52406189]])
```

```
post[0]
```

```
matrix([[0.30052135],
        [0.52406189]])
```

6 [Optional] Coin Flipping: Maximum Likelihood

1. [Optional] Suppose we flip a coin and get the following sequence of heads and tails:

$$\mathcal{D} = (H, H, T)$$

Give an expression for the probability of observing $\mathcal{D}$ given that the probability of heads is θ. That is, give an expression for $p(\mathcal{D} \mid \theta)$. This is called the **likelihood of θ for the data $\mathcal{D}$**.
$p(D|\theta) = \theta^2(1-\theta)$

2. [Optional] How many different sequences of 3 coin tosses have 2 heads and 1 tail? If we toss the coin 3 times, what is the probability of 2 heads and 1 tail? (Answer should be in terms of θ.)
   There are 3 different sequences of 2 heads and 1 tail, HHT, HTH, THH. The probability of 2 heads and 1 tail is $\binom{3}{2}\theta^2(1-\theta) = 3\theta^2(1-\theta)$

3. [Optional] More generally, give an expression for the likelihood $p(\mathcal{D} \mid \theta)$ for a particular sequence of flips $\mathcal{D}$ that has n_h heads and n_t tails. Make sure you have expressions that make sense even for $\theta = 0$ and $n_h = 0$, and other boundary cases. You may use the convention that $0^0 = 1$, or you can break your expression into cases if needed.
   For a particular sequence of flips, $p(D|\theta) = \theta^{n_h}(1-\theta)^{n_t}$

4. [Optional] Show that the maximum likelihood estimate of θ given we observed a sequence with n_h heads and n_t tails is
   $$\hat{\theta}_{\text{MLE}} = \frac{n_h}{n_h + n_t}.$$
   You may assume that $n_h + n_t \geq 1$. (Hint: Maximizing the log-likelihood is equivalent and is often easier.)

   $$p(D|\theta) = \binom{n}{n_h}\theta^{n_h}(1-\theta)^{n_t}$$
   $$log(p(D|\theta)) = log\binom{n}{n_h} + n_h log\theta + n_t log(1-\theta)$$
   Take the derivative w.r.t θ and set the derivative $= 0, \hat{\theta_{MLE}} = \frac{n_h}{n_h + n_t}$

7 [Optional] Coin Flipping: Bayesian Approach with Beta Prior

We'll now take a Bayesian approach to the coin flipping problem, in which we treat θ as a random variable sampled from some prior distribution $p(\theta)$. We'll represent the ith coin flip by a random variable $X_i \in \{0, 1\}$, where $X_i = 1$ if the ith flip is heads. We assume that the X_i's are conditionally independent given θ. This means that the joint distribution of the coin flips and θ factorizes as follows:

$$\begin{aligned} p(x_1, \ldots, x_n, \theta) &= p(\theta)p(x_1, \ldots, x_n \mid \theta) \text{ (always true)} \\ &= p(\theta)\prod_{i=1}^{n} p(x_i \mid \theta) \text{ (by conditional independence).} \end{aligned}$$

1. [Optional] Suppose that our prior distribution on θ is Beta(h, t), for some $h, t > 0$. That is, $p(\theta) \propto \theta^{h-1}(1-\theta)^{t-1}$. Suppose that our sequence of flips $\mathcal{D}$ has n_h heads and n_t tails. Show that the posterior distribution for θ is Beta$(h + n_h, t + n_t)$. That is, show that
   $$p(\theta \mid \mathcal{D}) \propto \theta^{h-1+n_h}(1-\theta)^{t-1+n_t}.$$

We say that the Beta distribution is **conjugate** to the Bernoulli distribution since the prior and the posterior are both in the same family of distributions (i.e. both Beta distributions).

$$\begin{aligned} p(\theta \mid \mathcal{D}) &\propto p(D|\theta)p(\theta) \\ &= \theta^{h-1}(1-\theta)^{t-1}\theta^{n_h}(1-\theta^{n_t}) \\ &= \theta^{h-1+n_h}(1-\theta)^{t-1+n_t} \end{aligned}$$

2. [Optional] Give expressions for the MLE, the MAP, and the posterior mean estimates of θ. [Hint: You may use the fact that a Beta(h,t) distribution has mean $h/(h+t)$ and has mode $(h-1)/(h+t-2)$ for $h,t>1$. For the Bayesian solutions, you should note that as $h+t$ gets very large, and assuming we keep the ratio $h/(h+t)$ fixed, the posterior mean and MAP approach the prior mean $h/(h+t)$, while for fixed h and t, the posterior mean approaches the MLE as the sample size $n = n_h + n_t \to \infty$.

$$\begin{aligned} \hat{\theta}_{MLE} &= \frac{n_h}{n_h + n_t} \\ \hat{\theta}_{MAP} &= \frac{n_h + h - 1}{n_h + n_t + h + t - 2} \\ \hat{\theta}_{PosteriorMean} &= \frac{n_h + h}{n_h + n_t + h + t} \end{aligned}$$

3. [Optional] What happens to $\hat{\theta}_{\text{MLE}}$, $\hat{\theta}_{\text{MAP}}$, and $\hat{\theta}_{\text{POSTERIOR MEAN}}$ as the number of coin flips $n = n_h + n_t$ approaches infinity?

   When n approaches infinity, the effect of the prior is extremely small.$\hat{\theta_{MAP}}$ and $\hat{\theta}_{\text{POSTERIOR MEAN}}$ would converge to θ. The variance of θ_{MLE} would decrease as n increases, and θ_{MLE} would also converge to θ.

4. [Optional] The MAP and posterior mean estimators of θ were derived from a Bayesian perspective. Let's now evaluate them from a frequentist perspective. Suppose θ is fixed and unknown. Which of the MLE, MAP, and posterior mean estimators give **unbiased** estimates of θ, if any? [Hint: The answer may depend on the parameters h and t of the prior. Also, let's consider the total number of flips $n = n_h + n_t$ to be given (not random), while n_h and n_t are random, with $n_h = n - n_t$.]

   $\mathbb{E}(\hat{\theta}_{MLE}) = \mathbb{E}(\frac{n_h}{n}) = \frac{1}{n}\mathbb{E}(n_h) = \frac{n\theta}{n} = \theta$

   $\hat{\theta}_{MLE}$ is an unbiased estimator.

   $\mathbb{E}(\hat{\theta}_{MAP}) = \mathbb{E}(\frac{n_h + h - 1}{n + h + t - 2}) = \mathbb{E}(\frac{n\theta + h - 1}{n + h + t - 2})$

   $\hat{\theta}_{MAP}$ is only unbiased if $h = t = 1$.

   $\mathbb{E}(\hat{\theta}_{\text{POSTERIOR MEAN}}) = \mathbb{E}(\frac{n_h + h}{n + h + t}) = \mathbb{E}(\frac{n\theta + h}{n + h + t})$

   $\hat{\theta}_{\text{POSTERIOR MEAN}}$ is a biased estimator.

5. [Optional] Suppose somebody gives you a coin and asks you to give an estimate of the probability of heads, but you can only toss the coin 3 times. You have no particular reason to believe this is an unfair coin. Would you prefer the MLE or the posterior mean as a point estimate of θ? If the posterior mean, what would you use for your prior?

   I would prefer posterior mean because when $n = 3$, the variance of MLE is very high, and θ_{MLE} is unstable. I would choose beta(2,2).

8 [Optional] Hierarchical Bayes for Click-Through Rate Estimation

In mobile advertising, ads are often displayed inside apps on a phone or tablet device. When an ad is displayed, this is called an "impression." If the user clicks on the ad, that is called a "click." The probability that an impression leads to a click is called the "click-through rate" (CTR).

Suppose we have $d = 1000$ apps. For various reasons[5], each app tends to have a different overall CTR. For the purposes of designing an ad campaign, we want estimates of all the app-level CTRs, which we'll denote by $\theta_1, \ldots, \theta_{1000}$. Of course, the particular user seeing the impression and the particular ad that is shown have an effect on the CTR, but we'll ignore these issues for now. [Because so many clicks on mobile ads are accidental, it turns out that the overall app-level CTR often dominates the effect of the particular user and the specific ad.]

If we have enough impressions for a particular app, then the empirical fraction of clicks will give a good estimate for the actual CTR. However, if we have relatively few impressions, we'll have some problems using the empirical fraction. Typical CTRs are less than 1%, so it takes a fairly large number of observations to get a good estimate of CTR. For example, even with 100 impressions, the only possible CTR estimates given by the MLE would be $0\%, 1\%, 2\%, \ldots, 100\%$. The 0% estimate is almost certainly much too low, and anything 2% or higher is almost certainly much too high. Our goal is to come up with reasonable point estimates for $\theta_1, \ldots, \theta_{1000}$, even when we have very few observations for some apps.

If we wanted to apply the Bayesian approach worked out in the previous problem, we could come up with a prior that seemed reasonable. For example, we could use the following $\text{Beta}(3, 400)$ as a prior distribution on each θ_i:

In this basic Bayesian approach, the parameters 3 and 400 would be chosen by the data scientist based on prior experience, or "best guess", but without looking at the new data. Another approach would be to use the data to help you choose the parameters a and b in $\text{Beta}(a, b)$. This would **not** be a Bayesian approach, though it is frequently used in practice. One method in this direction is called **empirical Bayes**. Empirical Bayes can be considered a frequentist approach, in which we estimate a and b from the data $\mathcal{D}$ using some estimation technique, such as maximum likelihood. The proper Bayesian approach to this type of thing is called **hierarchical Bayes**, in which we put another prior distribution on a and b. We'll investigate each of these approaches below.

[5]The primary reason is that different apps place the ads differently, making it more or less difficult to avoid clicking the ad.

Mathematical Description

We'll now give a mathematical description of our model, assuming the prior parameters a and b are directly chosen by the data scientist. Let $n_1, \ldots, n_d$ be the number of impressions we observe for each of the d apps. In this problem, we will not consider these to be random numbers. For the ith app, let $c_i^1, \ldots, c_i^{n_i} \in \{0, 1\}$ be indicator variables determining whether or not each impression was clicked. That is, $c_i^j = 1(j\text{th impression on } i\text{th app was clicked})$. We can summarize the data on the ith app by $\mathcal{D}_i = (x_i, n_i)$, where $x_i = \sum_{j=1}^{n_i} c_i^j$ is the total number of impressions that were clicked for app i. Let $\theta = (\theta_1, \ldots, \theta_d)$, where θ_i is the CTR for app i.

In our Bayesian approach, we act as though the data were generated as follows:

1. Sample $\theta_1, \ldots, \theta_d$ i.i.d. from Beta(a, b).
2. For each app i, sample $c_i^1, \ldots, c_i^{n_i}$ i.i.d. from Bernoulli(θ_i).

8.1 [Optional] Empirical Bayes for a single app

We start by working out some details for Bayesian inference for a single app. That is, suppose we only have the data $\mathcal{D}_i$ from app i, and nothing else. Mathematically, this is exactly the same setting as the coin tossing setting above, but here we push it further.

1. Give an expression for $p(\mathcal{D}_i \mid \theta_i)$, the likelihood of $\mathcal{D}_i$ given the probability of click θ_i, in terms of θ_i, x_i and n_i.

$$p(D_i|\theta_i) = \theta_i^{x_i}(1-\theta_i)^{n_i - x_i}$$

2. We will take our prior distribution on θ_i to be Beta(a, b). The corresponding probability density function is given by

$$p(\theta_i) = \text{Beta}(\theta_i; a, b) = \frac{1}{B(a,b)}\theta_i^{a-1}\left(1-\theta_i\right)^{b-1},$$

where $B(a, b)$ is called the Beta function. Explain (without calculation) why we must have

$$\int \theta_i^{a-1}\left(1-\theta_i\right)^{b-1}\, d\theta_i = B(a,b).$$

$p(\theta_i)$ is a probability density function, so $\int p(\theta_i)d\theta = 1$. Thus $\int \theta_i^{a-1}\left(1-\theta_i\right)^{b-1}\, d\theta_i = B(a,b)$.

3. Give an expression for the posterior distribution $p(\theta_i \mid \mathcal{D}_i)$. In this case, include the constant of proportionality. In other words, do not use the "is proportional to" sign $\propto$ in your final expression. You **may** reference the Beta function defined above. [Hint: This problem is essentially a repetition of an earlier problem.]

$$\begin{aligned} p(\theta_i|D_i) =& \frac{p(D_i|\theta_i)p(\theta_i)}{p(D_i)} \\ &= \frac{\theta_i^{x_i}(1-\theta_i)^{n_i-x_i}\theta_i^{a-1}(1-\theta_i)^{b-1}}{B(a,b)p(D_i)} \\ &= \frac{\theta_i^{x_i+a-1}(1-\theta_i)^{n_i-x_i+b-1}}{B(a,b)p(D_i)} \end{aligned}$$

4. Give a closed form expression for $p(\mathcal{D}_i)$, the marginal likelihood of $\mathcal{D}_i$, in terms of the a, b, x_i, and n_i. You may use the normalization function $B(\cdot, \cdot)$ for convenience, but you should not have any integrals in your solution. (Hint: $p(\mathcal{D}_i) = \int p(\mathcal{D}_i \mid \theta_i)\, p(\theta_i)\, d\theta_i$, and the answer will be a ratio of two beta function evaluations.)

$$\begin{aligned} p(\mathcal{D}_i) &= \int p(\mathcal{D}_i \mid \theta_i)\, p(\theta_i)\, d\theta_i \\ &= \int \frac{\theta_i^{x_i+a-1}(1-\theta_i)^{n_i-x_i+b-1}}{B(a,b)} d\theta_i \\ &= \frac{B(x_i+a, n_i-x_i+b)}{B(a,b)} \end{aligned}$$

5. The maximum likelihood estimate for θ_i is x_i/n_i. Let $p_{\text{MLE}}(\mathcal{D}_i)$ be the marginal likelihood of $\mathcal{D}_i$ when we use a prior on θ_i that puts all of its probability mass at x_i/n_i. Note that

$$\begin{aligned} p_{\text{MLE}}(\mathcal{D}_i) &= p\left(\mathcal{D}_i \mid \theta_i = \frac{x_i}{n_i}\right) p\left(\theta_i = \frac{x_i}{n_i}\right) \\ &= p\left(\mathcal{D}_i \mid \theta_i = \frac{x_i}{n_i}\right). \end{aligned}$$

Explain why, or prove, that $p_{\text{MLE}}(\mathcal{D}_i)$ is larger than $p(\mathcal{D}_i)$ for any other prior we might put on θ_i. If it's too hard to reason about all possible priors, it's fine to just consider all Beta priors. [Hint: This does not require much or any calculation. It may help to think about the integral $p(\mathcal{D}_i) = \int p(\mathcal{D}_i \mid \theta_i)\, p(\theta_i)\, d\theta_i$ as a weighted average of $p(\mathcal{D}_i \mid \theta_i)$ for different values of θ_i, where the weights are $p(\theta_i)$.]

$$\begin{aligned} p(D_i) &= \int p(D_i|\theta_i)p(\theta_i)d\theta_i \\ &\leq \int p(D_i|\theta_{MLE})p(\theta)d\theta_i \\ &= p(D_i|\theta_{MLE}) \int p(\theta_i)d\theta \\ &= p(D_i|\theta_{MLE}) \end{aligned}$$

6. One approach to getting an **empirical Bayes** estimate of the parameters a and b is to use maximum likelihood. Such an empirical Bayes estimate is often called an **ML-2** estimate, since it's maximum likelihood, but at a higher level in the Bayesian hierarchy. To emphasize the dependence of the likelihood of $\mathcal{D}_i$ on the parameters a and b, we'll now write it as $p(\mathcal{D}_i \mid a, b)$[6]. The empirical Bayes estimates for a and b are given by

$$(\hat{a}, \hat{b}) = \underset{(a,b)\in(0,\infty)\times(0,\infty)}{\arg\max}\; p(\mathcal{D}_i \mid a, b).$$

To make things concrete, suppose we observed $x_i = 3$ clicks out of $n_i = 500$ impressions. A plot of $p(\mathcal{D}_i \mid a, b)$ as a function of a and b is given in Figure 1. It appears from this plot that

[6]Note that this is a slight (though common) abuse of notation, because a and b are not random variables in this setting. It might be more appropriate to write this as $p(\mathcal{D}_i; a, b)$ or $p_{a,b}(\mathcal{D}_i)$. But this isn't very common.

Figure 1: A plot of $p(\mathcal{D}_i \mid a, b)$ as a function of a and b.

the likelihood will keep increasing as a and b increase, at least if a and b maintain a particular ratio. Indeed, this likelihood function never attains its maximum, so we cannot use ML-2 here. Explain what's happening to the prior as we continue to increase the likelihood. [Hint: It is a property of the Beta distribution (not difficult to see), that for any $\theta \in (0, 1)$, there is a Beta distribution with expected value θ and variance less than ε, for any $\varepsilon > 0$. What's going in here is similar to what happens when you attempt to fit a gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ to a single data point using maximum likelihood.]

When keeping increase the likelihood, the effect of the prior on the posterior will also keep increasing and eventually dominate the posterior distribution.

8.2 [Optional] Empirical Bayes Using All App Data

In the previous section, we considered working with data from a single app. With a fixed prior, such as Beta(3,400), our Bayesian estimates for θ_i seem more reasonable to me[7] than the MLE when our sample size n_i is small. The fact that these estimates seem reasonable is an immediate consequence of the fact that I chose the prior to give high probability to estimates that seem reasonable to me, before ever seeing the data. Our earlier attempt to use empirical Bayes (ML-2) to choose the prior in a data-driven way was not successful. With only a single app, we were essentially overfitting the prior to the data we have. In this section, we'll consider using the data from all the apps, in which case empirical Bayes makes more sense.

1. Let $\mathcal{D} = (\mathcal{D}_1, \ldots, \mathcal{D}_d)$ be the data from all the apps. Give an expression for $p(\mathcal{D} \mid a, b)$, the **marginal likelihood** of $\mathcal{D}$. Expression should be in terms of a, b, x_i, n_i for $i = 1, \ldots, d$. Assume data from different apps are independent. (Hint: This problem should be easy, based on a problem from the previous section.)

$$p(D|a,b) = \Pi_{i=1}^{d} p(D_i|a,b) = \Pi_{i=1}^{d} \frac{B(x_i + a, n_i - x_i + b)}{B(a,b)}$$

2. Explain why $p(\theta_i \mid \mathcal{D}) = p(\theta_i \mid \mathcal{D}_i)$, according to our model. In other words, once we choose values for parameters a and b, information about one app does not give any information about other apps.

$$\begin{aligned} p(\theta_i|D) &= \frac{p(D|\theta_i)p(\theta_i)}{p(D)} \\ &= \frac{p(D_i|\theta_i)p(\theta_i)\Pi_{i\neq 1}^{d} p(D_k)}{\Pi_{i\neq 1}^{d} p(D_k)p(D_i)} \\ &= \frac{p(D_i|\theta_i)p(\theta_i)}{p(D_i)} \\ &= p(\theta_i \mid \mathcal{D}_i) \end{aligned}$$

[7] I say "to me", since I am the one who chose the prior. You may have an entirely different prior, and think that my estimates are terrible.

3. Suppose we have data from 6 apps. 3 of the apps have a fair number of impressions, and 3 have relatively few. Suppose we observe the following:

	Num Clicks	Num Impressions
App 1	50	10000
App 2	160	20000
App 3	180	60000
App 4	0	100
App 5	0	5
App 6	1	2

Compute the empirical Bayes estimates for a and b. (Recall, this amounts to computing $(\hat{a}, \hat{b}) = \arg\max_{(a,b)\in\mathbf{R}^{>0}\times\mathbf{R}^{>0}} p(\mathcal{D} \mid a, b)$.) This will require solving an optimization problem, for which you are free to use any optimization software you like (perhaps scipy.optimize would be useful). The empirical Bayes prior is then $\text{Beta}(\hat{a}, \hat{b})$, where $\hat{a}$ and $\hat{b}$ are our ML-2 estimates. Give the corresponding prior mean and standard deviation for this prior.

4. Complete the following table:

	NumClicks	NumImpressions	MLE	MAP	PosteriorMean	PosteriorSD
App 1	50	10000	0.5%			
App 2	160	20000	0.8%			
App 3	180	60000	0.3%			
App 4	0	100	0%			
App 5	0	5	0%			
App 6	1	2	50%			

Make sure to take a look at the PosteriorSD values and note which are big and which are small.

8.3 [Optional] Hierarchical Bayes

In Section 8.2 we managed to get empirical Bayes ML-II estimates for a and b by assuming we had data from multiple apps. However, we didn't really address the issue that ML-II, as a maximum likelihood method, is prone to overfitting if we don't have enough data (in this case, enough apps). Moreover, a true Bayesian would reject this approach, since we're using our data to determine our prior. If we don't have enough confidence to choose parameters for a and b without looking at the data, then the only proper Bayesian approach is to put another prior on the parameters a and b. If you are very uncertain about values for a and b, you could put priors on them that have high variance.

1. [Optional] Suppose P is the $\text{Beta}(a, b)$ distribution. Conceptually, rather than putting priors on a and b, it's easier to reason about priors on the mean m and the variance v of P. If we parameterize P by its mean m and the variance v, give an expression for the density function $\text{Beta}(\theta; m, v)$. You are free to use the internet to get this expression – just be confident it's correct. [Hint: To derive this, you may find it convenient to write some expression in terms of $\eta = a + b$.]

2. [Optional] Suggest a prior distribution to put on m and v. [Hint: You might want to use one of the distribution families given in this lecture.

3. [Optional] Once we have our prior on m and v, we can go "full Bayesian" and compute posterior distributions on $\theta_1, \ldots, \theta_d$. However, these no longer have closed forms. We would have to use approximation techniques, typically either a Monte Carlo sampling approach or a variational method, which are beyond the scope of this course[8]. After observing the data $\mathcal{D}$, m and v will have some posterior distribution $p(m, v \mid \mathcal{D})$. We can approximate that distribution by a point mass at the mode of that distribution $(m_{\text{MAP}}, v_{\text{MAP}}) = \arg\max_{m,v} p(m, v \mid \mathcal{D})$. **Give expressions** for the posterior distribution $p(\theta_1, \ldots, \theta_d \mid \mathcal{D})$, with and without this approximation. You do not need to give any explicit expressions here. It's fine to have expressions like $p(\theta_1, \ldots, \theta_d \mid m, v)$ in your solution. Without the approximation, you will probably need some integrals. It's these integrals that we need sampling or variational approaches to approximate. While one can see this approach as a way to approximate the proper Bayesian approach, one could also be skeptical and say this is just another way to determine your prior from the data. The estimators $(m_{\text{MAP}}, v_{\text{MAP}})$ are often called **MAP-II estimators**, since they are MAP estimators at a higher level of the Bayesian hierarchy.

[8] If you're very ambitious, you could try out a package like PyStan to see what happens.