Homework 10

Harvard University
Fall 2018
Instructors: Rahul Dave
Due Date: Sunday, November 18th, 2018 at 11:59pm

Instructions:

- Upload your final answers in the form of a Jupyter notebook containing all work to Canvas.
- Structure your notebook and your work to maximize readability.

Collaborators

Michelle (Chia Chi) Ho, Jiejun Lu, Jiawen Tong

In [1]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from IPython.display import display

import scipy.stats
from sklearn.model_selection import train_test_split
from theano import shared
import theano.tensor as tt
import pymc3 as pm
from pymc3 import Model
from pymc3.math import invlogit

%matplotlib inline
sns.set_style('whitegrid')
```

```
/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `flo
at` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

Question 1: Metropoflix and Chill (What's your Net Worth)?

coding required

Suppose we ask you to memorize the order of the top five movies on IMDB. When we quiz you on the order afterwards, you may not recall the correct order, but the mistakes you make in your recall can be modeled by simple probabilistic models.

Let's say that the top five movies are:

1. The Shawshank Redemption
2. The Godfather
3. The Godfather: Part II
4. Black Panther
5. Pulp Fiction

Let's represent this ordering by the vector $\omega = (1, 2, 3, 4, 5)$.

If you were to mistakenly recall the top five movies as:

1. The Godfather
2. The Godfather: Part II
3. Pulp Fiction
4. Black Panther
5. The Shawshank Redemption

We'd represent your answer by the vector $\theta = (2, 3, 5, 4, 1)$.

Unfortunately, your answer is wrong. Fortunately (for our purposes) we have a way of quantifying just how wrong. Define the Hamming distance between two top five rankings, θ, ω, as follows:

$$d(\theta, \omega) = \sum_{i=1}^{5} \mathbb{I}_{\theta_i \neq \omega_i},$$

where $\mathbb{I}_{\theta_i \neq \omega_i}$ is an indicator function that returns 1 if $\theta_i \neq \omega_i$, and 0 otherwise.

For example, the Hamming distance between your answer and the correct answer is $d(\theta, \omega) = 4$, because you only ranked Black Panther correctly.

Finally, let's suppose that the probability of giving a particular answer (expressed as θ) is modeled as

$$p(\theta \mid \omega, \lambda) \propto e^{-\lambda\, d(\theta,\, \omega)}$$

where λ can be thought of as an inverse temperature

1.1. Implement a Metropolis sampler to produce sample guesses from 500 individuals, with the λ values, $\lambda = 0.2, 0.5, 1.0$. What are the top five possible guesses?

1.2. Compute the probability that The Shawshank Redemption is ranked as the top movie (ranked number 1) by the Metropolis algorithm sampler. Compare the resulting probabilities for the various λ values.

1.3. How does λ affect the probability that The Shawshank Redemption is ranked as the top movie?

Gratuitous Titular Reference:

It's 2018 -- Even Wikipedia knows what Netflix and Chill (https://en.wikipedia.org/wiki/Netflix_and_chill) is about. (mixtape by Grime MC Merky ACE].

Drake's the type of dude (https://knowyourmeme.com/memes/drake-the-type-of) to not care about netflix and chill but about that net net net worth (https://youtu.be/DRS_PpOrUZ4?t=224)

Drake may wanna know if Kiki/KB (https://www.thefader.com/2018/10/24/real-kiki-drake-in-my-feelings-interview-kyanna-barber) is feeling him, but the NTSB (https://www.ntsb.gov) definitely isn't (https://www.cnn.com/2018/07/25/entertainment/ntsb-in-my-feelings/index.html)

Shout out Nawlins (https://riverbeats.life/neworleans/drake-shares-his-in-my-feelings) and Atlanta (http://www.thefader.com/2018/06/29/drake-sampled-atlanta-scorpion).

In [2]:
```python
# 1.1 metropolis sampling
def p(theta, w, lam):
    return np.exp(-lam * (theta!=w).sum())

def q_draw(theta):
    return np.random.permutation(theta)

def metropolis(p, q_draw, w, lam, N, start, burnin=0.1, thin=2):
    samples = np.empty((N, 5))
    x_prev = start
    accepted = np.zeros((N,))

    for i in range(N):
        x_star = q_draw(x_prev)
        p_star = p(x_star, w, lam)
        p_prev = p(x_prev, w, lam)
        pdf_ratio = p_star / p_prev
        if np.random.uniform() < min(1, pdf_ratio):
            samples[i] = x_star
            x_prev = x_star
            accepted[i] = 1
        else:
            samples[i, :] = x_prev
        if i % 100 == 0:
            print('i = {}'.format(i), end='\r')

    # throw away burnin and thin
    samples = samples[int(burnin*N)::thin]
    accepted_count = np.sum(accepted[int(burnin*N)::thin])

    return samples, accepted_count/len(samples)

```

In [3]:
```python
# draw samples from Metropolis
w = np.array([1, 2, 3, 4, 5])
lam_arr = np.array([0.2, 0.5, 1.0])
theta_0 = np.array([2, 3, 5, 4, 1])

target_N = 500
burnin = 0.1
thin = 50
N = int((target_N * thin) / (1 - burnin))
print('target_N = {}, N = {}'.format(target_N, N))

samples_dict = {}
for lam in lam_arr:
    samples_dict[lam] = {}
    samples, acp_rate = metropolis(p, q_draw, w, lam, N, theta_0, burnin=burnin, thin=thin)
    samples_dict[lam]['samples'] = samples
    samples_dict[lam]['acp_rate'] = acp_rate
    print('lambda = {} | acp_rate = {}'.format(lam, acp_rate))

```

```
target_N = 500, N = 27777
lambda = 0.2 | acp_rate = 0.886
lambda = 0.5 | acp_rate = 0.708
lambda = 1.0 | acp_rate = 0.374
```

In [4]:
```
print('top 5 guesses with counts')
for lam in lam_arr:
    lam_name = 'lambda = {}'.format(lam)
    df = pd.Series([np.str(s) for s in samples_dict[lam]['samples']], name=lam_name).to_frame()
    df['count'] = 1
    display(df.groupby(lam_name)['count'].sum().sort_values(ascending=False).head())
    print('--------------------')
```

```
top 5 guesses with counts

lambda = 0.2
[1. 4. 3. 2. 5.]    10
[5. 1. 4. 3. 2.]     9
[1. 2. 4. 5. 3.]     9
[1. 5. 4. 2. 3.]     9
[5. 2. 3. 4. 1.]     9
Name: count, dtype: int64

--------------------

lambda = 0.5
[1. 2. 3. 4. 5.]    30
[2. 1. 3. 4. 5.]    18
[1. 2. 3. 5. 4.]    12
[5. 2. 1. 4. 3.]    12
[1. 5. 2. 3. 4.]    10
Name: count, dtype: int64

--------------------

lambda = 1.0
[1. 2. 3. 4. 5.]    96
[3. 2. 1. 4. 5.]    20
[1. 2. 3. 5. 4.]    17
[5. 2. 3. 4. 1.]    16
[2. 3. 1. 4. 5.]    16
Name: count, dtype: int64

--------------------
```

In [5]:
```
# 1.2 probability of (theta[0] == 1)
for lam in lam_arr:
    samples = samples_dict[lam]['samples']
    print('lambda = {} | p(guess[0] = 1) = {}'.format(lam, (samples[:, 0]==1).mean()))
```

```
lambda = 0.2 | p(guess[0] = 1) = 0.236
lambda = 0.5 | p(guess[0] = 1) = 0.296
lambda = 1.0 | p(guess[0] = 1) = 0.48
```

Answer 1.3

Larger λ results in more punishment. Therefore, the larger λ values, the more correct guesses and the higher probability that The Shawshank Redemption is ranked as the top movie.

Question 2: In a Flash the Iris devient un Fleur-de-Lis.

coding required

We've done classification before, but the goal of this problem is to introduce you to the idea of classification using Bayesian inference.

Consider the famous Fisher flower Iris data set a multivariate data set introduced by Sir Ronald Fisher (1936) as an example of discriminant analysis. The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, you will build a model to predict the species.

For this problem only consider two classes: **virginica** and **not-virginica**.

The iris data can be obtained here (https://piazza.com/redirect/s3?bucket=uploads&prefix=attach%2Fjlo4e4ari3r4wd%2Fj9vjyzv62x149%2Fjoe90cwt3dla%2Firis.csv).

Let (X, Y) be our dataset, where $X = \{\vec{x}_1, \ldots \vec{x}_n\}$ and $\vec{x}_i$ is the standard feature vector corresponding to an offset 1 and the four components explained above. $Y \in \{0, 1\}$ are the scalar labels of a class. In other words the species labels are your Y data (virginica = 0 and virginica=1), and the four features -- petal length, petal width, sepal length and sepal width -- along with the offset make up your X data.

The goal is to train a classifier, that will predict an unknown class label $\hat{y}$ from a new data point x.

Consider the following glm (logistic model) for the probability of a class:

$$p(y) = \frac{1}{1 + e^{-x^T\beta}}$$

(or $logit(p) = x^T\beta$ in more traditional glm form)

where β is a 5D parameter to learn.

Then given p at a particular data point x, we can use a bernoulli likelihood to get 1's and 0's. This should be enough for you to set up your model in pymc3. (Note: You might want to set up p as a deterministic explicitly so that pymc3 does the work of giving you the trace).

2.1. Use a 60-40 stratified (preserving class membership) split of the dataset into a training set and a test set. (Feel free to take advantage of scikit-learn's `train_test_split`).

2.2. Choose a prior for $\beta \sim N(0, \sigma^2 I)$ and write down the formula for the posterior $p(\beta|Y, X)$. Since we dont care about regularization here, just use the mostly uninformative value $\sigma = 10$.

2.3. Find the MAP for the posterior on the training set.

2.4. Implement a PyMC3 model to sample from this posterior of β.

2.5. Generate 5000 samples of β. Visualize the betas and generate a traceplot and autocorrelation plots for each beta component.

2.6. Based on your samples construct an estimate for the posterior mean.

2.7. Select at least 2 datapoints and visualize a histogram of the posterior probabilities. Denote the posterior mean and MAP on your plot for each datapoint

Although having the posterior probabilities is nice, they are not enough. We need to think about how to make predictions based on our machinery. If we define the following:

- p_{MEAN} : using the posterior mean betas to generate probabilities for each data point
- p_{MAP} : using the posterior MAP betas to generate probabilities for each data point
- p_{CDF} : using the fraction of your posterior samples have values above 0.5 for each data point
- p_{PP}: using the fraction of 1s out of the samples drawn from the posterior predictive distribution for each data point

2.8. Plot the distributions of p_{MEAN} , p_{CDF} , p_{MAP} and p_{PP} over all the data points in the training set. How are these different?

How do we turn these probabilities into predictions? There are two ways to make these predictions, given an estimate of $p(y = 1 \mid x)$:

- Sample from the Bernoulli likelihood at the data point x to decide if that particular data points classification $y(x)$ should be a 1 or a 0.
- Do the intuitive "machine-learning-decision-theoretic" (MLDT) thing and you assign a data point x a classification 1 if $p(y = 1|x) > 0.5$.

2.9. Plot the posterior-predictive distribution of the misclassification rate with respect to the true class identities $y(x)$ of the data points x (in other words you are plotting a histogram with the misclassification rate for the n_{trace} posterior-predictive samples) on the training set.

2.10. For every posterior sample, consider whether the data point ought to be classified as a 1 or 0 from the $p > 0.5 \implies y = 1$ decision theoretic prespective. Using the MLDT defined above, overlay a plot of the histogram of the misclassification rate for the posterior on the corresponding plot for the posterior-predictive you constructed in 2.9. Which case (from posterior-predictive or from-posterior) has a wider mis-classification distribution?

2.11. Repeat 2.9 and 2.10 for the test set (i.e. make predictions). Describe and interpret the widths of the resulting distributions.

Gratuitous Titular References:

The Iris Dataset (https://en.wikipedia.org/wiki/Iris_flower_data_set) was introduced by Ronald Fisher as part of a famous article (https://onlinelibrary.wiley.com/doi/epdf/10.1111/j.1469-1809.1936.tb02137.x) introducing LDA (https://en.wikipedia.org/wiki/Linear_discriminant_analysis).

The three iris variants in the dataset were at the time difficult to tell apart morphologically (https://www.jstor.org/stable/2394164?seq=1#page_scan_tab_contents)

While the origin of the Fleur-de-Lis is debated (https://www.heraldica.org/topics/fdl.htm), it is most likely an Iris florentina (https://en.wikipedia.org/wiki/Iris_florentina) or Iris pseudacorus (https://en.wikipedia.org/wiki/Iris_pseudacorus) but not a lily flower (https://www.collinsdictionary.com/dictionary/english-french/lily).

Iris West (https://en.wikipedia.org/wiki/Iris_West) is a love interest of Barry Allen (https://en.wikipedia.org/wiki/Flash_(Barry_Allen) one of the main incarnations of The Flash (https://en.wikipedia.org/wiki/Flash_(comics). Coming from Central City (https://en.wikipedia.org/wiki/Central_City_(DC_Comics) she is most likely classified as **not-virginica**.

In [6]:
```
# read data
df_iris = pd.read_csv('iris.csv')

# add class label 0/1
df_iris['Y'] = df_iris['class'].apply(lambda x: int(x.split('-')[-1] == 'virginica'))

# add ones
df_iris['const'] = np.ones((len(df_iris),))
df_iris.head()
```

Out[6]:

	sepal_length	sepal_width	petal_length	petal_width	class	Y	const
0	5.1	3.5	1.4	0.2	Iris-setosa	0	1.0
1	4.9	3.0	1.4	0.2	Iris-setosa	0	1.0
2	4.7	3.2	1.3	0.2	Iris-setosa	0	1.0
3	4.6	3.1	1.5	0.2	Iris-setosa	0	1.0
4	5.0	3.6	1.4	0.2	Iris-setosa	0	1.0

In [7]:
```
# extract features
features = ['const'] + df_iris.columns.values[:4].tolist()
print('feature dimension = {}'.format(len(features)))

# 2.1 stratified train/test split
X_train, X_test, y_train, y_test = train_test_split(df_iris[features], df_iris['Y'], stratify=df_iris['Y'],
                                                    test_size=0.4, random_state=100)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
feature dimension = 5
```

Out[7]:
```
((90, 5), (90,), (60, 5), (60,))
```

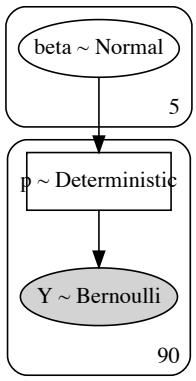
Answer 2.2

$$
\begin{aligned}
p(\beta|Y, X) &\propto p(\beta)p(Y, X|\beta) \\
&= \frac{1}{\sqrt{2\pi|\sigma^2 I|}} \exp\left\{\beta^T(\sigma^2 I)^{-1}\beta\right\} \left(\prod_i^N p(y_i = 1)^{y_i} p(y_i = 0)^{1-y_i}\right)
\end{aligned}
$$

In [8]:
```
X_shared = shared(X_train.values)

with Model() as iris_model:
    # Model parameter
    beta = pm.Normal('beta', mu=0, sd=10, shape=len(features))

    # Calculate probabilities of p(Y=1)
    p = pm.Deterministic('p', invlogit(tt.dot(X_shared, beta)))

    # Data likelihood
    Y = pm.Bernoulli('Y', p=p, observed=y_train.values)
```

In [9]:
```
pm.model_to_graphviz(iris_model)
```

Out[9]:



In [10]:
```
# 2.3 parameters MAP estimates
map_params = pm.find_MAP(model=iris_model)
map_params['beta']
```

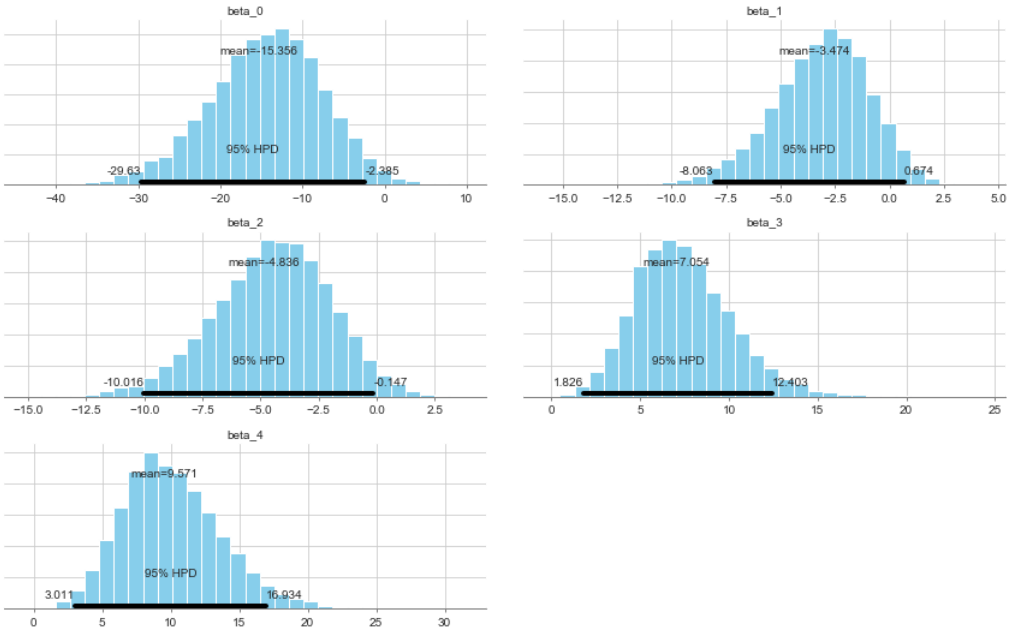
```
logp = -24.004, ||grad|| = 0.19822: 100%|██████████| 40/40 [00:00<00:00, 1299.50it/s]
```

Out[10]:
```
array([-12.67971932,  -2.59690798,  -3.84718621,   5.41561429,
         7.82401923])
```

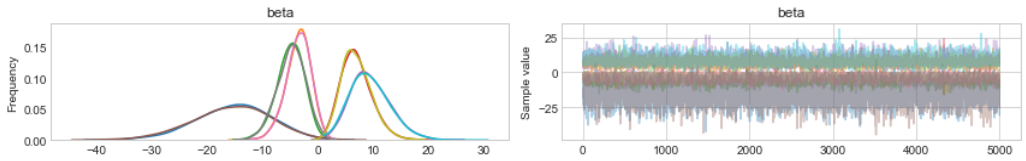
In [11]:
```
# 2.4 sample with posteior betas
with iris_model:
    train_trace = pm.sample(5000, step=pm.NUTS(), start=map_params)
```

```
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [beta]
Sampling 2 chains: 100%|██████████| 11000/11000 [01:29<00:00, 123.10draws/s]
There were 3 divergences after tuning. Increase `target_accept` or reparameterize.
```

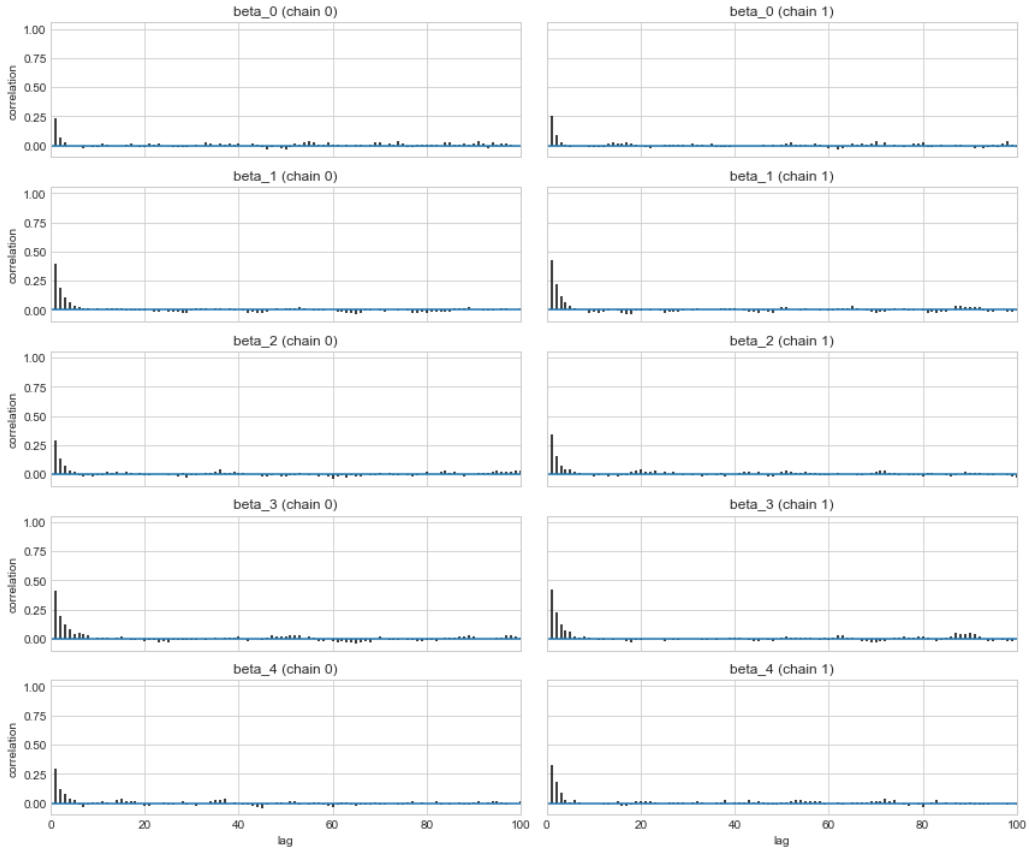
In [12]:
```
# 2.5 visualize beta posteriors
pm.plot_posterior(train_trace, varnames=['beta'])
plt.tight_layout()
```



In [13]:
```
# 2.5 visualize beta posteriors - histogram & trace
pm.traceplot(train_trace, varnames=['beta'])
plt.tight_layout()
```



In [14]:
```
# 2.5 visualize beta posteriors - autocorrelations
pm.autocorrplot(train_trace, varnames=['beta'])
plt.tight_layout()
```



In [15]:
```
# 2.6 beta posterior mean
beta_post_mean = train_trace.get_values('beta').mean(axis=0)
beta_post_mean
```

Out[15]:
```
array([-15.35552161,  -3.47359305,  -4.83596425,   7.05410305,
         9.57053982])
```

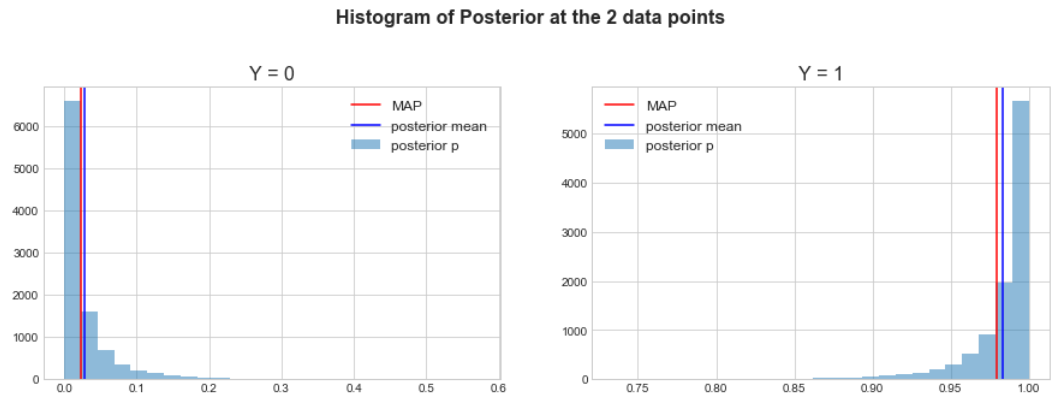
In [16]:
```
train_trace.get_values('beta').shape
```

Out[16]:
```
(10000, 5)
```

In [17]:
```
# 2.7 Plot posterior distribution, mean & MAP at 2 data points
y_train.head()
```

Out[17]:
```
111    1
46     0
90     0
40     0
92     0
Name: Y, dtype: int64
```

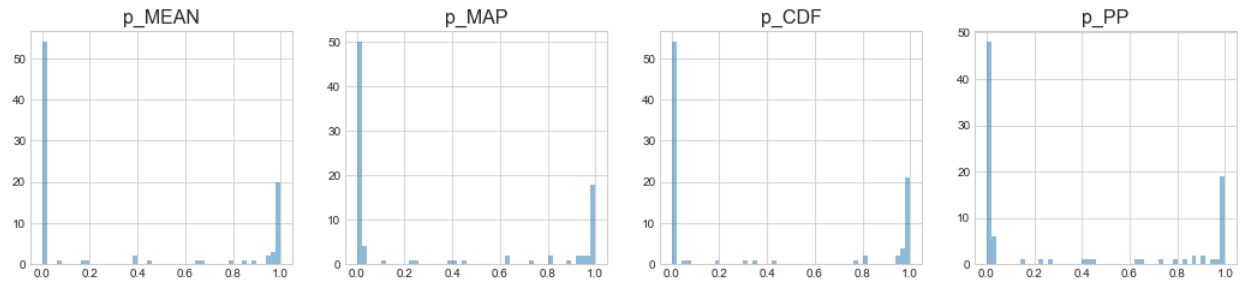
In [18]:
```
idx_y1 = 0 # select 1 data point with y=1
idx_y0 = 2 # select 1 data point with y=0

fig, ax = plt.subplots(1, 2, figsize=(15, 5))
plt.suptitle('Histogram of Posterior at the 2 data points', fontsize=16, weight='heavy')
plt.subplots_adjust(top=0.8)

ax[0].hist(train_trace['p'][:, idx_y0], alpha=0.5, bins=25, label='posterior p')
ax[0].axvline(map_params['p'][idx_y0], c='r', label='MAP')
ax[0].axvline(train_trace['p'][:, idx_y0].mean(), c='b', label='posterior mean')
ax[0].set_title('Y = 0', fontsize=16)
ax[0].legend(fontsize=12)

ax[1].hist(train_trace['p'][:, idx_y1], alpha=0.5, bins=25, label='posterior p')
ax[1].axvline(map_params['p'][idx_y1], c='r', label='MAP')
ax[1].axvline(train_trace['p'][:, idx_y1].mean(), c='b', label='posterior mean')
ax[1].set_title('Y = 1', fontsize=16)
ax[1].legend(fontsize=12)

plt.show()
```



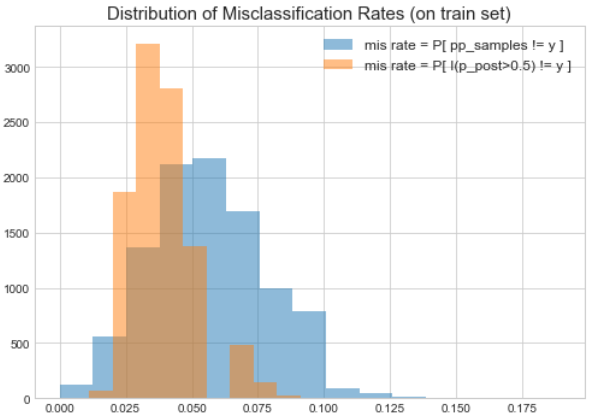
In [19]:
```
# 2.8 distribution of p's from different beta estimates
y_train_PP = pm.sample_ppc(train_trace, model=iris_model, samples=10000)

p_MEAN_train = 1 / (1 + np.exp(-X_train.dot(beta_post_mean)))
p_MAP_train = 1 / (1 + np.exp(-X_train.dot(map_params['beta'])))
p_CDF_train = (train_trace['p'] > 0.5).mean(axis=0)
p_PP_train = y_train_PP['Y'].mean(axis=0)
p_dict_train = {'p_MEAN': p_MEAN_train, 'p_MAP': p_MAP_train, 'p_CDF': p_CDF_train, 'p_PP': p_PP_train}
```

```
100%|██████████| 10000/10000 [00:06<00:00, 1627.93it/s]
```

In [20]:
```
fig, ax = plt.subplots(1, 4, figsize=(18, 4))
plt.suptitle(r'Distribution of p from Different $\beta$ Estimates (on train set)', fontsize=16, weight='heavy')
plt.subplots_adjust(top=0.8)
for i, (k, v) in enumerate(p_dict_train.items()):
    ax[i].hist(v, alpha=0.5, bins=50)
    ax[i].set_title(k, fontsize=16)
plt.show()
```



In [21]:
```
# 2.9 2.10 distribution of misclassification rates
mis_pp_train = (y_train_PP['Y'] != np.tile(y_train, [10000, 1])).mean(axis=1)
mis_p_post_train = ((train_trace['p']>0.5) != np.tile(y_train, [10000, 1])).mean(axis=1)

fig, ax = plt.subplots(1, 1, figsize=(7, 5))
ax.hist(mis_pp_train, alpha=0.5, bins=15, label='mis rate = P[ pp_samples != y ]')
ax.hist(mis_p_post_train, alpha=0.5, bins=10, label='mis rate = P[ I(p_post>0.5) != y ]')
ax.set_title('Distribution of Misclassification Rates (on train set)', fontsize=15)
plt.legend(fontsize=12)
plt.tight_layout()
```



Answer 2.9 2.10

On the **training** set, the distribution of misclassification rates from posterior predictive samples is wider than that from using β posteriors.

In [22]:
```
# posterior predictive samples on test set
X_shared.set_value(X_test.values)
y_test_PP = pm.sample_ppc(train_trace, model=iris_model, samples=10000)
y_test_PP['Y'].shape
```

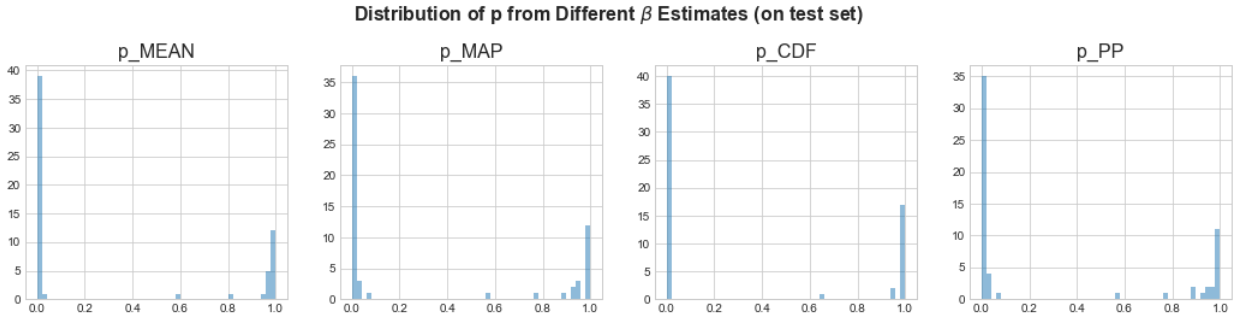
```
100%|██████████| 10000/10000 [00:07<00:00, 1340.20it/s]
```

Out[22]:
```
(10000, 60)
```

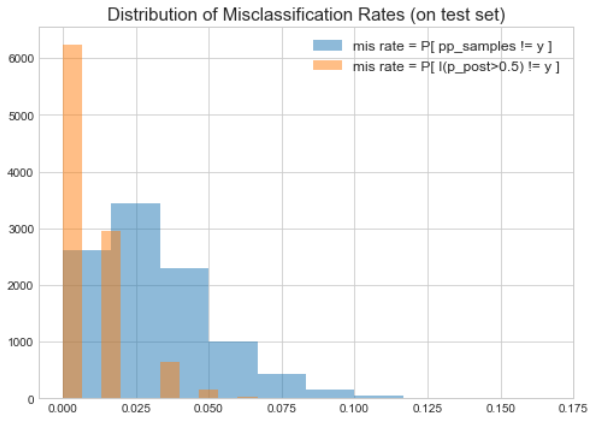
In [23]:
```
# X_test.dot(beta_posterior)
p_post_test = 1 / (1 + np.exp(-X_test.dot(train_trace['beta'].T).T))
p_post_test.shape
```

Out[23]:
```
(10000, 60)
```

In [24]:
```
# 2.11 distribution of misclassification rates - test set
p_MEAN_test = 1 / (1 + np.exp(-X_test.dot(beta_post_mean)))
p_MAP_test = 1 / (1 + np.exp(-X_test.dot(map_params['beta'])))
p_CDF_test = (p_post_test > 0.5).mean(axis=0)
p_PP_test = y_test_PP['Y'].mean(axis=0)
p_dict_test = {'p_MEAN': p_MEAN_test, 'p_MAP': p_MAP_test, 'p_CDF': p_CDF_test, 'p_PP': p_PP_test}

fig, ax = plt.subplots(1, 4, figsize=(18, 4))
plt.suptitle(r'Distribution of p from Different $\beta$ Estimates (on test set)', fontsize=16, weight='heavy')
plt.subplots_adjust(top=0.8)
for i, (k, v) in enumerate(p_dict_test.items()):
    ax[i].hist(v, alpha=0.5, bins=50)
    ax[i].set_title(k, fontsize=16)
plt.show()
```



In [25]:
```
mis_pp_test = (y_test_PP['Y'] != np.tile(y_test, [10000, 1])).mean(axis=1)
mis_p_post_test = ((p_post_test>0.5) != np.tile(y_test, [10000, 1])).mean(axis=1)

fig, ax = plt.subplots(1, 1, figsize=(7, 5))
ax.hist(mis_pp_test, alpha=0.5, label='mis rate = P[ pp_samples != y ]')
ax.hist(mis_p_post_test, alpha=0.5, label='mis rate = P[ I(p_post>0.5) != y ]')
ax.set_title('Distribution of Misclassification Rates (on test set)', fontsize=15)
plt.legend(fontsize=12)
plt.tight_layout()
```



Answer 2.11

Similarly, on the **test** set, the distribution of misclassification rates from posterior predictive samples is wider than that from using β posteriors.

Question 3 - Our Yelp Restaurant Review is in and the Fish is So Raw!

no coding required

In this course, we've spent a lot of time learning algorithms for performing inference on complex models. We've also spent time using these models to make decisions regarding our data. But in nearly every assignment, the model for the data is specified in the problem statement. In real life, the creative and, arguably, much more difficult task is to start with a broadly defined goal and then to customize or create a model which will meet this goal in some way.

This homework problem is atypical in that it does not involve any programming or (necessarily) difficult mathematics/statistics. The process of answering these questions seriously will however give you an idea of how one might create or select a model for a particular application and your answers will help you with formalizing the model if and when you're called upon to do so.

Grading: We want you to make a genuine effort to mold an ambiguous and broad real-life question into a concrete data science or machine learning problem without the pressure of getting the "right answer". As such, we will grade your answer to this homework question on a pass/fail basis. Any reasonable answer that demonstrates actual effort will be given a full grade.

We've compiled for you a fairly representative selection of Yelp reviews (https://piazza.com/redirect/s3?bucket=uploads&prefix=attach%2Fjlo4e4ari3r4wd%2Fj9vjyzv62x149%2Fjoe92vh7ni6e%2Fyelp_reviews.zip for a (now closed)) sushi restaurant called Ino's Sushi in San Francisco. Read the reviews and form an opinion regarding the various qualities of Ino's Sushi. Answer the following:

Answers for Question 3

3.1. If the task is to summarize the quality of a restaurant in a simple and intuitive way, what might be problematic with simply classifying this restaurant as simply "good" or "bad"? Justify your answers with specific examples from the dataset.

Answer 3.1

The ratings distribution is very polarized (either 1-star or 5-star) and thus the entire set of reviews cannot be simply represented by 1 aggregated value such as mean to justify whether it is good or bad.

3.2. For Ino's Sushi, categorize the food and the service, separately, as "good" or "bad" based on all the reviews in the dataset. Be as systematic as you can when you do this.

(**Hint:** Begin by summarizing each review. For each review, summarize the reviewer's opinion on two aspects of the restaurant: food and service. That is, generate a classification ("good" or "bad") for each aspect based on what the reviewer writes.)

Answer 3.2

Review id	Food	Service
01	good	bad
02	good	bad
03	N/A	bad
04	good	bad

Review id	Food	Service
05	good	good
06	good	bad
07	good	N/A
08	good	N/A
09	good	bad
10	good	good

3.3. Identify statistical weaknesses in breaking each review down into an opinion on the food and an opinion on the service. That is, identify types of reviews that make your method of summarizing the reviewer's optinion on the quality of food and service problemmatic, if not impossible. Use examples from your dataset to support your argument.

Answer 3.3

- Since reviews are written by human subjects, they are not guaranteed to write about their opinion on both food and service. However, by breaking each review down, we would have missingness in one of the responses when a review only focuses on one aspect. For example, Review 3 only focuses on service while Review 7 and 8 only focus on food.
- Different users have different priors for reviews, i.e. "amazing" means different things for different people.

3.4. Identify all the ways in which the task in 3.2 might be difficult for a machine to accomplish. That is, break down the classification task into simple self-contained subtasks and identify how each subtask can be accomplished by a machine (i.e. which area of machine learning, e.g. topic modeling, sentiment analysis etc, addressess this type of task).

Answer 3.4

- Segregate the contents for food and service & decide the classification label: good vs bad:

  > - manual label by humans, or
  > - use pretrained models to predict the label (check by humans if necessary)

- Extract review texts to features for topic modeling, for example:

  > - word count vectors for Latent Dirichlet Allocation modeling, or
  > - other generic tokenized strings for natural language processing

---

3.5. Now let us think of a different problem, a regression problem in which our aim is to predict and do inference on what rating a given user of Yelp might give a particular restaurant. How might you estimate the across-user quality of a restaurant from data? And how might you estimate the across-restaurant curmudgeonlyness of a user?

Answer 3.5

Assuming that food and service ratings have already extracted from the reviews, we would use a hierarchical model where each user has a prior θ_i and a review rating $y_i \sim f(\theta_i)$. The procedures we propose are:

- For each user, gather all reviews written by him/her for all restaurants
- Compute the prior parameter of this user. For example, we model the mean rating of all users came from a normal distribution $\theta_i \sim \mathcal{N}(\mu_0, \sigma_0)$
- The specific review ratings of user i, ($\vec{y_i}$, a 2D vector representing food and service ratings), came from another normal distribution $y_{ij} \sim \mathcal{N}(\theta_i, \sigma_1)$

(Assuming σ_0 and σ_1 are given fixed parameters.)

3.6 Additionally, consider a "space of latent factors" where aspects of the user's taste interact with aspects of the restaurant. An example of such a factor might be the user's propensity to get emotional after having the perfect filet-mignon. How might you combine this information with that in 3.5 to improve your prediction and inference?

Answer 3.6

- For each user, augment the user prior parameters to 2D, (θ_{i0}, θ_{i1}), where θ_{i0} is the mean rating for a non-emotional user and θ_{i1} is the mean rating for an emotional user. (θ_{i0}, θ_{i1}) came from a joint normal distribution:

$$\begin{pmatrix} \theta_{i0} \\ \theta_{i1} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_0 \\ \mu_1 \end{pmatrix}, \Sigma_0 \right)$$

- The specific review ratings of user i, $\vec{y_i} = (\vec{y_i^0}, \vec{y_i^1})$, would be a 2 x 2D vector representing food and service ratings under non-emotional and emotional scale, each of which independently came from a normal distribution:

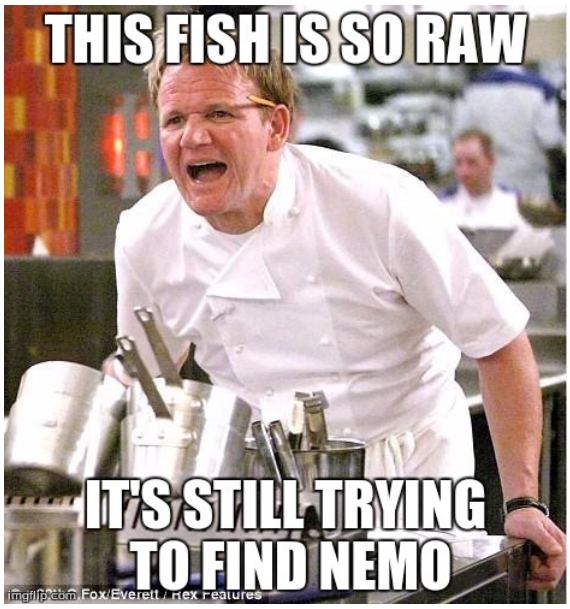
$$y_{ij}^0 \sim \mathcal{N}(\theta_{i0}, \sigma_1)$$

$$y_{ij}^1 \sim \mathcal{N}(\theta_{i1}, \sigma_2)$$

(Assuming Σ_0, σ_1 and σ_2 are given fixed parameters.)

Gratuitous Titular Reference:

Sushi is not raw fish (http://www.todayifoundout.com/index.php/2011/12/sushi-is-not-raw-fish)



More Gordon Ramsey memes (https://knowyourmeme.com/memes/people/gordon-ramsay)