1

$$p(w|D) = \frac{p(w)p(D|w)}{p(D)} \propto p(w)p(D|w) = p(w)exp(-NLL_D(w))$$

2

$$P(w|D) \propto \frac{1}{\sqrt{2\pi|\sum|}}exp(-\frac{1}{2}w^T(\sum)^{-1}w + \frac{y-1}{2}x^Tw - log(1+e^{-x^Tw}))$$
$$= \frac{1}{\sqrt{2\pi|\sum|}(1+e^{e^{-x^Tw}})}exp(-\frac{1}{2}w^T(\sum)^{-1}w + \frac{y-1}{2}x^Tw)$$
$$\propto \frac{1}{(1+e^{e^{-x^Tw}})}exp(-\frac{1}{2}w^T(\sum)^{-1}w + \frac{y-1}{2}x^Tw)$$

Not in Gaussian Family

3

equal with get

$$argmin(-log(exp(-NLL_D(w))p(w))))$$
$$NLL_D(w) = -\sum_{i=1}^{n}(y_ilogf(w^Tx_i) + (1-y_i)log[1-f(w^Tx_i)])$$
$$f(w^Tx_xi) = \frac{1}{1+e^{-w^Tx}}$$

=>

$$argmin(\frac{1}{n}\sum_{i=1}^{n}log(1+exp(-yw^Tx)) + \frac{1}{2n}w^T(\sum)^{-1}w)$$

=>

$$\lambda||w||^2 = \frac{1}{2n}w^T\sum^{-1}w$$
$$\sum = \frac{1}{2n\lambda}I$$

4

If $w \sim N(0, I)$, then $\sum = I, \lambda = \frac{1}{2n}$

5

$$L(\theta_1,\theta_2) = p(D_r, D_c|\theta_1,\theta_2) = p(D_r|\theta_1,\theta_2)p(D_c|\theta_1,\theta_2) = (\theta_1\theta_2)^{n_h}(1-\theta_1\theta_2)^{n_t}\theta_1^{c_h}(1-\theta_1)^{c_t}$$
$$\frac{\partial L(\theta_1,\theta_2)}{\partial\theta_2} = (1-\theta_1)^{c_t}\theta_1^{c_h}(\theta_1\theta_2)^{n_h}(1-\theta_1\theta_2)^{n_t}((\theta_1n_t+\theta_1n_h)\theta_2 - h_n) = 0$$
$$\frac{\partial L(\theta_1,\theta_2)}{\partial\theta_1} = (c_h+c_t)\theta_1 - c_h = 0$$
$$\theta_1 = \frac{c_h}{N_c}, \theta_2 = \frac{N_cn_h}{N_rc_h}$$

6

$P(\theta_1) = Beta(h,t) = \theta_1^{h-1} \ (1-\theta_1)^{\ t-1}$

MAP: $\hat{\theta} = argmax_{\theta_1} p(\theta_1|D_c)$

$$p(\theta_1|D_c) \propto P(D_c|\theta_1)P(\theta_1) = \theta_1^{c_h}(1-\theta_1)^{c_t}\theta_1^{h-1} \ (1-\theta_1)^{\ t-1} = \theta_1^{c_h+h-1}(1-\theta_1)^{c_t+t-1}$$
$$\frac{\partial}{\partial\theta_1} = 0$$
$$\theta_1 = \frac{c_h+h-1}{N_c+h+t-2}, \theta_2 = \frac{(N_c+h+t-2)n_h}{(c_h+h-1)N_r}$$

7

The pointwise maximum of $max f_1(x), .. f_m(x)$ is convex, the every norm on R^n is convex,

for w_1, w_2, $f(\theta w_1 + (1-\theta)w_2) =< \theta w_1 + (1-\theta)w_2, \Psi(x_i, y) - \Psi(x_i, y_i) >$

$$\theta < w_1, \Psi(x_i, y) - \Psi(x_i, y_i) > + \ (1-\theta) \ < w_2, \Psi(x_i, y) - \Psi(x_i, y_i) >$$
$$<= \theta f(w_1) + (1-\theta) f(w_2)$$

f(x) is convex, so J(w) is convex.

8

$$\partial f_i(x_0) \in \partial f(x_0), y = \hat{y_i}$$
$$\frac{\partial}{\partial w}[\Delta(y_i, \hat{y_i}) + < w, \Psi(x_i, y_i) - \Psi(x_i, y_i) >] = \Psi(x_i, y_i) - \Psi(x_i, y_i)$$
$$g = 2\lambda w^T + \frac{1}{n}\sum_{i=0}^{n} \Psi(x_{i+j}, \hat{y_{i+j}}) - \Psi(x_{i+j}, y_{i+j})$$

9

$$\partial J(w) = 2\lambda w^T + \Psi(x_i, \hat{y_i}) - \Psi(x_i, y_i)$$

10

$$\partial J(w) = 2\lambda w^T + \frac{1}{m}\sum_{j=0}^{m} \Psi(x_{i+j}, \hat{y_{i+j}}) - \Psi(x_{i+j}, y_{i+j})$$

Optional

if y = y'

- when $y = 1, y' = 1$

$$\Delta(y, y') = 0, h(x, y') = \frac{g(x)}{2}, h(x, y) = \frac{g(x)}{2}$$
$$\ell(h, (x, y)) = max[0 + \frac{g(x)}{2} - \frac{g(x)}{2}] = 0$$

- when $y = -1, y' = -1$

$$\Delta(y,y')=0, h(x,y')=\frac{-g(x)}{2}, h(x,y)=\frac{-g(x)}{2}$$
$$\ell(h,(x,y))=max[0+\frac{-g(x)}{2}-\frac{-g(x)}{2}]=0$$

if y != y'

- when $y=1, y'=-1$

$$\Delta(y,y')=1, h(x,y')=\frac{-g(x)}{2}, h(x,y)=\frac{g(x)}{2}$$
$$\ell(h,(x,y))=max[1+\frac{-g(x)}{2}-\frac{g(x)}{2}]=1-g(x)=1-yg(x)$$

- when $y=-1, y'=1$

$$\Delta(y,y')=1, h(x,y')=\frac{g(x)}{2}, h(x,y)=\frac{-g(x)}{2}$$
$$\ell(h,(x,y))=max[1+\frac{g(x)}{2}-\frac{-g(x)}{2}]=1+g(x)=1-yg(x)$$

11

```
from sklearn.base import BaseEstimator, ClassifierMixin, clone


class OneVsAllClassifier(BaseEstimator, ClassifierMixin):

    def __init__(self, estimator, n_classes):
        self.n_classes = n_classes
        self.estimators = [clone(estimator) for _ in range(n_classes)]
        self.fitted = False

    def fit(self, X, y=None):
        for i, estimator in enumerate(self.estimators):
            y_mat = [1 if _class == i else 0 for _class in y]
            estimator.fit(X, y_mat)
        self.fitted = True
        return self

    def decision_function(self, X):
        if not self.fitted:
            raise RuntimeError(
                "You must train classifer before predicting data.")

        if not hasattr(self.estimators[0], "decision_function"):
            raise AttributeError(
                "Base estimator doesn't have a decision_function attribute.")

        score = np.zeros((len(X), self.n_classes))
        for i, estimator in enumerate(self.estimators):
            score[:, i] = estimator.decision_function(X)
        return score

    def predict(self, X):
```

```
        score = self.decision_function(X)
        return np.argmax(score, axis=1)
```

12

oeffs 0

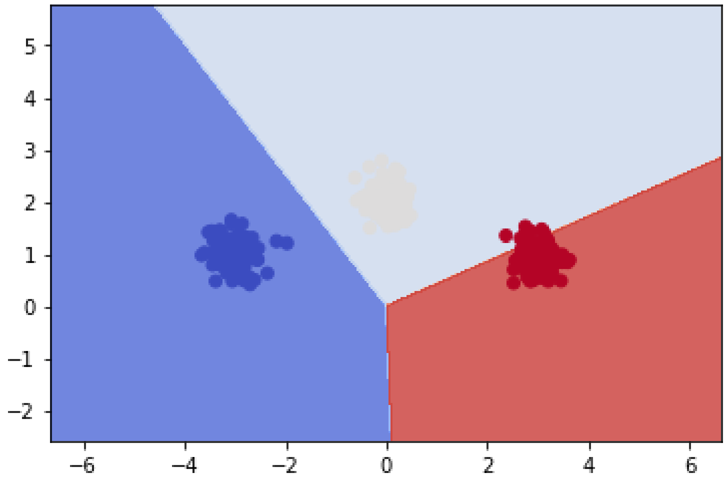
[[-1.05853334 -0.90294603]]

Coeffs 1

[[0.42121645 0.27171776]]

Coeffs 2

[[0.89164752 -0.82601734]]



13

```
def featureMap(X, y, num_classes):
    num_samples, num_inFeatures = (1, X.shape[0]) if len(
        X.shape) == 1 else (X.shape[0], X.shape[1])
    features = np.zeros((num_samples, num_classes * num_inFeatures))
    if num_samples == 1:
        features = np.zeros(num_classes * num_inFeatures)
        features[num_inFeatures*y:num_inFeatures*y+num_inFeatures] = X
        return features
    for i in range(num_samples):
        features[i, num_inFeatures*y:num_inFeatures*y+num_inFeatures] = X[i]
    return features
```

14

```
def sgd(X, y, num_outFeatures, subgd, eta=0.1, T=10000):
    num_samples = X.shape[0]
    w = np.zeros(num_outFeatures)
    for t in range(T):
        orderList = list(range(num_samples))
        random.shuffle(orderList)
        for i in orderList:
            j = subgd(X[i], y[i], w)
            w = w - eta*j
    return w
```

15

```
    def subgradient(self, x, y, w):
        score = np.zeros(self.num_classes)
        for _class in range(self.num_classes):
            diff = self.Psi(x, y) - self.Psi(x, _class)
            inner = np.sum(w*diff)
            score[_class] =  self.Delta(y, _class) + inner
        y_best = np.argmax(score)
        return 2*self.lam*w.T+self.Psi(x, y_best)-self.Psi(x, y)

    def fit(self, X, y, eta=0.1, T=10000):
        self.coef_ = sgd(X, y, self.num_outFeatures, self.subgradient, eta, T)
        self.fitted = True
        return self

    def decision_function(self, X):
        if not self.fitted:
            raise RuntimeError(
                "You must train classifer before predicting data.")

        score = np.zeros((X.shape[0], self.num_classes))
        for i in range(self.num_classes):
            score[:, i] = self.Psi(X, i) @ self.coef_
        return score

    def predict(self, X):
        score = self.decision_function(X)
        return np.argmax(score, axis=1)
```

16

w:
[-0.85646499 -0.47186337 0.05835316 0.50240647 0.79811183 -0.0305431]
array([[100,   0,   0],
       [0, 100,   0],
       [0,   0, 100]])