10-701 Introduction to Machine Learning

Homework 1

Due Oct 2, 11:59 am

Rules:

1. Homework submission is done via CMU Autolab system. Please compile your writeup and code in a single pdf file and submit to https://autolab.cs.cmu.edu/courses/10701-f15. Please let us know asap if you have trouble accessing Autolab.
2. Like conference websites, repeated submission is allowed. So please feel free to refine your answers. We will only grade the latest version.
3. Autolab may allow submission after the deadline, note however it is because of the late day policy. Please see course website for policy on late submission.
4. Please typeset your homework using appropriate software such as LaTeX. We will not accept scanned copies of handwritten papers.
5. You are allowed to collaborate on the homework, but you should write up your own solution and code. Please indicate your collaborators in your submission.

---

1 Probability and Statistics Review (25 Points) (Mrinmaya)

1.1 Exponential Families

Many commonly used distributions in Statistics and Machine Learning fall under the category of Exponential Family of distributions. Exponential family is a set of probability distributions whose probability density function (or probability mass function, if discrete) can be expressed in the form: $f_X(x) = h(x)\exp\left(\eta(\theta)\cdot T(x) - A(\theta)\right)$ where $T(x)$, $h(x)$, $\eta(\theta)$, and $A(\theta)$ are known.

(a) Show that Multinomial distribution, Multi-variate Gaussian distribution and Dirichlet distribution are members of the exponential family.

Answer:

(1) For multinomial distribution, we have

$$\begin{aligned} p(x|\theta) &= \frac{\Gamma(\sum_{i=1}^k x_i + 1)}{\prod_{i=1}^k \Gamma(x_i+1)} \prod_{i=1}^k \theta_i^{x_i} \\ &= \frac{\Gamma(\sum_{i=1}^k x_i + 1)}{\prod_{i=1}^k \Gamma(x_i+1)} \exp(\log \prod_{i=1}^k \theta_i^{x_i}) \\ &= \frac{\Gamma(\sum_{i=1}^k x_i + 1)}{\prod_{i=1}^k \Gamma(x_i+1)} \exp(\sum_{i=1}^k x_i \log \theta_i) \end{aligned} \tag{1}$$

Obviously, the multinomial distribution belongs to the exponential family with $h(x) = \frac{\Gamma(\sum_{i=1}^k x_i+1)}{\prod_{i=1}^k \Gamma(x_i+1)}, \eta(\theta) = [\log\theta_1, \log\theta_2, \ldots, \log\theta_k]$ and $T(x) = [x_1, x_2, \ldots, x_k]^T, A(\theta) = 0$.

(2) As for multivariate Gaussian distribution, we have

$$\begin{aligned} p(x|\mu,\Sigma) =& \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)) \\ &= \frac{1}{(2\pi)^{k/2}} \exp(-\frac{1}{2}tr(\Sigma^{-1}xx^T) + \mu^T\Sigma^{-1}x - \frac{1}{2}\mu^T\Sigma^{-1}\mu - \frac{1}{2}\log|\Sigma|) \end{aligned} \tag{2}$$

Accordingly, we get it's exponential family term as (here we use θ to denote the parameters μ, Σ):

$$\begin{aligned} \eta(\theta) &= [\Sigma^{-1}\mu; -\frac{1}{2}vec(\Sigma^{-1})] \\ T(x) &= [x; vec(xx^T)] \\ A(\theta) &= \frac{1}{2}\mu^T\Sigma^{-1}\mu + \frac{1}{2}\log|\Sigma| \\ h(x) &= (2\pi)^{-k/2} \end{aligned} \tag{3}$$

(3) For Dirichlet distribution, we have

$$\begin{aligned} p(x|\theta) &= \frac{\Gamma(\sum_{i=1}^k \theta_i)}{\prod_{i=1}^k \Gamma(\theta_i)} \prod_{i=1}^k x_i^{\theta_i - 1} \\ &= \exp(\log \frac{\Gamma(\sum_{i=1}^k \theta_i)}{\prod_{i=1}^k \Gamma(\theta_i)} \prod_{i=1}^k x_i^{\theta_i - 1}) \\ &= \exp(\log \frac{\Gamma(\sum_{i=1}^k \theta_i)}{\prod_{i=1}^k \Gamma(\theta_i)} + \sum_{i=1}^k (\theta_i - 1)\log x_i) \end{aligned} \tag{4}$$

Obviously, Dirichlet distribution belongs to the exponential family with $h(x) = 1, \eta(\theta) = [\theta_1 - 1, \theta_2 - 1, \ldots, \theta_k - 1], T(x) = [\log x_1, \log x_2, \ldots, \log x_k]^T$ and $A(\theta) = -\log \frac{\Gamma(\sum_{i=1}^k \theta_i)}{\prod_{i=1}^k \Gamma(\theta_i)}$

(b) Consider dataset $D = \{x_i\}_{i=1}^N$ which is independently and identically distributed (i.i.d.) according to some known exponential family distribution $f(x|\boldsymbol{\theta}) = h(x)\exp\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{T}(x) - A(\boldsymbol{\theta})\right)$. Let the prior for the parameter $\boldsymbol{\theta}$ be given by $p_\pi(\boldsymbol{\theta}|\boldsymbol{\chi},\nu) = f(\boldsymbol{\chi},\nu)\exp\left(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\chi} - \nu A(\boldsymbol{\theta})\right)$. Compute the posterior and show that it takes the same form as the prior.

Note: This notion is called "Conjugacy" and this will be very useful in Bayesian modelling.

Answer:

The posterior can be expressed as

$$\begin{aligned} p(\boldsymbol{\theta}|\boldsymbol{\chi},\nu,D) &\propto p(\boldsymbol{\theta}|\boldsymbol{\chi},\nu)p(D|\boldsymbol{\theta}) \\ &\propto p(\boldsymbol{\theta}|\boldsymbol{\chi},\nu)\prod_{i=1}^{N} p(x_i|\boldsymbol{\theta}) \\ &\propto f(\boldsymbol{\chi},\nu)\exp(\boldsymbol{\theta}^T\boldsymbol{\chi} - \nu A(\boldsymbol{\theta}))\prod_{i=1}^{N} h(x_i)\exp(\boldsymbol{\theta}^T\boldsymbol{T}(x_i) - A(\boldsymbol{\theta})) \\ &\propto f(\boldsymbol{\chi},\nu)\prod_{i=1}^{N} h(x_i)\exp(\boldsymbol{\theta}^T(\chi + \sum_{i=1}^{N}\boldsymbol{T}(x_i)) - (\nu+N)A(\boldsymbol{\theta})) \end{aligned} \tag{5}$$

which takes the same form as the prior.

1.2 Maximum Likelihood Estimation

We learnt about Maximum Likelihood estimation in class. For a fixed set of data and underlying statistical model, the method of maximum likelihood selects the set of values of the model parameters that maximises the likelihood function.

(a) Consider the model where the data $D = \{x_i\}_{i=1}^N$ is i.i.d. according to some known exponential family distribution. Further, let $\eta(\theta) = \theta$. Show that the maximum likelihood solution is given by: $A'(\hat{\theta}_{ML}) = \frac{1}{N}\sum_i T(x_i)$

Answer:

We can maximize the log likelihood function to find the MLE. The log likelihood function is written as:

$$\begin{aligned} \log p(D|\theta) &= \log\prod_{i=1}^{N} p(x_i|\theta) \\ &= \log\prod_{i=1}^{N} h(x_i)\exp(\theta T(x_i) - A(\theta)) \\ &= \sum_{i=1}^{N}\log h(x_i) + \theta\sum_{i=1}^{N} T(x_i) - NA(\theta) \end{aligned} \tag{6}$$

Taking derivative w.r.t θ and set it to zero:

$$\begin{aligned} \frac{\partial \log p(D|\theta)}{\partial\theta} &= \sum_{i=1}^{N} T(x_i) - NA'(\theta) = 0 \\ \Rightarrow A'(\theta) &= \frac{1}{N}\sum_{i=1}^{N} T(x_i) \end{aligned} \tag{7}$$

which indicates that the estimator is given by $A'(\hat{\theta}_{ML}) = \frac{1}{N}\sum_{i=1}^{N} T(x_i)$.

(b) Now, consider the special case where the data is i.i.d according to the uniform distribution: $x_i \sim$ *uniform*$(0,\theta)$ i.e. $p(x_i|\theta) = \begin{cases} \frac{1}{\theta} & x_i \in [0,\theta] \\ 0 & otherwise \end{cases}$

Now, compute the maximum likelihood estimator $\hat{\theta}_{ML}$.

Answer:

When the data point follows the uniform distribution, we have the the likelihood as follows:

$$\begin{aligned} p(D|\theta) &= \prod_{i=1}^{N} p(x_i|\theta) \\ &= \frac{1}{\theta^N} I(\theta \geq \max(\{x_i\}_{i=1}^N)) \end{aligned} \tag{8}$$

where $I(*)$ is the indicator function. Observe that the likelihood function reaches maxima when $\theta = \max(\{x_i\}_{i=1}^N)$, thus the MLE is $\hat{\theta}_{ML} = \max(\{x_i\}_{i=1}^N)$.

2 Decision Boundary of Naive Bayes (10 Points) (Yuntian)

Consider the problem of predicting a label $Y \in \mathcal{Y}$ given an input feature vector $X \in \mathcal{X}$. Suppose $\mathcal{X} = \mathbb{R}^d$, $Y \in \{0, 1\}$. We make the Naive Bayes assumption that features are conditionally independent given label, i.e. $P(X|Y) = \prod_{i=1}^d P(X_i|Y)$, where X_i denotes the i-th component of X. In addition, we impose a Bernoulli prior on Y: $P(Y = 1) = \pi$.

(a) Assume that $P(X_i|Y)$ is in the exponential family: $P(X_i = x_i|Y = y) = h_i(x_i) \exp(\theta_{iy} \cdot T_i(x_i) - A_i(\theta_{iy}))$. Compute the posterior distribution $P(Y|X)$. Compute the decision boundary $\{x \in \mathcal{X} : P(Y = 1|X = x) = P(Y = 0|X = x)\}$. (You can use sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ to simplify the expression).

Answer

$$\begin{aligned} P(Y = y|X = x) &\propto P(X = x|Y = y)P(Y = y) \\ &= \textstyle\prod_{i=1}^d P(X_i = x_i|Y = y)P(Y = y) \\ &= \textstyle\prod_{i=1}^d h_i(x_i) \exp(\theta_{iy} \cdot T_i(x_i) - A_i(\theta_{iy}))\pi^y(1-\pi)^{1-y} \\ &= \left(\textstyle\prod_{i=1}^d h_i(x_i)\right) \exp\left(\textstyle\sum_{i=1}^d (\theta_{iy} \cdot T_i(x_i) - A_i(\theta_{iy}))\right) \pi^y(1-\pi)^{1-y} \end{aligned} \tag{9}$$

Hence we have

$$\begin{aligned} & P(Y = 1|X = x) \\ = & \frac{\pi \exp\left(\sum_{i=1}^d (\theta_{i1} \cdot T_i(x_i) - A_i(\theta_{i1}))\right)}{\pi \exp\left(\sum_{i=1}^d (\theta_{i1} \cdot T_i(x_i) - A_i(\theta_{i1}))\right) + (1-\pi) \exp\left(\sum_{i=1}^d (\theta_{i0} \cdot T_i(x_i) - A_i(\theta_{i0}))\right)} \\ = & \sigma(\textstyle\sum_{i=1}^d (\theta_{i1} - \theta_{i0}) \cdot T_i(x_i) - \sum_{i=1}^d (A_i(\theta_{i1}) - A_i(\theta_{i0})) + \ln \frac{\pi}{1-\pi}) \end{aligned} \tag{10}$$

Similary,

$$P(Y = 0|X = x) = \sigma(\sum_{i=1}^{d} (\theta_{i0} - \theta_{i1}) \cdot T_i(x_i) - \sum_{i=1}^{d} (A_i(\theta_{i0}) - A_i(\theta_{i1})) + \ln \frac{1-\pi}{\pi}) \tag{11}$$

For x on the decision boundary, we shall have

$$P(y = 1|X = x) = P(y = 0|X = x) \tag{12}$$

which implies that

$$P(y = 1|X = x) = 0.5 \tag{13}$$

Substitute into Equation 10, we have

$$\sum_{i=1}^{d} (\theta_{i1} - \theta_{i0}) \cdot T_i(x_i) - \sum_{i=1}^{d} (A_i(\theta_{i1}) - A_i(\theta_{i0})) + \ln \frac{\pi}{1-\pi} = 0 \tag{14}$$

(b) Assume that $P(X_i|Y=y)$ is a Gaussian distribution with mean μ_{iy} and variance σ_i^2 (Note that the variance here does not depend on label Y), show that the decision boundary is linear in terms of X. (Hint: Recall from Problem 1.1 that Gaussian distribution is in the exponential family).

Answer As $P(X_i|Y=y)$ follows a Gaussian distribution parameterized by $\{\mu_{iy}, \sigma_i^2\}$, we first rewrite it in the exponential family form[1]:

$$\begin{aligned} P(X_i = x_i|Y=y) &= \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp(\frac{-(x_i-\mu_{iy})^2}{2\sigma_i^2}) \\ &= \frac{1}{\sqrt{2\pi}} \exp(-\ln\sigma_i - \frac{x_i^2}{2\sigma_i^2} + \frac{\mu_{iy}x_i}{\sigma_i^2} - \frac{\mu_{iy}^2}{2\sigma_i^2}) \\ &= h_i(x_i)\exp(\eta(\theta_{iy})\cdot T_i(x_i) - A(\theta_{iy})) \end{aligned} \tag{15}$$

with:

$$\begin{aligned} \theta_{iy} &= [\mu_{iy}, \sigma_i^2]^T \\ h_i(x_i) &= \frac{1}{\sqrt{2\pi}} \\ \eta(\theta_{iy}) &= [\frac{\mu_{iy}}{\sigma_i^2}, -\frac{1}{2\sigma_i^2}]^T \\ T_i(x_i) &= [x_i, x_i^2]^T \\ A_i(\theta_{iy}) &= \frac{\mu_{iy}^2}{2\sigma_i^2} + \ln\sigma_i \end{aligned} \tag{16}$$

Replace θ_{iy} with $\eta(\theta_{iy})$, and substitute the other terms into the general boundary expression for exponential family distributions in Eq.14, the square term x^2 is cancelled and we have

$$\begin{aligned} &\sum_{i=1}^{d}(\eta(\theta_{i1}) - \eta(\theta_{i0}))\cdot T_i(x_i) - \sum_{i=1}^{d}(A_i(\theta_{i1}) - A_i(\theta_{i0})) + \ln\frac{\pi}{1-\pi} = 0 \\ \Rightarrow &\sum_{i=1}^{d}\frac{\mu_{i1}-\mu_{i0}}{\sigma_i^2}x_i - \sum_{i=1}^{d}\frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2} + \ln\frac{\pi}{1-\pi} = 0 \end{aligned} \tag{17}$$

Obviously, the decision boundary is linear with X.

3 KNN Classification (15 Points) (Yuntian)

Consider a classification problem using kNN. We have N training points $x_1, x_2, \cdots x_N$ and corresponding labels $y_1, y_2, \cdots y_N$. If we wish to classify a new data point, the classification rule is simply a majority vote among its k nearest neighbours in the training set.

(a) Consider $k=1$ case, is it possible to build a decision tree (the decision at each node can only take the form of "$x^i \leq t$ or $x^i > t$" where x^i denotes the i-th feature of x, t is a real number and can be different in different nodes) which behaves exactly the same as the 1-NN classifier?

Answer Let d denote the number of features.

- If $d=1$, we can compute the distance between two points by using only one feature, so we can build a decision tree with the same decision boundary as 1-NN.

[1] Adapted from Hao Zhang's homework

- If $d > 1$, the decision boundary of 1-NN lies on the voronoi diagram edges of the training points. An 2-D illustration can be seen from Figure 1[2]. On the other hand, the decision boundary of a decision tree consists of piecewise hyper-planes (lines for $d = 2$) parallel to feature axes. An 2-D illustration is shown in Figure 2. Hence decision tree cannot behave exactly the same as 1-NN in general.

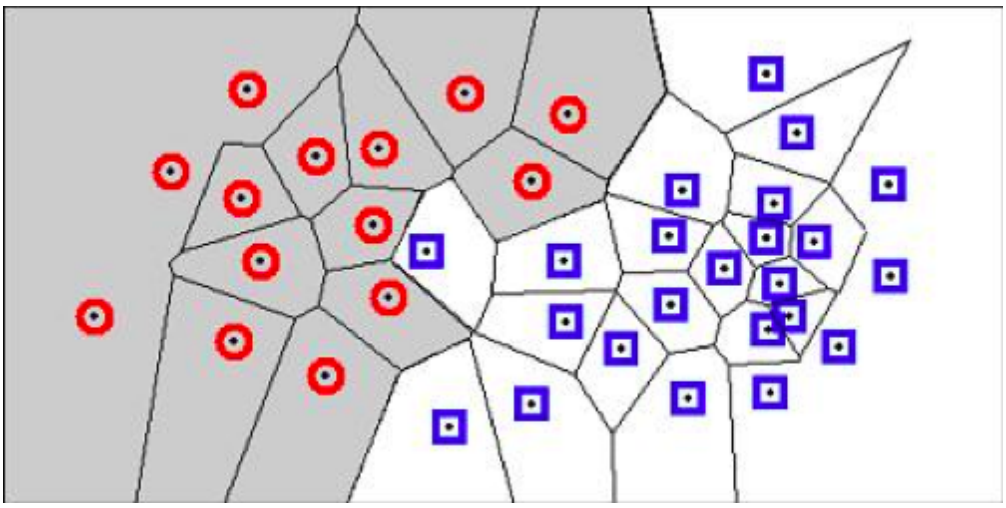

Figure 1: Decision boundary of 1-NN

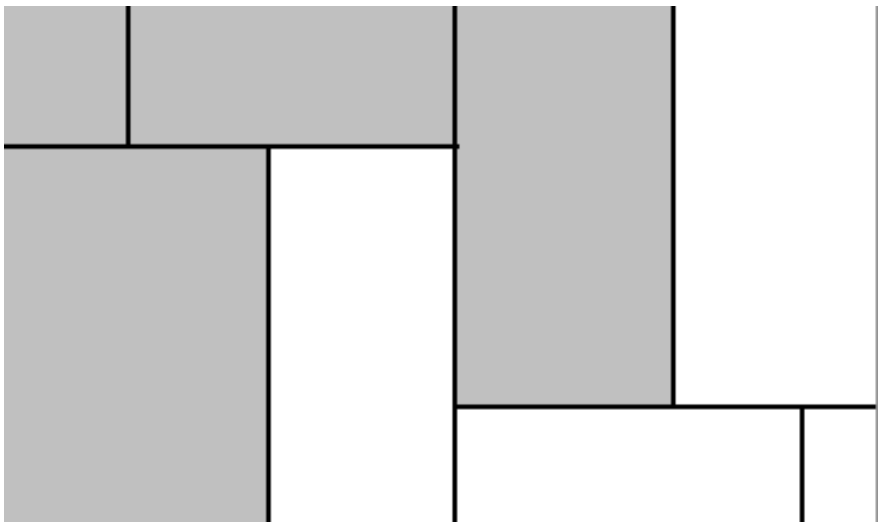

Figure 2: Decision boundary of decision tree

(b) Recall from class that the decision rule of kNN can be viewed using a probabilistic interpretation: Suppose we have a data set comprising of N_k points in class C_k. For a new point x, we draw a ball around x containing exactly K nearest neighbours. Suppose this ball has volume V and contains K_k points from class C_k. Then we can estimate the density conditioned on class C_k as:

$$P(x|C_k) = \frac{K_k}{N_k V}$$

[2] http://users.eecs.northwestern.edu/~yingwu/teaching/EECS510/Notes/NearestNeighbor_1_handout.pdf

Similarly, we estimate the unconditioned density as:

$$P(x) = \frac{K}{NV}$$

We also estimate the prior as:

$$P(C_k) = \frac{N_k}{N}$$

In class we have shown that under this model, Bayes decision rule will yield kNN classification. Show that the density $P(x)$ given by this model is an improper distribution whose integral over all space is not guaranteed to sum up to 1. (You only need to construct a special case to show that).

Answer Consider one-dimension data classification by 1-NN: Assume that we have only one training data point located at $x = 0$, then $p(x) = \frac{1}{|x|}$, $\int_{-\infty}^{\infty} p(x)dx = \infty$.

(c) In kNN classification, the nearest neighbours of a given test point depend on the distance metric. We often implicitly use Euclidean distance as the distance metric, but we can use other distance metrics as well. Consider a binary classification problem using 1-NN. Assume the labels can only be C_1 and C_2. Denote the number of training points as N. For a test point x, denote its nearest neighbour as x' (note that the nearest neighbour depends on the distance metric), then the probability that x is misclassified is:

$$P_N(e|x) = P(C_1|x)P(C_2|x') + P(C_2|x)P(C_1|x')$$

The asymptotic error rate (i.e. the error rate when the size of training set is infinite) is:

$$P(e|x) = \lim_{N\to\infty} P_N(e|x)$$

It can be shown that when $N \to \infty$, the asymptotic error rate $P(e|x)$ of 1-NN is upper bounded by twice the minimum achievable error rate. However, what we care about is $P_N(e|x)$. Therefore, if we can bound the difference between $P_N(e|x)$ and $P(e|x)$, then $P_N(e|x)$ is also upper bounded (As $P_N(e|x)$ depends on the training set, we can only bound it in a probabilistic sense, but that's not the focus of this problem). Our objective here is to use a proper distance metric to minimize the expected squared difference between $P(e|x)$ and $P_N(e|x)$: $\min \mathbb{E}[(P_N(e|x) - P(e|x))^2|x]$. Note that the expectation here is taken with respect to the training set, conditioned on the test point x.

Denote $\nabla P(C_1|x) \equiv \frac{\partial P(C_1|x)}{\partial x}$. Approximate $P(C_1|x')$ by the first order Taylor expansion at x: $P(C_1|x') \simeq P(C_1|x) + \nabla P(C_1|x)^T(x' - x)$, show that $\mathbb{E}[(P_N(e|x) - P(e|x))^2|x]$ is minimized by using the distance metric $d(x, x') \equiv |\nabla P(C_1|x)^T(x - x')|$.

Note: In practice, we can estimate the direction of $\nabla P(C_1|x)$ from training data. Denote it as $\hat{\nabla}$, then we can use $d(x, x') \equiv |\hat{\nabla}^T(x - x')|$ as the distance metric, because the scaling of $\nabla P(C_1|x)$ does not affect the nearest neighbour.

Answer When $N \to \infty$, $x' \to x$, hence

$$\begin{aligned} & P(e|x) \\ = \; & \lim_{N\to\infty} P_N(e|x) \\ = \; & \lim_{N\to\infty} P(C_1|x)P(C_2|x') + P(C_2|x)P(C_1|x') \\ = \; & P(C_1|x)P(C_2|x) + P(C_2|x)P(C_1|x) \\ = \; & 2P(C_1|x)P(C_2|x) \end{aligned} \tag{18}$$

Therefore

$$\begin{aligned} & \mathbb{E}[(P_N(e|x) - P(e|x))^2|x] \\ = \; & \mathbb{E}[(P(C_1|x)P(C_2|x') + P(C_2|x)P(C_1|x') - 2P(C_1|x)P(C_2|x))^2 \,|x] \\ = \; & \mathbb{E}[((P(C_1|x) - P(C_2|x))(P(C_1|x) - P(C_1|x')))^2 \,|x] \\ = \; & \mathbb{E}[\big((P(C_1|x) - P(C_2|x))(\nabla P(C_1|x)^T(x - x'))\big)^2 \,|x] \\ = \; & \mathbb{E}[(P(C_1|x) - P(C_2|x))^2 d^2(x, x')|x] \end{aligned} \tag{19}$$

Now we are ready to prove that by using distance metric $d(x, x')$, the expectation is minimized: for any distance metric $d'(x, x')$, we prove that the expectation in Equation 19 is greater than that achieved by using $d(x, x')$:

For any training set $x_1, \cdots, x_N$ and the test point x, the nearest neighbor x' found by the distance metric $d(x, x')$ is:

$$x' = \underset{x_i \in \{x_1, \cdots, x_N\}}{\operatorname{argmin}} d^2(x, x_i) \tag{20}$$

while the nearest neighbor x'' found by distance metric $d'(x, x')$ is:

$$x'' = \underset{x_i \in \{x_1, \cdots, x_N\}}{\operatorname{argmin}} d'^2(x, x_i) \tag{21}$$

It's straightforward to see that $d^2(x, x') \leq d^2(x, x'')$, hence

$$(P(C_1|x) - P(C_2|x))^2 d^2(x, x') \leq (P(C_1|x) - P(C_2|x))^2 d^2(x, x'') \tag{22}$$

As the inequality holds for any training set, it also holds after we take the expectaion with respect to the training set:

$$\mathbb{E}[(P(C_1|x) - P(C_2|x))^2 d^2(x, x')|x] \leq \mathbb{E}[(P(C_1|x) - P(C_2|x))^2 d^2(x, x'')|x] \tag{23}$$

Therefore $\mathbb{E}[(P_N(e|x) - P(e|x))^2|x]$ is minimized by using the distance metric $d(x, x')$. The proof completes.

4 Decision Trees (25 Points) (Yuntian)

4.1 Decision tree on two features

Consider training a decision tree on n two dimensional vectors $x = (x_1, x_2)$.

(a) Assume we have two equal vectors x and x' in our training set (that is, all attributes of x and x' including the labels are exactly the same). Can removing x' from our training data change the decision tree we learn for this dataset? Explain briefly.

Answer Yes, the decision tree can change. The conditional entropy in each split depends on the set of samples and copying a vector twice may change the distribution leading to a selection of a different attribute to split on.

(b) Assume that the training instances are linearly separable. That is, there exists a $\{w, b\}$ such that

$$y = \begin{cases} +1 & w^T x + b > 0 \\ -1 & w^T x + b \leq 0 \end{cases}$$

Can a decision tree correctly classify these vectors? (the decision at each node can only take the form of "$x_i \leq t$ or $x_i > t$" where x_i is a feature, t is a real number and can be different in different nodes). If so, what is an upper bound on the depth of the corresponding decision tree (as tight as possible)? If not, why not?

Answer Yes. One possible strategy is to split the points according to their x_1 values. Since the data is linearly separable, for each x_1 value there is a cutoff on x_2 so that values above it are in class 1 and below it in class -1. Splitting the data based on the x1 values can be done with a $\log(n)$ depth tree and then we only need at most one more node to correctly classify all points so the total depth is $O(\log n)$.

(c) Now assume that these n inputs are not linearly separable (that is, no $\{w, b\}$ exists for correctly classifying all inputs using the above rule). Can a decision tree correctly classify these vectors? (the decision at each node can only take the form of "$x_i \leq t$ or $x_i > t$" where x_i is a feature, t is a real number and can be different in different nodes) If so, what is an upper bound on the depth of the corresponding decision tree (as tight as possible)? If not, why not?

Answer Yes. Similar to what we did for (b) we can split the points according to their x_1 values. However, since they are not linearly separable we cannot assume a cutoff on x_2 anymore. Instead, we may need to consider different values for x_2 again, the can be done in at most $\log(n)$ depth for a total of $2\log(n)$ and an $O(\log n)$ depth.

4.2 C4.5 Algorithm

Day	Outlook	Temperature	Humidity	Wind	Play
D1	Sun	26	High	Low	No
D2	Sun	25	High	High	No
D3	Overcast	25	High	Low	Yes
D4	Rain	24	High	Low	Yes
D5	Rain	19	Normal	Low	Yes
D6	Rain	20	Normal	High	No
D7	Overcast	20	Normal	High	Yes
D8	Sun	23	High	Low	No
D9	Sun	20	Normal	Low	Yes
D10	Rain	25	Normal	Low	Yes
D11	Sun	24	Normal	High	Yes
D12	Overcast	22	High	High	Yes
D13	Overcast	23	Normal	Low	Yes
D14	Rain	23	High	High	No

Table 1: Dataset

We have learnt ID3 algorithm in class. One limitation of ID3 is that it is overly sensitive to features with large numbers of values. For example, if each training instance has a unique id, then the information gain will be maximized by using this id as feature, while this is not useful. C4.5 algorithm improves this by using information gain ratio instead of information gain to evaluate features. Denote the feature for an input by X, and its label by Y. Recall that in ID3, we choose feature X that maximizes information gain $IG(X)$:

$$IG(X) \equiv H(Y) - H(Y|X)$$

Now we define split information as follows: Suppose there are$|D|$ samples at the current node, and after splitting by feature X, they fall into V child nodes. Assume the number of samples that pass through each child node is $|D_1|, |D_2|, \cdots |D_V|$, then the split information of feature X is

$$SplitInfo(X) \equiv -\sum_{j=1}^{V} \frac{|D_j|}{|D|} \log \frac{|D_j|}{|D|}$$

The information gain ratio is defined as

$$GainRatio(X) \equiv \frac{IG(X)}{SplitInfo(X)}$$

C4.5 algorithm uses information gain ratio to determine the best splitting attribute. The intuition is that $SplitInfo(X)$ acts as a normalizer to penalize features with a large number of values. For example, if $\forall i, j$ we have $|D_i| = |D_j|$, then $SplitInfo(X) = \log V$, so features with smaller V is preferred.

Draw the decision tree of the dataset in table 1[3] using both the ID3 and C4.5 algorithms. The features here are Outlook, Temperature, Humidity and Wind, and the label to be predicted is Play. Treat Temperature as a discrete feature.

[3] http://saiconference.com/Downloads/SpecialIssueNo10/Paper_3-A_comparative_study_of_decision_tree_ID3_and_C4.5.pdf

Answer The tree produced by ID3 algorithm is in Figure 3[4], and the tree produced by C4.5 algorithm is in Figure 4[5].

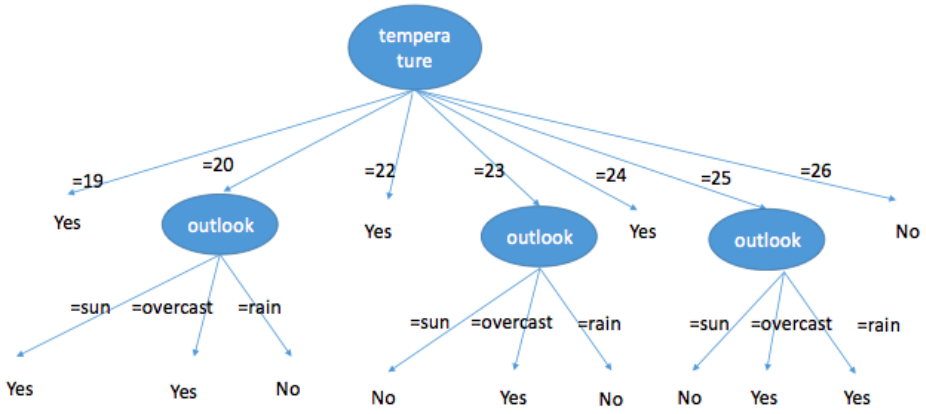

Figure 3: Decision Tree of ID3

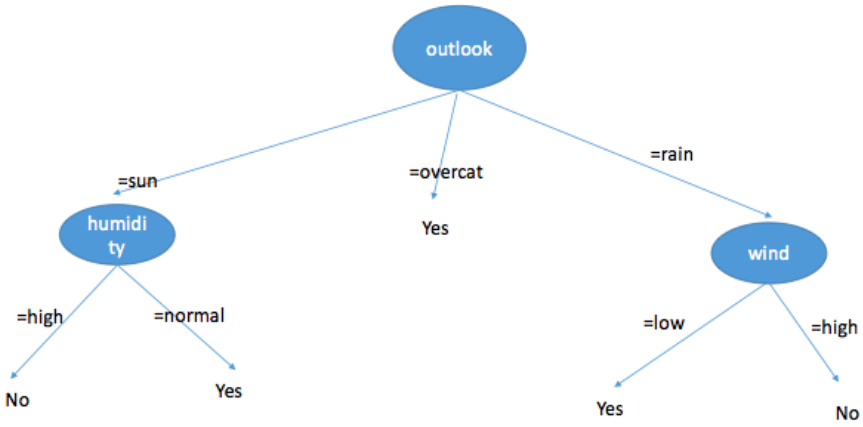

Figure 4: Decision Tree of C45

The trees are produced by the following code:

```
import sys, math, operator

def load_dataset(input_filename):
    with open(input_filename) as fin:
        line_idx = 0
```

[4]provided by Hao Zhang
[5]provided by Hao Zhang

```
        features = []
        for line in fin:
            line_idx += 1
            line_split = line.split()
            line_converted = []
            for item in line_split:
                item_strip = item.strip()
                if item_strip != '&' and len(item_strip) > 0:
                    line_converted.append(item_strip)
            features.append(line_converted)
        return features

# Generate id3 tree from data set and append to node
def GEN_ID3(node, dataset):
    if len(dataset) > 1:
        feature_1 = dataset[1]
        num_features = len(feature_1) - 2

        label_dict = {}
        for i in range(len(dataset)-1):
            instance = dataset[i+1]
            label = instance[len(instance)-1]
            if not label in label_dict:
                label_dict[label] = 0
            label_dict[label] = label_dict[label] + 1

        labels = label_dict.keys()
        if len(labels) == 1:
            node['decision'] = labels[0]
            return
        entropy = 0
        for label in labels:
            percent = float(label_dict[label]) / (len(dataset)-1)
            entropy += -1.0 * percent * math.log(percent)

        information_gain = {}
        cache_dict = {}
        for feature_id in range(num_features):
            feature_name = dataset[0][feature_id+1]
            label_dict = {}
            for i in range(len(dataset)-1):
                instance = dataset[i+1]
                feature_val = instance[feature_id+1]
                label = instance[len(instance)-1]
                if not feature_val in label_dict:
                    label_dict[feature_val] = {}
                if not label in label_dict[feature_val]:
                    label_dict[feature_val][label] = 0
                if not 'total_cnt' in label_dict[feature_val]:
                    label_dict[feature_val]['total_cnt'] = 0
                label_dict[feature_val]['total_cnt'] += 1
                label_dict[feature_val][label] = \
                        label_dict[feature_val][label] + 1
            cache_dict[feature_name] = label_dict
            feature_vals = label_dict.keys()
            entropy_total = 0
            for feature_val in feature_vals:
                entropy_feature = 0
                labels = label_dict[feature_val].keys()
                total_cnt = label_dict[feature_val]['total_cnt']
                for label in labels:
                    cnt = label_dict[feature_val][label]
                    percent = float(cnt) / total_cnt
                    entropy_feature += -1.0*percent*math.log(percent)
                entropy_total += float(total_cnt) / \
```

```
                        (len(dataset)-1) * entropy_feature
            information_gain[feature_id] = entropy - entropy_total
        feature_sel_id = max(information_gain.iteritems(), \
                key=operator.itemgetter(1))[0]
        feature_sel = dataset[0][feature_sel_id+1]
        node[feature_sel] = {}
        label_dict = cache_dict[feature_sel]
        feature_vals = label_dict.keys()
        for feature_val in feature_vals:
            dataset_feature = []
            dataset_feature.append(dataset[0])
            for i in range(len(dataset)-1):
                instance = dataset[i+1]
                if feature_val == instance[feature_sel_id+1]:
                    dataset_feature.append(instance)
            node[feature_sel][feature_val] = {}
            GEN_ID3(node[feature_sel][feature_val], dataset_feature)

# Generate C4.5 tree from dataset and append to node
def GEN_C45(node, dataset):
    if len(dataset) > 1:
        feature_1 = dataset[1]
        num_features = len(feature_1) - 2

        label_dict = {}
        for i in range(len(dataset)-1):
            instance = dataset[i+1]
            label = instance[len(instance)-1]
            if not label in label_dict:
                label_dict[label] = 0
            label_dict[label] = label_dict[label] + 1

        labels = label_dict.keys()
        if len(labels) == 1:
            node['decision'] = labels[0]
            return
        entropy = 0
        for label in labels:
            percent = float(label_dict[label]) / (len(dataset)-1)
            entropy += -1.0 * percent * math.log(percent)

        information_gain = {}
        cache_dict = {}
        for feature_id in range(num_features):
            feature_name = dataset[0][feature_id+1]
            label_dict = {}
            for i in range(len(dataset)-1):
                instance = dataset[i+1]
                feature_val = instance[feature_id+1]
                label = instance[len(instance)-1]
                if not feature_val in label_dict:
                    label_dict[feature_val] = {}
                if not label in label_dict[feature_val]:
                    label_dict[feature_val][label] = 0
                if not 'total_cnt' in label_dict[feature_val]:
                    label_dict[feature_val]['total_cnt'] = 0
                label_dict[feature_val]['total_cnt'] += 1
                label_dict[feature_val][label] = \
                        label_dict[feature_val][label] + 1
            cache_dict[feature_name] = label_dict
            feature_vals = label_dict.keys()
            entropy_total = 0
            split_info = 0
            for feature_val in feature_vals:
                entropy_feature = 0
```

```
                labels = label_dict[feature_val].keys()
                total_cnt = label_dict[feature_val]['total_cnt']
                for label in labels:
                    cnt = label_dict[feature_val][label]
                    percent = float(cnt) / total_cnt
                    entropy_feature += -1.0*percent*math.log(percent)
                entropy_total += float(total_cnt) / \
                        (len(dataset)-1) * entropy_feature
                split_percent = float(total_cnt) / (len(dataset)-1)
                split_info += -1.0*split_percent*math.log(split_percent)
            if split_info != 0:
                information_gain[feature_id] = \
                        (entropy - entropy_total)/split_info
        feature_sel_id = max(information_gain.iteritems(), \
                key=operator.itemgetter(1))[0]
        feature_sel = dataset[0][feature_sel_id+1]
        node[feature_sel] = {}
        label_dict = cache_dict[feature_sel]
        feature_vals = label_dict.keys()
        for feature_val in feature_vals:
            dataset_feature = []
            dataset_feature.append(dataset[0])
            for i in range(len(dataset)-1):
                instance = dataset[i+1]
                if feature_val == instance[feature_sel_id+1]:
                    dataset_feature.append(instance)
            node[feature_sel][feature_val] = {}
            GEN_C45(node[feature_sel][feature_val], dataset_feature)

# Generate id3 recursively
def generate_id3(dataset):
    tree = {}
    GEN_ID3(tree, dataset)
    return tree

# Generate C4.5 recursively
def generate_c45(dataset):
    tree = {}
    GEN_C45(tree, dataset)
    return tree


if __name__ == '__main__':
    if len(sys.argv) != 3:
        print >> sys.stderr, 'Usage: python %s '\
                '<input-filename> <method>' %sys.argv[0]
        print >> sys.stderr, '<method> can be either ID3 or C45'
        sys.exit(1)

    input_filename = sys.argv[1]
    method = sys.argv[2]


    dataset = load_dataset(input_filename)
    if method == 'ID3':
        tree = generate_id3(dataset)
    elif method == 'C45':
        tree = generate_c45(dataset)
    else:
        print >> sys.stderr, 'Unknown argument %s' %method
        sys.exit(1)
    print tree
```

The dataset used by the above code is:

```
Day & Outlook & Temperature & Humidity & Wind & Play &
D1 & Sun & 26 & High & Low & No &
D2 & Sun & 25 & High & High & No &
D3 & Overcast & 25 & High & Low & Yes &
D4 & Rain & 24 & High & Low & Yes &
D5 & Rain & 19 & Normal & Low & Yes &
D6 & Rain & 20 & Normal & High & No &
D7 & Overcast & 20 & Normal & High & Yes &
D8 & Sun & 23 & High & Low & No &
D9 & Sun & 20 & Normal & Low & Yes &
D10 & Rain & 25 & Normal & Low & Yes &
D11 & Sun & 24 & Normal & High & Yes &
D12 & Overcast & 22 & High & High & Yes &
D13 & Overcast & 23 & Normal & Low & Yes &
D14 & Rain & 23 & High & High & No &
```

The output will be the generated tree (in the form of a dictionary) corresponding to the input arguments.

5 Naive Bayes for sentiment classification (25 Points) (Mrinmaya)

The goal of this assignment is to implement a Naive Bayes classifier as described in the lecture and to apply it to the task of sentiment classification. You are free to design the code as you wish and to implement the code in any language that you wish.

We will work with the following movie review data set: http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz

The data set contains movie reviews classified as positive or negative reviews. We will provide standard train and test splits in the data - use reviews 000 to 799 for training and reviews 800-999 (in both pos and neg classes) for testing. Ignore the cross-validation tags. Please use this exact split only. There are two steps to this assignment: the pre-processing step and the classification step.

5.1 Pre-processing step

This first step converts the movie reviews into features to be used by the naive Bayes classifier. You will be using the bag of words approach described in class. For completeness, the following steps outline the process involved:

1. Form the vocabulary: Use all the positive and negative sentiment carrying words provided by Hu and Liu: http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar[6] as the vocabulary. The words in the vocabulary will form the feature set. You may want to keep the vocabulary in alphabetical order to help you with debugging your assignment, but this is not necessary.

2. Now, convert the training data into a set of features. Let M be the size of your vocabulary. For each review, you will convert it into a feature vector of size M. Each slot in that feature vector takes the value of 0 or 1. If the i^{th} slot is 1, it means that the i^{th} word in the vocabulary is present in the review; otherwise, if it is 0, then the i^{th} word is not present in the review. Most of the feature vector slots will be 0 so it will be efficient if you only store the indices that have 1's.

5.2 Classification step

Build a naive Bayes classifier as described in class.

[6]You will need to remove the header information lines from the files positive-words.txt and negative-words.txt

1. In the first phase, which is the training phase, the naive Bayes classifier reads in the training data along with the training labels and learns the parameters used by the classifier.

2. In the testing phase, the trained naive Bayes classifier classifies the data in the testing data file. You will need to convert the reviews in the testing data into a feature vector, just like in the training data where a 1 in the i^{th} slot indicates the presence of the i^{th} word in the vocabulary while a 0 indicates the absence. If you encounter a word in the testing data that is not present in your vocabulary, ignore that word.

3. Output the accuracy of the naive Bayes classifier by comparing the predicted class label of each review in the testing data to the actual class label. The accuracy is the number of correct predictions divided by the total number of predictions.

Note: Make sure that you follow the implementation hints given in the lecture. Specifically, you may want to do the probability calculations in log space to prevent numerical instability. Also, use laplace prior of 0.1 (also called "add 0.1 smoothing") to avoid zero probabilities.

5.3 Results

Your results must be reported in a write-up (to be attached with the written component of this assignment).

(a) Train Accuracy: Run your classifier by training on the training split and then testing on training split itself. Report the accuracy. In this situation, you are training and testing on the same data. This is a sanity check: your accuracy should be fairly high.

(b) Test Accuracy: Run your classifier by training on the training split and then testing on testing split. Report the accuracy. Are you over-fitting?

(c) Most sentiment carrying words: For both positive and negative classes, print top 10 words i.e. words that have highest $p(word|sentiment = positive)$ and $p(word|sentiment = negative)$.

5.4 Experiment with your code

Now, we will play around with the code a little bit and try to boost up the classifier accuracy if we can.

Negation Handling: A major problem faced during the task of sentiment classification is that of handling negations. Since we are using each word as feature, the word "good" in the phrase "not good" will be contributing to positive sentiment rather that negative sentiment as the presence of "not" before it is not taken into account. To mitigate this, we will introduce a simple fix - for all words preceded by a not or n't in the training set, introduce a new feature called "not_" + word. Add these features to the ones included in the vocabulary you used above.

Perform negation handling and report Train and Test Accuracy on the data set as before. Are you over-fitting? Does your accuracy increase? Why or Why not? Explain in your report.

Including Bi-grams: Often, information about sentiment is conveyed by adjectives or more specifically by certain combinations of adjectives with other parts of speech. This information can be captured by adding features like consecutive pairs of words (also called bi-grams) where the first word is an adverb of degree and the second word is a sentiment carrying word. Words like "very" or "definitely" don't provide much sentiment information on their own, but phrases like "very bad" or "definitely recommended" increase the probability of a document being negatively or positively biased. To utilize this we will add some more bi-grams to our vocabulary. Specifically, we will add all pairs where the first word is one of the following seven adverbs: "extremely", "quite", "just", "almost", "very", "too", "enough", and the second word is any word in the original vocabulary you used above.

Perform bi-gram inclusion and report Train and Test Accuracy on the data set as before. Are you over-fitting? Does your accuracy increase? Why or Why not? Explain in your report.

5.5 Extra Credit (5 Points)

Now that you have your cool Sentiment classifier ready, try to improve it a little more and earn a bonus of 5 more points.

One idea to do this is to allow a vocabulary of all words in your training set (do not use the Hu and Liu vocabulary), perform negation handling and include all bi-grams (all consecutive words in the training set) to your vocabulary. The use of higher dimensional features like all possible bi-grams will result in a blow up in the number of features. So it might be useful to down select our features. While we have not covered feature selection in class, you are encouraged to read up on these and implement one. Our suggestion is to do "Forward Greedy Selection" but you are free to implement one of your choice.

Describe your feature selector. How many features did you finally select? Why? Report your Train and Test Accuracy on the data set as before. Are you over-fitting now? Does your accuracy increase? Why or Why not? Explain in your report.

5.6 Submission Requirements

You are required to submit your code as well as your report. Attach your report with the written component of the assignment. Submission instructions will be announced soon.

The code is attached in the end of the write-up.

5.3

The code for this part is attached in section 6.1, and the utility functions are attached in section 6.2.

(a)

The train accuracy I achieved is 0.949375.

(b)

The test accuracy I achieved is 0.8175. As the training accuracy is substantially higher than the test accuracy, my answer is yes, I am overfitting.

(c)

The top 10 sentiment carrying words for positive class is as follows:

```
like
good
well
best
great
work
plot
love
enough
right
```

The top 10 sentiment carrying words for negative class is as follows:

```
like
good
bad
plot
```

well
better
best
work
enough
love

5.4

Negation Handling

The code for this part is attached in section 6.3, and the utility functions are attached in section 6.2. The train accuracy: 0.958125, the test accuracy: 0.83. Yes, I am still overfitting. The accuracy slightly increase, because I add more features.

Including Bi-grams

The code for this part is attached in section 6.4, and the utility functions are attached in section 6.2. The training accuracy is 0.967500, the test accuracy is 0.830000. Yes, I am still overfitting. The train accuracy slightly increases, but the test accuracy not. The main reason is that my feature dimension is too large, and I am suffering very serious overfitting. Maybe some feature selection techniques will help.

6 Code for Problem 5

6.1 Original Code: Main Script

```
% 10701 Homework 1, Problem 5
%% Build the vocabulary
vocab_path = 'opinion-lexicon-English';
positive_words = {};
negative_words = {};

% read the positive words
path = fullfile(vocab_path, 'positive-words.txt');
f = fopen(path, 'r');
for i = 1:36
    word = fgetl(f);
end
while word ~= -1
   positive_words{end+1} = word;
   word = fgetl(f);
end
fclose(f);
% read the negative words
path = fullfile(vocab_path, 'negative-words.txt');
f = fopen(path, 'r');
for i = 1:36
    word = fgetl(f);
end
while -1 ~= word
   negative_words{end+1} = word;
    word = fgetl(f);
end
fclose(f);

vocabulary = [positive_words, negative_words]';
vocab_label = [ones(numel(positive_words), 1); zeros(numel(negative_words), 1)];
```

```
[vocabulary, idx] = sort(vocabulary);
vocab_label = vocab_label(idx);
dim = numel(vocabulary);

%% Extract features
train_ids = 0:799; num_train = length(train_ids) * 2;
test_ids = 800:999; num_test = length(test_ids) * 2;

train_features = sparse(zeros(dim, num_train));
train_labels = zeros(num_train, 1);
test_features = sparse(zeros(dim, num_test));
test_labels = zeros(num_test, 1);

root = 'review_polarity/txt_sentoken';
pos_list = dir(fullfile(root, 'pos', 'cv*'));
pos_list = {pos_list.name};
neg_list = dir(fullfile(root, 'neg', 'cv*'));
neg_list = {neg_list.name};

pos_root = fullfile(root, 'pos');
neg_root = fullfile(root, 'neg');
for i = 0:1:999
    fprintf('Extracting feature for document %d...\n', i);
    if i <= 799
        % get positive
        name = pos_list{i+1};
        path = fullfile(pos_root, name);
        train_labels(i * 2 + 1) = 1;
        train_features(:, i * 2 + 1) = extract_feature(path, vocabulary);

        % get negative
        name = neg_list{i+1};
        path = fullfile(neg_root, name);
        train_labels(i*2 + 2) = 0;
        train_features(:, i*2 + 2) = extract_feature(path, vocabulary);
    else
         % get positive
        name = pos_list{i+1};
        path = fullfile(pos_root, name);
        test_labels(i * 2 + 1 - 1600) = 1;
        test_features(:, i * 2 + 1 - 1600) = extract_feature(path, vocabulary);

        % get negative
        name = neg_list{i+1};
        path = fullfile(neg_root, name);
        test_labels(i*2 + 2 - 1600) = 0;
        test_features(:, i*2 + 2 - 1600) = extract_feature(path, vocabulary);
    end
end

%% Classfication
% load data
load('train');
load('test');

% learn parameters
[params, prior] = learn(train_features, train_labels);

% predict on train set
train_predictions = zeros(size(train_features, 2), 1);
for i = 1:size(train_features, 2)
    fprintf('Predicting training sample %d...\n', i);
    train_predictions(i) = predict(train_features(:, i), params, prior);
end
```

```
% predict on test set
test_predictions = zeros(size(test_features, 2), 1);
for i = 1:size(test_features, 2)
    fprintf('Predicting test sample %d...\n', i);
    test_predictions(i) = predict(test_features(:, i), params, prior);
end
train_acc = sum(train_predictions == train_labels) / num_train;
test_acc = sum(test_predictions == test_labels) / num_test;
fprintf('Train accuracy: %f...\n', train_acc);
fprintf('Test accuracy: %f...\n', test_acc);
%ra get the most sentiment word
[~, idx] = sort(params(:, 1), 'descend');
pos_top10 = vocabulary(idx(1:10));
[~, idx] = sort(params(:, 2), 'descend');
neg_top10 = vocabulary(idx(1:10));

fprintf('Top 10 sentiment words (positive): \n');
for i = 1:10
    fprintf('%s\n', pos_top10{i});
end
fprintf('Top 10 sentiment words (negative): \n');
for i = 1:10
    fprintf('%s\n', neg_top10{i});
end
```

6.2 Utility Functions

```
function [ str ] = convert_int( i )
%CONVERT_INT Summary of this function goes here
%   Detailed explanation goes here
    if i < 10
        str = ['00' num2str(i)];
    elseif i <100
        str = ['0' num2str(i)];
    else
        str = num2str(i);
    end
end
```

```
function [ flag ] = endswith( str1, str2 )
%ENDSWITH Summary of this function goes here
%   Detailed explanation goes here
    flag = 0;
    if length(str1) >= length(str2)
        str1 = lower(str1);
        str2 = lower(str2);
        len = length(str2);
        s = str1(end - len + 1 :  end);
        if strcmp(s, str2)
            flag = 1;
        end
    end
end
```

```
function [ feature ] = extract_feature( path, vocabulary )
%EXTRACT_FEATURE Summary of this function goes here
%   Detailed explanation goes here
    feature = zeros(numel(vocabulary), 1);
    f = fopen(path, 'r');
    tokens = textscan(f, '%s');
```

```
    tokens = tokens{1};
    fclose(f);

    for i = 1:numel(tokens)
        [C2] = strcmp(tokens{i}, vocabulary);
        if sum(C2) >= 1
           idx = find(C2);
           feature(idx) = 1;
        end
    end
    feature = sparse(feature);
end


function [ feature ] = extract_feature_negation( path, vocabulary )
%EXTRACT_FEATURE Summary of this function goes here
%   Detailed explanation goes here
    feature = zeros(2 * numel(vocabulary), 1);
    f = fopen(path, 'r');
    tokens = textscan(f, '%s');
    tokens = tokens{1};
    fclose(f);
    num_words = numel(vocabulary);
    for i = 1:numel(tokens)
        [C2] = strcmp(tokens{i}, vocabulary);
        if sum(C2) >= 1
            idx = find(C2);
            if i > 1
                if endswith(tokens{i-1}, 'not') || endswith(tokens{i-1}, 'n''t')
                    feature(idx + num_words) = 1;
                else
                    feature(idx) = 1;
                end
            else
                feature(idx) = 1;
            end
        end
    end
    feature = sparse(feature);
end


function [ feature ] = extract_feature_bigram( path, vocabulary )
%EXTRACT_FEATURE Summary of this function goes here
%   Detailed explanation goes here
    feature = zeros(9 * numel(vocabulary), 1);
    f = fopen(path, 'r');
    tokens = textscan(f, '%s');
    tokens = tokens{1};
    fclose(f);
    num_words = numel(vocabulary);
    for i = 1:numel(tokens)
        [C2] = strcmp(tokens{i}, vocabulary);
        if sum(C2) >= 1
            idx = find(C2);
            if i > 1
                if endswith(tokens{i-1}, 'not') || endswith(tokens{i-1}, 'n''t')
                    feature(idx + num_words) = 1;
                elseif strcmp(tokens{i-1}, 'extremely')
                    feature(idx + num_words * 2) = 1;
                elseif strcmp(tokens{i-1}, 'quite')
                    feature(idx + num_words * 3) = 1;
                elseif strcmp(tokens{i-1}, 'just')
                    feature(idx + num_words * 4) = 1;
```

```
                elseif strcmp(tokens{i-1}, 'almost')
                    feature(idx + num_words * 5) = 1;
                elseif strcmp(tokens{i-1}, 'very')
                    feature(idx + num_words * 6) = 1;
                elseif strcmp(tokens{i-1}, 'too')
                    feature(idx + num_words * 7) = 1;
                elseif strcmp(tokens{i-1}, 'enough')
                    feature(idx + num_words * 8) = 1;
                else
                    feature(idx) = 1;
                end
            else
                feature(idx) = 1;
            end
        end
    end
    feature = sparse(feature);
end


function [ params, prior ] = learn( features, labels)
%CONDITIONAL Summary of this function goes here
%   Detailed explanation goes here
    dim = size(features, 1);
    params = zeros(dim, 2);
    alpha = 0.1;

    % p(x_i | y = 1)
    idx = find(labels == 1);
    params(:, 1) = (sum(features(:, idx), 2) ...
        + alpha * ones(dim , 1)) / (length(idx) + alpha * 2);

    % p(x_i | y = 0)
    idx = find(labels == 0);
    params(:, 2) = (sum(features(:, idx), 2) ...
        + alpha * ones(dim , 1)) / (length(idx) + alpha * 2);

    prior = sum(labels == 1) / length(labels);
end


function [ pred ] = predict( feature, params, prior )
%PREDICT Summary of this function goes here
%   Detailed explanation goes here
    pred = 1;
    dim = length(feature);
    logpos = log(prior);
    logneg = log(1 - prior);
    for i = 1:dim
        if feature(i) == 1
            logpos = logpos + log(params(i, 1));
        else
            logpos = logpos + log(1 - params(i, 1));
        end
    end

    for i = 1:dim
       if feature(i) == 1
            logneg = logneg + log(params(i, 2));
        else
            logneg = logneg + log(1 - params(i, 2));
        end
    end
```

```
    if logpos < logneg
        pred = 0;
    end
end
```

6.3 Negation Handling: Main Script

```
% 10701 Homework 1, Problem 5
%% Build the vocabulary
vocab_path = 'opinion-lexicon-English';
positive_words = {};
negative_words = {};

% read the positive words
path = fullfile(vocab_path, 'positive-words.txt');
f = fopen(path, 'r');
for i = 1:36
    word = fgetl(f);
end
while word ~= -1
   positive_words{end+1} = word;
   word = fgetl(f);
end
fclose(f);
% read the negative words
path = fullfile(vocab_path, 'negative-words.txt');
f = fopen(path, 'r');
for i = 1:36
    word = fgetl(f);
end
while -1 ~= word
   negative_words{end+1} = word;
    word = fgetl(f);
end
fclose(f);

vocabulary = [positive_words, negative_words]';
vocab_label = [ones(numel(positive_words), 1); zeros(numel(negative_words), 1)];
[vocabulary, idx] = sort(vocabulary);
vocab_label = vocab_label(idx);
% negation doubles the length of the vocabulary
dim = 2 * numel(vocabulary);

%% Extract features
train_ids = 0:799; num_train = length(train_ids) * 2;
test_ids = 800:999; num_test = length(test_ids) * 2;

train_features = sparse(zeros(dim, num_train));
train_labels = zeros(num_train, 1);
test_features = sparse(zeros(dim, num_test));
test_labels = zeros(num_test, 1);

root = 'review_polarity/txt_sentoken';
pos_list = dir(fullfile(root, 'pos', 'cv*'));
pos_list = {pos_list.name};
neg_list = dir(fullfile(root, 'neg', 'cv*'));
neg_list = {neg_list.name};

pos_root = fullfile(root, 'pos');
neg_root = fullfile(root, 'neg');
for i = 0:1:999
    fprintf('Extracting feature for document %d...\n', i);
    if i <= 799
```

```
        % positive
        name = pos_list{i+1};
        path = fullfile(pos_root, name);
        train_labels(i * 2 + 1) = 1;
        train_features(:, i * 2 + 1) = extract_feature_negation(path, vocabulary);

        % get negative
        name = neg_list{i+1};
        path = fullfile(neg_root, name);
        train_labels(i*2 + 2) = 0;
        train_features(:, i*2 + 2) = extract_feature_negation(path, vocabulary);
    else
        % get positive
        name = pos_list{i+1};
        path = fullfile(pos_root, name);
        test_labels(i * 2 + 1 - 1600) = 1;
        test_features(:, i * 2 + 1 - 1600) = extract_feature_negation(path, vocabulary);

        % get negative
        name = neg_list{i+1};
        path = fullfile(neg_root, name);
        test_labels(i*2 + 2 - 1600) = 0;
        test_features(:, i*2 + 2 - 1600) = extract_feature_negation(path, vocabulary);
    end
end

%% Classfication
% learn parameters
[params, prior] = learn(train_features, train_labels);

% predict on train set
train_predictions = zeros(size(train_features, 2), 1);
for i = 1:size(train_features, 2)
    fprintf('Predicting training sample %d...\n', i);
    train_predictions(i) = predict(train_features(:, i), params, prior);
end

% predict on test set
test_predictions = zeros(size(test_features, 2), 1);
for i = 1:size(test_features, 2)
    fprintf('Predicting test sample %d...\n', i);
    test_predictions(i) = predict(test_features(:, i), params, prior);
end
train_acc = sum(train_predictions == train_labels) / num_train;
test_acc = sum(test_predictions == test_labels) / num_test;
fprintf('Train accuracy: %f...\n', train_acc);
fprintf('Test accuracy: %f...\n', test_acc);

% get the most sentiment word
not_vocabulary = cellfun(@(x) ['not ' x], vocabulary, 'UniformOutput', false);
V = cat(1, vocabulary, not_vocabulary);
[~, idx] = sort(params(:, 1), 'descend');
pos_top10 = V(idx(1:10));
[~, idx] = sort(params(:, 2), 'descend');
neg_top10 = V(idx(1:10));

fprintf('Top 10 sentiment words (positive): \n');
for i = 1:10
    fprintf('%s\n', pos_top10{i});
end
fprintf('Top 10 sentiment words (negative): \n');
for i = 1:10
    fprintf('%s\n', neg_top10{i});
end
```

6.4 Including Bi-grams: Main Script

```
% 10701 Homework 1, Problem 5
%% Build the vocabulary
vocab_path = 'opinion-lexicon-English';
positive_words = {};
negative_words = {};

% read the positive words
path = fullfile(vocab_path, 'positive-words.txt');
f = fopen(path, 'r');
for i = 1:36
    word = fgetl(f);
end
while word ~= -1
   positive_words{end+1} = word;
   word = fgetl(f);
end
fclose(f);
% read the negative words
path = fullfile(vocab_path, 'negative-words.txt');
f = fopen(path, 'r');
for i = 1:36
    word = fgetl(f);
end
while -1 ~= word
   negative_words{end+1} = word;
    word = fgetl(f);
end
fclose(f);

vocabulary = [positive_words, negative_words]';
vocab_label = [ones(numel(positive_words), 1); zeros(numel(negative_words), 1)];
[vocabulary, idx] = sort(vocabulary);
vocab_label = vocab_label(idx);
% negation doubles the length of the vocabulary
dim = 9 * numel(vocabulary);

%% Extract features
train_ids = 0:799; num_train = length(train_ids) * 2;
test_ids = 800:999; num_test = length(test_ids) * 2;

train_features = sparse(zeros(dim, num_train));
train_labels = zeros(num_train, 1);
test_features = sparse(zeros(dim, num_test));
test_labels = zeros(num_test, 1);

root = 'review_polarity/txt_sentoken';
pos_list = dir(fullfile(root, 'pos', 'cv*'));
pos_list = {pos_list.name};
neg_list = dir(fullfile(root, 'neg', 'cv*'));
neg_list = {neg_list.name};

pos_root = fullfile(root, 'pos');
neg_root = fullfile(root, 'neg');
for i = 0:1:999
    fprintf('Extracting feature for document %d...\n', i);
    if i <= 799
        name = pos_list{i+1};
        path = fullfile(pos_root, name);
        train_labels(i * 2 + 1) = 1;
        train_features(:, i * 2 + 1) = extract_feature_bigram(path, vocabulary);
```

```
        % get negative
        name = neg_list{i+1};
        path = fullfile(neg_root, name);
        train_labels(i*2 + 2) = 0;
        train_features(:, i * 2 + 2) = extract_feature_bigram(path, vocabulary);
    else
        % get positive
        name = pos_list{i+1};
        path = fullfile(pos_root, name);
        test_labels(i * 2 + 1 - 1600) = 1;
        test_features(:, i * 2 + 1 - 1600) = extract_feature_bigram(path, vocabulary);

        % get negative
        name = neg_list{i+1};
        path = fullfile(neg_root, name);
        test_labels(i*2 + 2 - 1600) = 0;
        test_features(:, i*2 + 2 - 1600) = extract_feature_bigram(path, vocabulary);
    end
end

%% Classfication
load('train_bigram');
load('test_bigram');
% learn parameters
[params, prior] = learn(train_features, train_labels);

% predict on train set
train_predictions = zeros(size(train_features, 2), 1);
for i = 1:size(train_features, 2)
    fprintf('Predicting training sample %d...\n', i);
    train_predictions(i) = predict(train_features(:, i), params, prior);
end

% predict on test set
test_predictions = zeros(size(test_features, 2), 1);
for i = 1:size(test_features, 2)
    fprintf('Predicting test sample %d...\n', i);
    test_predictions(i) = predict(test_features(:, i), params, prior);
end
train_acc = sum(train_predictions == train_labels) / num_train;
test_acc = sum(test_predictions == test_labels) / num_test;
fprintf('Train accuracy: %f...\n', train_acc);
fprintf('Test accuracy: %f...\n', test_acc);

% get the most sentiment word
not_vocabulary = cellfun(@(x) ['not ' x], vocabulary, 'UniformOutput', false);
V = cat(1, vocabulary, not_vocabulary);
[~, idx] = sort(params(:, 1));
pos_top10 = V(idx(end-9:end));
[~, idx] = sort(params(:, 2));
neg_top10 = V(idx(end-9:end));

fprintf('Top 10 sentiment words (positive): \n');
for i = 1:10
    fprintf('%s\n', pos_top10{i});
end
fprintf('Top 10 sentiment words (negative): \n');
for i = 1:10
    fprintf('%s\n', neg_top10{i});
end
```