CS 464

Introduction to Machine Learning

Homework #1

Kerem Ayöz

21501569 / Section:2

1 The Chess Game

Question 1.1

(0.6) * (0.6) + (0.6) * (0.4) * (0.6) + (0.4) * (0.6) * (0.6) = **0.648**

(Wins first two) + (Draw after 2 games and wins final game)

Question 1.2

If he plays bold always in sudden death mode:
(0.65) * (0.65) * (0.6) = **0.2535**
(Draw + Draw + Win)

If he plays timid in all three games:
0
(Cannot win in that situation)

Question 1.3

(0.4) / (0.35 + 0.4) = **0.53**

(Played bold and lost) / (Played bold and lost + Played timid and lost)

Question 1.4

(0.75) * (0.85) + (0.25) * (0.35 + 0.4) = **0.825**

(Correct prediction and won) + (Wrong prediction and lost)

2 Medical Diagnosis

Question 2.1

P (S = disease) = **0.005**
P (S = healthy) = **0.995**
P (T = positive | S = disease) = **0.97**
P (T = negative | S = disease) = **0.03**
P (T = positive | S = healthy) = **0.02**
P (T = negative | S = healthy) = **0.98**

Question 2.2

P (S=disease | T=positive) = $\frac{P(T=positive \mid S=disease) * P\ (S=disease)}{P(T=positive)}$

P (S=disease | T=positive) = $\frac{(0.97)*(0.005)}{(0.97)*(0.005)+(0.02)*(0.995)} = \mathbf{0.196}$

3 MLE and MAP

Question 3.1

P (D | θ) = $\prod_{i=1}^{n} P(y_i \mid x_i, \theta) = \prod_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}$

L (θ) = $\sum_{i=1}^{n}(x_i \ln\lambda - \lambda - ln(x_i!))$

L' (θ) = $\sum_{i=1}^{n}\left(\frac{x_i}{\lambda} - 1\right) = 0$

$\hat{\boldsymbol{\lambda}}_{\mathbf{MLE}} = \overline{\boldsymbol{x}}$

Question 3.2

P (θ | D) ~ P (D | θ) P(θ) = $(\prod_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}) \frac{\beta^{\alpha} \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}$

L (θ) = $[\sum_{i=1}^{n}(x_i \ln\lambda - \lambda - ln(x_i!)] + \alpha\ln\beta + (\alpha - 1)\ln\lambda - \beta\lambda - \ln\Gamma(\alpha))$

L' (θ) = $[\sum_{i=1}^{n}\left(\frac{x_i}{\lambda} - 1\right)] + \frac{(\alpha-1)}{\lambda} - \beta = 0$

$$\frac{\bar{x}n}{\lambda} - n = \beta - \frac{(\alpha - 1)}{\lambda}, \qquad \bar{x}n - n\lambda = \beta\lambda - \alpha + 1, \qquad \overline{x}n + \alpha - 1 = \lambda(n + \beta)$$

$\hat{\boldsymbol{\lambda}}_{\mathbf{MAP}} = \frac{\bar{\boldsymbol{x}}\boldsymbol{n}+\boldsymbol{\alpha}-\mathbf{1}}{\boldsymbol{\beta}+\boldsymbol{n}}$

Question 3.3

P (θ | D) ~ P (D | θ) P(θ) = $\prod_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!} \frac{1}{b-a}$

L (θ) = $\sum_{i=1}^{n}(x_i \ln\lambda - \lambda - ln(x_i!) - \ln(b - a))$

L' (θ) = $\sum_{i=1}^{n}\left(\frac{x_i}{\lambda} - 1\right) = 0$

$\hat{\boldsymbol{\lambda}}_{\mathbf{MAP}} = \overline{\boldsymbol{x}}$ **(same with MLE estimator)**

4 Sentiment Analysis on Tweets

Question 4.1

It can be ignored since it is constant, it depends only on the distribution of the input features which is constant in terms of estimated label.

Question 4.2

Negative = 0.60544740437158, Neutral = 0.22344603825136 ,Positive = 0.17110655737704

Training set is skewed towards negative tweets, so it is unbalanced. This affects the MLE estimat ion since we also add P($Y = y_k$) to each term. One solution might be applying a k-fold cross valid ation which might give better results when algorithm left out a combination of k negative tweet. Also removing some of the negative labeled data points might be a solution however we need t o be careful while doing that since we might have lost data points that explains high variance in the dataset.

Question 4.3

We need (# of words) * (# of classes-1) + (# of labels -1) parameters to estimate which is 5722 * 2 = 11444 +2 parameters in our example.

Question 4.4

My test set accuracy is 0.6280737704918032. MLE is bad since some of the words in test sample s are not contained in training samples, algorithm is not be able to classify such results since we multiplied the probabilities of these words ($P(X_j \mid Y = y_k)$) as 0. This makes the probability as mi nus infinite therefore the tweet is not classified as class k even it is a labeled as class k in reality.

Question 4.5

My test set accuracy is 0.7530737704918032. Smoothing the probabilities will enable algorithm to be hallucinated, it behaves like it saw each word in example of each class of tweets at least o nce. Therefore, the probabilities in summation will never be minus infinite, just small numbers c ompared to the MLE case.

Question 4.6

My test set accuracy is 0.6431010928961749. Bernoulli model accuracy is close to Multinomial Model MLE estimation. The main reason is the occurrences of the words are maximum 2-3 on a verage in single tweet. Therefore, using Multinomial and Bernoulli models give close results. Mo del with Dirichlet prior is better than these two models since it also captures the non-existing w ords in tweets.

Question 4.7

Following words are the most common 20 words in the dataset;

flight
@united
@usairways
@americanair
@southwestair
@jetblue
cancelled
service
help
time
customer
hours
flights
hold
plane
delayed
gate
@virginamerica
call
flightled

Many of these words are expected words, since we used a dataset which contains Tweets about an airplane company.

Code

```
import numpy as np
import pandas as pd
np.warnings.filterwarnings('ignore')

# Load the dataset
test_feature = pd.read_csv("question-4-test-features.csv", header=None)
test_label = pd.read_csv("question-4-test-labels.csv", header=None)
train_feature = pd.read_csv("question-4-train-features.csv", header=None)
train_label = pd.read_csv("question-4-train-labels.csv", header=None)
labels = ["negative", "neutral", "positive"]

# Concatinated training data for multinomial and bernoulli
train_multinomial = pd.concat((train_feature, train_label[0].rename("label")), axis=1)
train_bernoulli = train_feature.copy()
train_bernoulli[train_bernoulli != 0] = 1
train_bernoulli = pd.concat((train_bernoulli, train_label[0].rename("label")), axis=1)

# Vocab is read
with open('question-4-vocab.txt', 'r', encoding="utf-8") as f:
    lines = f.readlines()
words = [word.split("\t")[0] for word in lines]
counts = [int(word.split("\t")[1][:-1]) for word in lines]

# Safe divide
def div(x, y):
    if y == 0:
        return 0
    return x / y

# Calculate the P(Y = yk) for each class
prior = [0, 0, 0]
label_counts = train_label[0].value_counts()
prior[0] = np.log(div(label_counts[0], len(train_feature)))  # negative
prior[1] = np.log(div(label_counts[1], len(train_feature)))  # neutral
prior[2] = np.log(div(label_counts[2], len(train_feature)))  # positive

# Total word counts in each class of documents
negative_words = train_multinomial.groupby("label").sum().sum(axis=1)[0]
neutral_words = train_multinomial.groupby("label").sum().sum(axis=1)[1]
positive_words = train_multinomial.groupby("label").sum().sum(axis=1)[2]
```

```
##################################################################################
##############################    MLE    ########################################
##################################################################################

# P(Xj | Y = yk) for each word Xj and each class yk, log is taken
pr_words_mle = np.full((len(words), 3), 0, dtype=float)

for i in range(len(words)):
    occurences = train_multinomial.iloc[:, [i, len(words)]].groupby("label").sum()[i]
    pr_words_mle[i][0] = np.log(div(occurences[0], negative_words))  # negative
    pr_words_mle[i][1] = np.log(div(occurences[1], neutral_words))  # neutral
    pr_words_mle[i][2] = np.log(div(occurences[2], positive_words))  # positive

# Calculate prediction values
pr_words_mle = np.nan_to_num(pr_words_mle)
results_mle = np.dot(test_feature, pr_words_mle) + np.array(prior)

# Fix the too big/small values
results_mle[results_mle > 1e100] = np.inf
results_mle[results_mle < -1e100] = -np.inf

# Predict
prediction_values = np.amax(results_mle, axis=1)
candidates = []
for i in range(len(prediction_values)):
    candidates.append(np.where(results_mle[i] == prediction_values[i])[0])
candidates = np.array(candidates)
predicted_mle = np.ones(np.shape(candidates), dtype=object)

for i in range(len(candidates)):
    if len(candidates[i]) > 1:  # tie
        predicted_mle[i] = labels[1]
    else:
        predicted_mle[i] = labels[candidates[i][0]]

# Accuracy
accuracy_mle = np.sum(predicted_mle == test_label[0].values) / len(test_label)
print("MLE accuracy: " + str(accuracy_mle))

##################################################################################
##############################    MAP    ########################################
##################################################################################

# P(Xj | Y = yk) for each word Xj and each class yk, log is taken, alpha=1
```

```
pr_words_map = np.full((len(words), 3), 0, dtype=float)

for i in range(len(words)):
    occurences = train_multinomial.iloc[:, [i, len(words)]].groupby("label").sum()[i]
    pr_words_map[i][0] = np.log(div(occurences[0] + 1, negative_words + len(words)))  # negative
    pr_words_map[i][1] = np.log(div(occurences[1] + 1, neutral_words + len(words)))  # neutral
    pr_words_map[i][2] = np.log(div(occurences[2] + 1, positive_words + len(words)))  # positive

# Calculate prediction values
pr_words_map = np.nan_to_num(pr_words_map)
results_map = np.dot(test_feature, pr_words_map) + np.array(prior)

# Fix the too big/small values
results_map[results_map > 1e100] = np.inf
results_map[results_map < -1e100] = -np.inf

# Predict
prediction_values = np.amax(results_map, axis=1)
candidates = []
for i in range(len(prediction_values)):
    candidates.append(np.where(results_map[i] == prediction_values[i])[0])
candidates = np.array(candidates)
predicted_map = np.ones(len(candidates), dtype=object)

for i in range(len(candidates)):
    if len(candidates[i]) > 1:  # tie
        predicted_map[i] = labels[1]
    else:
        predicted_map[i] = labels[candidates[i][0]]

# Accuracy
accuracy_map = np.sum(predicted_map == test_label[0].values) / len(test_label)
print("MAP accuracy: " + str(accuracy_map))

##########################################################################
##########################   Bernoulli  ###################################
##########################################################################

# P(Xj | Y = yk) for each word Xj and each class yk, log is taken, alpha=1
pr_words = np.full((len(words), 3), 0, dtype=float)

for i in range(len(words)):
    occurences = train_bernoulli.iloc[:, [i, len(words)]].groupby("label").sum()[i]
    pr_words[i][0] = div(occurences[0], label_counts[0])  # negative
```

```
    pr_words[i][1] = div(occurences[1], label_counts[1])  # neutral
    pr_words[i][2] = div(occurences[2], label_counts[2])  # positive

# Calculate prediction values
pr_words_comp = np.array(1) - pr_words
test_feature_comp = np.array(1) - test_feature
neg = test_feature * np.transpose(pr_words)[0] + test_feature_comp *
np.transpose(pr_words_comp)[0]
neu = test_feature * np.transpose(pr_words)[1] + test_feature_comp *
np.transpose(pr_words_comp)[1]
pos = test_feature * np.transpose(pr_words)[2] + test_feature_comp *
np.transpose(pr_words_comp)[2]

results_ber = np.transpose([np.sum(np.log(neg), axis=1), np.sum(np.log(neu), axis=1),
np.sum(np.log(pos), axis=1)]) + np.array(prior)

# Predict
prediction_values = np.amax(results_ber, axis=1)
candidates = []
for i in range(len(prediction_values)):
    candidates.append(np.where(results_ber[i] == prediction_values[i])[0])
candidates = np.array(candidates)
predicted_ber = np.ones(np.shape(candidates), dtype=object)
for i in range(len(candidates)):
    if len(candidates[i]) > 1:  # tie
        predicted_ber[i] = labels[1]
    else:
        predicted_ber[i] = labels[candidates[i][0]]

# Accuracy
accuracy_ber = np.sum(predicted_ber == test_label[0].values) / len(test_label)
print("Bernoulli accuracy: " + str(accuracy_ber))

# Find most common words
words = np.array([word.split("\t")[0] for word in lines])
counts = np.array([int(word.split("\t")[1][:-1]) for word in lines])

dataset = pd.DataFrame({'word': words, 'count': list(counts)}, columns=['word', 'count'])


common_ones = list(dataset.sort_values("count", ascending=False)[0:20]["word"])
print("Most common 20 words:")
for i in common_ones:
    print(i)
```