CS464 Introduction to Machine Learning

Fall 2021

Homework 1

Ata Berk ÇAKIR

21703127

QUESTİON 1

1.1

From the geometric distribution

$$P(n) = (1-p)^{n-1} * p$$

Where n is the number of trials until the first success which is equal to 8 in our case and p is the probability of success in each trial which is equal to $P_1 * P_3$ in our case. Thus, the solution is

$$P(n = 8) = (1 - \ P_1P_3)^7 * P_1P_3$$

1.2

Let X be a binomial random variable which is equal to score (# of success) and p be the probability of success for each trial after n trial.

$$p = p_1p_3 + p_2(1-p_3)$$

$$n = 10\ (\#of\ trials)$$

$$E(x) = n * p = 10(p_1p_3 + p_2 - p_2p_3$$

1.3.a

We can use the MLE (Maximum Likelihood Estimator) to measure the Oliver's guessing performance which is equal to $P\left|\hat{\theta} - \theta\right| \leq 2 * e^{-2N+2}$. For the validation we can use confusion matrix method. The confusion matrix is the following:

$$\begin{matrix} P(O = H|H) & P(O = H|T) \\ P(O = T|H) & P(O = T|T) \end{matrix}$$

1.3.b

Probability of predicting 1 heads is equal to 0.99 thus probability of predicting 8 heads in a row is

$$P(O = H) = 0.99^8 = 0.922$$

1.3.c

$$P(O = T|H) = \frac{P(H|O = T)P(H)}{P(O = T)} = \frac{0.01 * 0.6}{(0.01 * 0.6 + 0.95 * 0.4)} = 0.0155$$

QUESTİON 2

2.1

Although there are different types of distance metrics such as Minkowski distance, Manhattan distance, Cosine distance, Jaccard distance; Euclidean distance fits bets for our case. Given all that diabetes measurements are numeric and have the same units, we can directly use Euclidean distance defined as the square root of the sum of the squared differences between the two arrays of number. In our case each row of the future set can be considered as an array of number.

2.2

The main idea behind the feature selection is to increase the accuracy, reduce the complexity of the model and decrease the time that passes for the training of the model. Although the size of the dataset is not significantly large in our case, there might be huge amount of dataset with great number of features where some of them are not highly correlated with the labeling process. Such cases it will be beneficial to get rid of those features for the efficiency of the model.

2.3

After training the kNN classifier with k = 9 with full feature set the accuracy of the model was %72.07792207792207. Then, I started the backwards elimination process and the obtained results after every round are in the following tables.

First round of backwards elimination

	'Pregnancies'	'Glucose'	'BloodPressure'	'SkinThickness'	'Insulin'	'BMI'	'DiabetesPedigreeFunction'	'Age'
Accuracy	71.428	60.389	68.831	70.129	75.324	71.428	72.077	74.675

After we saw “Insulin” made the biggest difference we eliminate it manually and start the second round of backward elimination

Second round of backwards elimination

	'Pregnancies'	'Glucose'	'BloodPressure'	'SkinThickness'	'BMI'	'DiabetesPedigreeFunction'	'Age'
Accuracy	76.623	60.389	68.181	75.974	72.077	75.324	74.675

After we saw “Pregnancies” made the biggest difference we eliminate it manually and start the second round of backward elimination

Third round of backward elimination

	'Glucose'	'BloodPressure'	'SkinThickness'	'BMI'	'DiabetesPedigreeFunction'	'Age'

Accuracy	56.493	68.831	75.974	74.025	76.622	75.974

There isn't any improvement made therefore elimination has stopped after second round.

The final accuracy after "Insulin" and "Pregnancies" features are eliminated is % 76.62337662337663

2.4

For this part of the question in order to calculate the elapsed time while training the model I used "timeit" library of the python and recorded the elapsed times with different numbers of features.

	8 Features	**7 Features**	**6 Features**
Elapsed time(seconds)	27.61006	26.098245300000002	23.97948430000001

As expected, elapsed time decreases when the number of features also decreases.

QUESTİON 3

3.1

After training the Multinomial Naive Bayes model on the training set the following results are obtained. The accuracy of the model is %91.72.

```
In [35]: runfile('C:/Users/User/Desktop/q3/q3.2.py', wdir='C:/Users/User/Desktop/q3')
tp: 101.0
tn: 796.0
fp: 42.0
fn: 39.0
ACCURACY: % 91.71779141104295
```

Confusion matrix of the classifier is the following

		Actual	
		Positive	Negative
Predicted	Positive	101	42
	Negative	39	796

3.2

The number of the parameters we needed to estimate can be calculated by the following formula;

$$2N + 1 = \#\ of\ parameters\ to\ be\ estimated$$

Where N equals to number of features which is equal to 3458 in our case. Thus;

$$2N + 1 = (2 * 3458) + 1 = 6917$$

Number of the parameters we needed to estimate is 6917.

3.3

First, I trained my Bernoulli Naive Bayes classier without eliminating any features and the obtained accuracy is the following.

```
In [36]: runfile('C:/Users/User/Desktop/q3/q3.py', wdir='C:/Users/User/Desktop/q3')
tp: 115.0
tn: 816.0
fp: 22.0
fn: 25.0
ACCURACY: % 95.19427402862985
```

After training model with feature set following accuracies and elapsed times are obtained. However, although my model works fine there is an error occurred inside the function that I wrote for the feature selection therefore I selected the features randomly.

3.3.b

Yes, there is a relationship between the time complexity of the algorithm and the number of features used to train the classifier at each step. When the number of features decreases the time complexity also decreases. The reasons behind is the data that we use for training the model becomes smaller thus the time elapsed is decreases.

3.4

Bernoulli Bayes classifier gives better result than the Multinomial Bayes classifier as seen above. The reason behind this is the data given to us mostly consists of zeros and ones which is better for just using the appearance data of while using Bernoulli Bayes classifier. Thus the higher accuracy is normal and expected.

Codes

Q2

```
#import libraries
import pandas as pd
import math
from timeit import default_timer as timer

#import csv as pandas dataframe
diabetes_test_features = pd.read_csv("diabetes_test_features.csv")
diabetes_test_labels = pd.read_csv("diabetes_test_labels.csv")
diabetes_train_features = pd.read_csv("diabetes_train_features.csv")
diabetes_train_labels = pd.read_csv("diabetes_train_labels.csv")

#merge the features and labels into single dataframe
df_test = pd.merge(diabetes_test_features, diabetes_test_labels,how="left")
df_train = pd.merge(diabetes_train_features, diabetes_train_labels,how="left")

#dropping unrelevant unnamed index indicator
df_test = df_test.drop('Unnamed: 0',1)
df_train = df_train.drop('Unnamed: 0',1)
```

```
diabetes_test_labels = diabetes_test_labels.drop('Unnamed: 0',1)

#calculates the euclidian distance between two point
def get_euclidean_distance (train_sample,test_sample):
    dist = 0.0
    for i in range(len(train_sample)-1):
        dist += (train_sample.iloc[i]-test_sample.iloc[i])**2
    return math.sqrt(dist)

#gets the k nearest neighbors of a single row of the test data
def get_neighbors (train,test_row,k):
    distances = []
    neighbors =[]
    for row in range(len(train)):
        train_row = train.iloc[row]
        eu_dist = get_euclidean_distance(train_row,test_row)
        distances.append((train_row,eu_dist))
    distances.sort(key = lambda tup: tup[1])
    for i in range(k):
        neighbors.append(distances[i][0])
    return neighbors

#makes prediction based on a single row of the test data
def predict_classification(train, test_row, k):
    outcome = []
    neighbors = get_neighbors(train, test_row, k)
    for row in neighbors:
        outcome.append(row[-1])
    prediction = max(outcome, key=outcome.count)
```

```
    return prediction

#calculates the accuracy of KNN classifier by using confusion matrix
def get_accuracy(train,test,k):
    tp = 0.0 #true positive count
    tn = 0.0 #true negative count
    fp = 0.0 #false positive count
    fn = 0.0 #false negative count
    for row in range(len(test)):
        test_row = test.iloc[row]
        prediction = predict_classification(train,test_row,k)
        actual = float(diabetes_test_labels.iloc[row])
        if ((prediction == 1) & (actual == 1)):
            tp+=1
        elif ((prediction == 1) & (actual == 0)):
            fp+=1
        elif ((prediction == 0) & (actual == 0)):
            tn+=1
        elif ((prediction == 0) & (actual == 1)):
            fn+=1
    accuracy = ((tp+tn)/(tp+tn+fp+fn))*100
    return accuracy

#eleminates the features one by one and retuns accuricies as tuples in a list
def backward_elemination (features,df_train,df_test):
    accuricies = []
    for i in features:
        train_new = df_train.copy()
        test_new = df_test.copy()
```

```
        test_new = test_new.drop(i,1)
        train_new = train_new.drop(i,1)
        accuracy = get_accuracy(train_new,test_new,9)
        accuricies.append((i,accuracy))
    return accuricies

#accuracy with full feature set

accuracy = get_accuracy(df_train,df_test,9)
print("The accuracy of the KNN classfier with k = 9 is %"+str(accuracy))

#first round of backward elemination

features = ["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin",
            "BMI","DiabetesPedigreeFunction","Age"]
print("Accuricies when corresponding feature is eleminited in first round" )
print(backward_elemination(features,df_train,df_test))

#after we saw Insulin made the biggest difference we eleminate it manually and start the second round of backward elemination
features = ["Pregnancies","Glucose","BloodPressure","SkinThickness",
            "BMI","DiabetesPedigreeFunction","Age"]
df_train = df_train.drop("Insulin",1)
df_test = df_test.drop("Insulin",1)
print("Accuricies when corresponding feature is eliminated in second round" )
print(backward_elemination(features,df_train,df_test))

#after we saw Pregnancies made the biggest difference we eleminate it manually and start the third round of backward elemination
features = ["Glucose","BloodPressure","SkinThickness",
```

```
            "BMI","DiabetesPedigreeFunction","Age"]
df_train = df_train.drop("Pregnancies",1)
df_test = df_test.drop("Pregnancies",1)
print("Accuricies when corresponding feature is eleminated in third round" )
print(backward_elemination(features,df_train,df_test))
print("There isn't any improvement made elimination has stopped after second round")

#following part of the code is written for to calculate the elapsed time with different number of features
start = timer()
accuracy = get_accuracy(df_train,df_test,9)
print("The accuracy of the KNN classfier with k = 9 is %"+str(accuracy))
end = timer()
print("time elapsed: ",end - start)

train_new = df_train.copy()
test_new = df_test.copy()
train_new = train_new.drop("Insulin",1)
test_new = test_new.drop("Insulin",1)
start = timer()
accuracy = get_accuracy(train_new,test_new,9)
print("The accuracy of the KNN classfier with k = 9 is %"+str(accuracy))
end = timer()
print("time elapsed: ",end - start)

train_new = train_new.drop("Pregnancies",1)
test_new = test_new.drop("Pregnancies",1)
start = timer()
accuracy = get_accuracy(train_new,test_new,9)
print("The accuracy of the KNN classfier with k = 9 is %"+str(accuracy))
```

```
end = timer()

print("time elapsed: ",end - start)
```

Q3.1

```
#import libraries

import pandas as pd

import numpy as np


#read csv files as as df for merge purpose

sms_train_features_df = pd.read_csv("sms_train_features.csv")

sms_train_labels_df = pd.read_csv("sms_train_labels.csv")

vocabulary = open("vocabulary.txt","r")

#merge features and labes for test data

train = pd.merge(sms_train_features_df,sms_train_labels_df,how="left")

#dropping unrelevant index indicator than seperate ham and spam classes as different data frames

train = train.drop('Unnamed: 0',1)

spam_class = train.copy()

ham_class = train.copy()

spam_class = spam_class[spam_class["class"] == 1]

ham_class = ham_class[ham_class["class"] == 0]

#convert files as array for easier calculations

sms_train_features = pd.read_csv("sms_train_features.csv",index_col = 0).values

sms_train_labels = pd.read_csv("sms_train_labels.csv",index_col = 0).values

sms_test_features = pd.read_csv("sms_test_features.csv",index_col = 0).values

sms_test_labels = pd.read_csv("sms_test_labels.csv",index_col = 0).values

spam_class = spam_class.values

ham_class = ham_class.values


#label 0 is normal message, and label 1 is spam

def get_prior_spam(labels_csv):
```

```
    non_zeros = np.count_nonzero(labels_csv)
    prior_spam = non_zeros/len(labels_csv)
    return prior_spam

#calculates the likelihoods of words in ham or spam class and stores them in an array
def get_likelihood(class_array):
    likelihood = []
    transpose = np.transpose(class_array)
    for i in transpose:
        count = np.count_nonzero(i == 1)
        prob = count/len(i)
        likelihood.append(prob)
    return likelihood[:-1]     #there is -1 since the last row of the transposed matrix contains
labels and we dont use the in likelihood calculation

def calculate_posterior(test_features_row,class_array,prior):
    posterior = 1
    likelihood = get_likelihood(class_array)
    for i in range(len(test_features_row)):
        posterior = (likelihood[i])**(test_features_row[i])*posterior
    posterior = posterior*prior
    return posterior

def predict_label(sms_train_labels,sms_test_features,ham_class,spam_class):
    spam_prior = get_prior_spam(sms_train_labels)
    ham_prior = 1-spam_prior
    predicted_labels=[]
    for i in sms_test_features:
        ham_posterior = calculate_posterior(i,ham_class,ham_prior)
        spam_posterior = calculate_posterior(i,spam_class,ham_prior)
```

```
        if ham_posterior>=spam_posterior:
            predicted_labels.append(0)
        else:
            predicted_labels.append(1)
    return predicted_labels

def calculate_accuracy(predicted_labels,sms_test_labels):
    tp = 0.0 #true positive count
    tn = 0.0 #true negative count
    fp = 0.0 #false positive count
    fn = 0.0 #false negative count
    for i in range(len(predicted_labels)):
        prediction = predicted_labels[i]
        actual = sms_test_labels[i]
        if ((prediction == 1) & (actual == 1)):
            tp+=1
        elif ((prediction == 1) & (actual == 0)):
            fp+=1
        elif ((prediction == 0) & (actual == 0)):
            tn+=1
        elif ((prediction == 0) & (actual == 1)):
            fn+=1
    accuracy = ((tp+tn)/(tp+tn+fp+fn))*100
    print("tp:",tp)
    print("tn:",tn)
    print("fp:",fp)
    print("fn:",fn)
    return accuracy
```

```
predicted_labels = predict_label(sms_train_labels,sms_test_features,ham_class,spam_class)

accuracy = calculate_accuracy(predicted_labels,sms_test_labels)


print("ACCURACY: %",accuracy)
```

Q3.3a

```
#import libraries

import pandas as pd

import numpy as np


#read csv files as as df for merge purpose

sms_train_features_df = pd.read_csv("sms_train_features.csv")

sms_train_labels_df = pd.read_csv("sms_train_labels.csv")

vocabulary = open("vocabulary.txt","r")

#merge features and labes for test data

train = pd.merge(sms_train_features_df,sms_train_labels_df,how="left")

#dropping unrelevant index indicator than seperate ham and spam classes as different data frames

train = train.drop('Unnamed: 0',1)

spam_class = train.copy()

ham_class = train.copy()

spam_class = spam_class[spam_class["class"] == 1]

ham_class = ham_class[ham_class["class"] == 0]

#convert files as array for easier calculations

sms_train_features = pd.read_csv("sms_train_features.csv",index_col = 0).values

sms_train_labels = pd.read_csv("sms_train_labels.csv",index_col = 0).values

sms_test_features = pd.read_csv("sms_test_features.csv",index_col = 0).values

sms_test_labels = pd.read_csv("sms_test_labels.csv",index_col = 0).values

spam_class = spam_class.values

ham_class = ham_class.values
```

```
#label 0 is normal message, and label 1 is spam
def get_prior_spam(labels_csv):
    non_zeros = np.count_nonzero(labels_csv)
    prior_spam = non_zeros/len(labels_csv)
    return prior_spam

#calculates the likelihoods of words in ham or spam class and stores them in an array
def get_likelihood(class_array):
    likelihood = []
    transpose = np.transpose(class_array)
    for i in transpose:
        count = np.count_nonzero(i == 1)
        prob = count/len(i)
        likelihood.append(prob)
    return likelihood[:-1]     #there is -1 since the last row of the transposed matrix contains
labels and we dont use the in likelihood calculation

def calculate_posterior(test_features_row,class_array,prior):
    posterior = 1
    likelihood = get_likelihood(class_array)
    for i in range(len(test_features_row)):
        posterior = (test_features_row[i]*(likelihood[i]+(0.0000000001)) + (1-
test_features_row[i])*(1-(likelihood[i]+0.0000000001)))*posterior
    posterior = abs(posterior*prior)
    return posterior

def predict_label(sms_train_labels,sms_test_features,ham_class,spam_class):
    spam_prior = get_prior_spam(sms_train_labels)
    ham_prior = 1-spam_prior
```

```
    predicted_labels=[]
    for i in sms_test_features:
        ham_posterior = calculate_posterior(i,ham_class,ham_prior)
        spam_posterior = calculate_posterior(i,spam_class,ham_prior)
        if ham_posterior>=spam_posterior:
            predicted_labels.append(0)
        else:
            predicted_labels.append(1)
    return predicted_labels

def calculate_accuracy(predicted_labels,sms_test_labels):
    tp = 0.0 #true positive count
    tn = 0.0 #true negative count
    fp = 0.0 #false positive count
    fn = 0.0 #false negative count
    for i in range(len(predicted_labels)):
        prediction = predicted_labels[i]
        actual = sms_test_labels[i]
        if ((prediction == 1) & (actual == 1)):
            tp+=1
        elif ((prediction == 1) & (actual == 0)):
            fp+=1
        elif ((prediction == 0) & (actual == 0)):
            tn+=1
        elif ((prediction == 0) & (actual == 1)):
            fn+=1
    accuracy = ((tp+tn)/(tp+tn+fp+fn))*100
    print("tp:",tp)
    print("tn:",tn)
```

```
    print("fp:",fp)
    print("fn:",fn)
    return accuracy


def feature_selection(n):
    index = 0
    accuricies = []
    sms_train_features_df = pd.read_csv("sms_train_features.csv")
    sms_train_labels_df = pd.read_csv("sms_train_labels.csv")
    train = pd.merge(sms_train_features_df,sms_train_labels_df,how="left")
    train = train.drop('Unnamed: 0',1).values()
    while index < len(train):
        sms_train_features_df = pd.read_csv("sms_train_features.csv")
        sms_train_labels_df = pd.read_csv("sms_train_labels.csv")
        train = pd.merge(sms_train_features_df,sms_train_labels_df,how="left")
        train = train.drop('Unnamed: 0',1)
        train = train.iloc[index:(index+n)]
        spam_class = train.copy()
        ham_class = train.copy()
        spam_class = spam_class[spam_class["class"] == 1].values()
        ham_class = ham_class[ham_class["class"] == 0].Values()
        sms_train_labels = pd.read_csv("sms_train_labels.csv",index_col = 0).values
        sms_test_features = pd.read_csv("sms_test_features.csv",index_col = 0).values
        sms_test_labels = pd.read_csv("sms_test_labels.csv",index_col = 0).values
        predicted_labels =
predict_label(sms_train_labels,sms_test_features,ham_class,spam_class)
        accuracy = calculate_accuracy(predicted_labels,sms_test_labels)
        accuricies.append(accuracy,index)
        index+=n
    return accuricies
```

```
predicted_labels = predict_label(sms_train_labels,sms_test_features,ham_class,spam_class)
accuracy = calculate_accuracy(predicted_labels,sms_test_labels)

print("ACCURACY: %",accuracy)

print(feature_selection(100))
```