Umut Mücahit Köksaldı

21402234

31.10.2017

CS464 – 01

Homework 1

1.

1.1 W = random variable depicting the outcome of the game
S = random variable depicting the machine the game was played on.

$$P(S=\text{Venus} \mid W=\text{win}) = \frac{P(W=\text{win} \mid S=\text{Venus})\, P(S=\text{Venus})}{P(W=\text{win})}$$

$$P(S=\text{Venus}) = \frac{200}{1380} = \frac{20}{138} \;;\; P(W=\text{win}) = \frac{360}{1380} = \frac{36}{138}$$

$$P(W=\text{win} \mid S=\text{Venus}) = \frac{80}{200} \quad \therefore \quad \boxed{P(S=\text{Venus} \mid W=\text{win}) = \frac{2}{9}}$$

1.2 Probability of win on Venus $\Longrightarrow$ For you: $\frac{50}{100} = 0.5$, for friend: $\frac{30}{100} = 0.3$
Probability of win on Mars $\Longrightarrow$ For you: $\frac{260}{1090} \cong 0.239$, for friend: $\frac{20}{90} \cong 0.222\ldots$
On both machines, you are more likely to win.

1.3 Winning probability of you: $\frac{310}{1180} \cong 0.2627$
Winning probability of friend: $\frac{50}{190} \cong 0.2631$
Friend is more likely to win overall

1.4 The results seem contradicting, since in 1.2 we found that your chances to win were higher, however in 1.3 it is calculated that the friend's overall chance of wining is higher. This is due to the fact that your losses are a lot higher than 'friend's on Mars, which causes the denominator to grow more rapidly and as a result, your wins on Venus do not affect the calculation too much, even though the ratio is considerably higher for 'you' on Venus.

2.

2.1

To find the maximum likelihood estimator for λ;

$$P(x_1, x_2 \cdots x_n \mid \lambda) = \frac{e^{-\lambda}\lambda^{x_1}}{x_1!} \cdots \frac{e^{-\lambda}\lambda^{x_n}}{x_n!} = \frac{e^{-n\lambda}\lambda^{\Sigma x_i}}{x_1! \cdots x_n!}$$

$$\ln(P) = -n\lambda + (\ln\lambda)\Sigma x_i - \ln\left(\prod x_i!\right)$$

$$\underbrace{\frac{d(\ln(P))}{d\lambda} = -n + \frac{\Sigma x_i}{\lambda} = 0}_{\text{Derivative} = 0 \text{ to find the argmax}} \Longrightarrow \boxed{\hat{\lambda} = \frac{\sum_{i=1}^{n} x_i}{n}}$$

2.2

$$\lambda_{MAP} = \operatorname{argmax} P(\lambda \mid x_1, x_2, x_3 \cdots x_n)$$

$$= \operatorname{argmax} \frac{P(\overbrace{x_1, x_2, x_3 \cdots x_n}^{D} \mid \lambda)\, P(\lambda)}{\underbrace{P(D)}_{\text{constant with respect to}}}$$

$$P(x_1, x_2, \cdots x_n \mid \lambda) = \frac{e^{-n\lambda}\lambda^{\Sigma x_i}}{x_1! \cdots x_n!} \quad \text{from (2.1)}$$

$$P(\lambda) = \frac{1}{2\sqrt{2\pi}} e^{-\lambda^2/8}$$

$$\ln\left(P(D \mid \lambda) P(\lambda)\right) = \left(-n\lambda + (\ln\lambda)\Sigma x_i - \ln\left(\prod x_i!\right)\right) + \left(\frac{-\lambda^2}{8}\right) - \ln\left(2\sqrt{2\pi}\right) = A$$

$$\underbrace{\frac{dA}{d\lambda}}_{\substack{\text{Derivative} = 0\\ \text{to find}\\ \text{argmax}}} = -n + \frac{\Sigma x_i}{\lambda} - \frac{\lambda}{4} = 0 \Longrightarrow \underbrace{\frac{\lambda^2}{4} + n\lambda + (-\Sigma x_i) = 0}$$

$\boxed{\hat{\lambda}_{MAP} \text{ is the root of this quadratic equation}}$

$$\hat{\lambda}_{MAP} = \frac{-n \pm \sqrt{n^2 + \Sigma x_i}}{\frac{1}{2}} = \boxed{2(-n \pm n^2 + \Sigma x_i)}$$

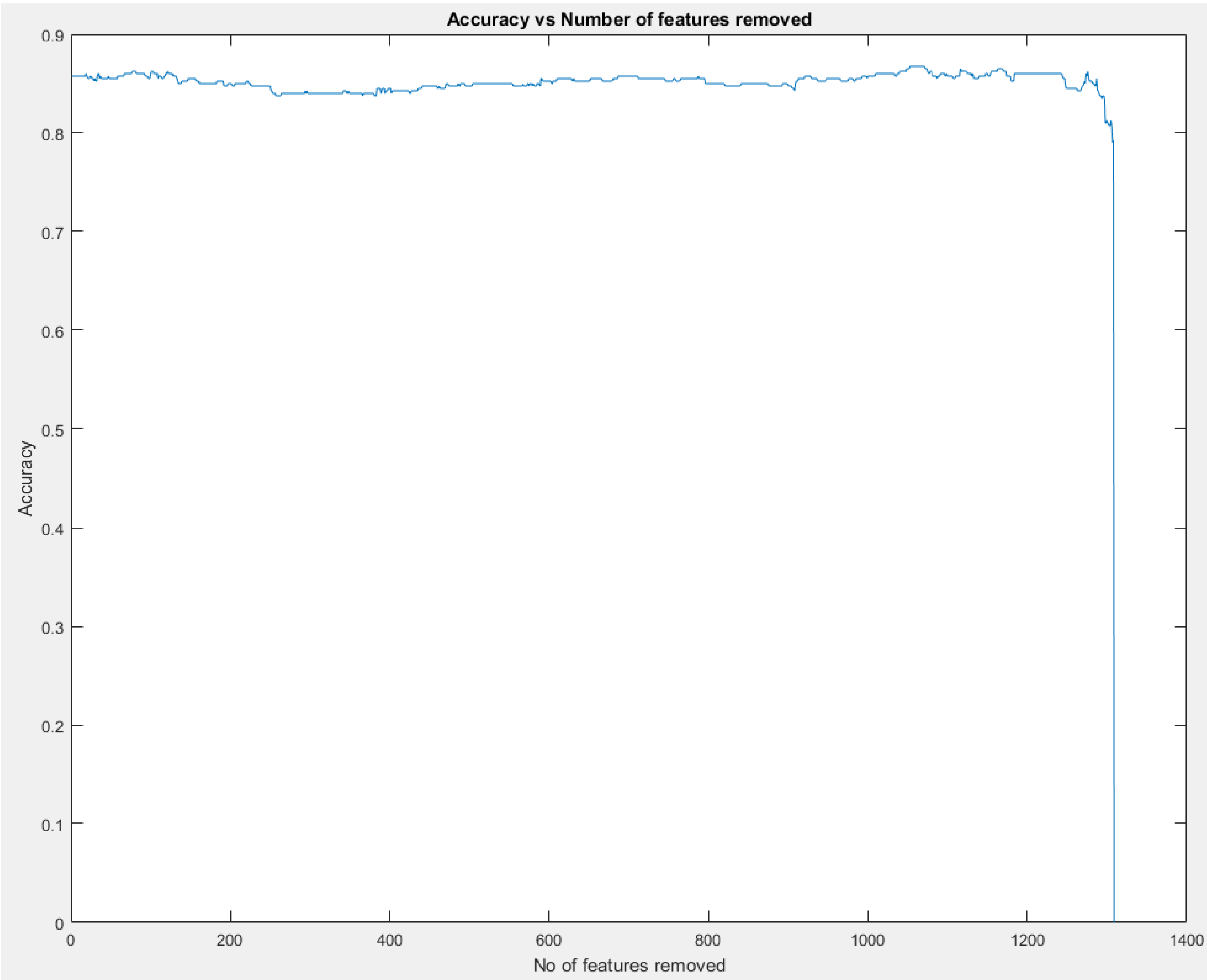
3.

3.1. The two models are equivalent because we do not care about the location of the words, we only need number of occurrences. In the first model, if a word appears more than once in the document, we check them more than once and multiply their probabilities. In the second model, we actually do the same work but instead of continuing to check all of the document, we simply take the probability of the first encountered word and raise it to the exponent of the number of occurrences of said word, thus calculating the same result.

3.2. We can ignore the denominator because it will be the same value for all calculated values of y. Since we are only interested in finding maximum y for our prediction, we do not need to calculate the exact value, we only need to calculate enough to compare it to other estimations of y. We then make the prediction depending on the greatest y value. If we had needed to know the exact probability of y, we would need to calculate the denominator.

3.3. Our data set is perfectly balanced with 700 student webpages and 700 faculty web pags in the dataset. The train and test portions are also perfectly balanced because the training set contains 500 student websites and 500 faculty websites, and the test set contains 200 student websites and 200 faculty websites.

3.6. After calculating the mutual information of the features in our dataset, we remove the features one by one starting from the least important until we are left with only one feature. The graph showing the accuracy after each removal is displayed below. Our initial accuracy is 0.8575, and the maximum accuracy we were able to obtain is 0.8675 and we have reached this value after taking out 700 features. Towards the end while taking out the last ~70 features, the accuracy continued to go down to 0.84 since we were taking out more important features.



Appendix (MATLAB Code)

```
% Author: Umut Mucahit Koksaldi
% ID: 21402234
% CS464 Assignment 1

function main(path)
    % QUESTION 3.4

    % import and pre-process the data
    train_file = fullfile(path, 'traindata.txt');
    test_file = fullfile(path, 'testdata.txt');
    train_data = importdata(train_file);
    test_data = importdata(test_file);
    X_train = train_data.data;
    y_train = train_data.textdata;
    X_test = test_data.data;
    y_test = test_data.textdata;
    y_train_categorical = zeros([1000 1]);
    y_test_categorical = zeros([400 1]);

    % replace categorical data in training set
    for i = 1:1000
        if strcmp(y_train(i), 'student')
            y_train_categorical(i) = 1;
        else
            y_train_categorical(i) = 0;
        end
    end

    % replace categorical data in test set
    for i = 1:400
        if strcmp(y_test(i), 'student')
            y_test_categorical(i) = 1;
        else
            y_test_categorical(i) = 0;
        end
    end

    % split the final categorical data into student and faculty subsbets
    X_train_student = X_train(1:500, :);
    X_train_faculty = X_train(501:1000, :);
    y_train_student = y_train(1:500, :);
    y_train_faculty = y_train(501:1000, :);
```

```
student_thetas = zeros([1 1309]);
faculty_thetas = zeros([1 1309]);
% determine student thetas
for i = 1:1309
    T_s = sum(X_train_student(:, i));
    totalSum = sum(sum(X_train_student));
    student_thetas(1, i) = T_s / totalSum;
end
% determine faculty thetas
for i = 1:1309
    T_s = sum(X_train_faculty(:, i));
    totalSum = sum(sum(X_train_faculty));
    faculty_thetas(1, i) = T_s / totalSum;
end

pred = zeros([400 1]);

% make predictions on the test set
for i = 1:400
    sum1 = 0;
    sum2 = 0;
    for j = 1:1309
        % check if the expression is not a number for the case of 0 * log(0)
        if ~isnan(X_test(i, j) * log(student_thetas(1, j)))
            sum1 = sum1 + X_test(i, j) * log(student_thetas(1, j));

        end
        % check if the expression is not a number for the case of 0 * log(0)
        if ~isnan(X_test(i, j) * log(faculty_thetas(1, j)))
            sum2 = sum2 + X_test(i, j) * log(faculty_thetas(1, j));
        end
    end
    % calculate the probabilities of student and faculty class
    argmaxY1 = log(1/2) + sum1;
    argmaxY0 = log(1/2) + sum2;
    % assign the class
    if argmaxY1 > argmaxY0
        pred(i, 1) = 1;
    else
        pred(i, 1) = 0;
    end
end

% construct the confusion matrices
```

```
    confusion_student = confusionmat(y_test_categorical(1:200, 1), pred(1:200,
1));
    confusion_faculty = confusionmat(y_test_categorical(201:400, 1),
pred(201:400, 1));

    % report accuracy from the confusion matrices
    accuracy_student = confusion_student(2, 2) / sum(sum(confusion_student));
    accuracy_faculty = confusion_faculty(1, 1) / sum(sum(confusion_faculty));

    display(accuracy_student);
    display(accuracy_faculty);

    % QUESTION 3.5
    % find the mutual information relationships
    mutual_information_matrix = zeros([1 1309]);

    accuracy_matrix = zeros([1 1309]);

    for i = 1:1309
        n = 1000;
        % calculate parameters
        n11 = sum(X_train_student(:,i)~=0);
        n10 = sum(X_train_student(:,i)==0);
        n01 = sum(X_train_faculty(:,i)~=0);
        n00 = sum(X_train_faculty(:,i)==0);

        % calculate the mutual information
        if ~isnan(((n11/n) * log2(n*n11 / ((n10 + n11) * (n10 + n11)))) + ((n01 /
n) * log2(n*n01 / ((n01 + n00) * (n01 + n11)))) + ((n10 / n) * log2(n*n10 / ((n10
+ n11) * (n10 + n00)))) + ((n00 / n) * log2(n*n00 / ((n00 + n01) * (n10 +
n00)))))
            mutual_inf = ((n11/n) * log2(n*n11 / ((n10 + n11) * (n10 + n11)))) +
((n01 / n) * log2(n*n01 / ((n01 + n00) * (n01 + n11)))) + ((n10 / n) * log2(n*n10
/ ((n10 + n11) * (n10 + n00)))) + ((n00 / n) * log2(n*n00 / ((n00 + n01) * (n10 +
n00))));
        else
            mutual_inf = 0;
        end
        mutual_information_matrix(1, i) = mutual_inf;
    end

    % 10 most important features' indices
    sorted_mi = sort(mutual_information_matrix, 'descend');
    sorted_mi_t = sorted_mi.';
```

```
    sorted_mi_mod =
reshape(sorted_mi_t(~isnan(sorted_mi_t)),[],size(sorted_mi,1)).';
    sorted_mi = sorted_mi_mod(1, 1:10);

    % indices of the 10 most important features stored in the array
    mi_top10_indices =
arrayfun(@(x)find(mutual_information_matrix==x,1),sorted_mi);
    disp('Indices of 10 most important features:');
    disp(mi_top10_indices);

    % QUESTION 3.6
    % find indices of the least important features
    ascending_sorted = sort(mutual_information_matrix, 'ascend');
    ascending_indices =
arrayfun(@(x)find(mutual_information_matrix==x,1),ascending_sorted);

    accuracy_matrix = zeros([1 1309]);

    % remove features at indices specified at ascending_indices, starting from
    % the last impornt one
    for i = 1:1308
        % reprocess data, removing features at specified indices
        removed_index = ascending_indices(1, i);
        X_train_student(:, removed_index) = [0];
        X_train_faculty(:, removed_index) = [0];
        X_test(:, removed_index) = [0];
        disp(i);
        % recalculate thetas with the removed feature
        % determine student thetas
        for j = 1:(1309)
            T_s = sum(X_train_student(:, j));
            totalSum = sum(sum(X_train_student));
            student_thetas(1, j) = T_s / totalSum;
        end
        % determine faculty thetas
        for j = 1:(1309)
            T_s = sum(X_train_faculty(:, j));
            totalSum = sum(sum(X_train_faculty));
            faculty_thetas(1, j) = T_s / totalSum;
        end

        % make predictions on the test data again
        for k = 1:400
            sum1 = 0;
            sum2 = 0;
```

```
            for j = 1:(1309)
                % check if the expression is not a number for the case of 0 *
log(0)
                if ~isnan(X_test(k, j) * log(student_thetas(1, j)))
                    sum1 = sum1 + X_test(k, j) * log(student_thetas(1, j));

                end
                % check if the expression is not a number for the case of 0 *
log(0)
                if ~isnan(X_test(k, j) * log(faculty_thetas(1, j)))
                    sum2 = sum2 + X_test(k, j) * log(faculty_thetas(1, j));
                end
            end
            % calculate the probabilities of student and faculty class
            argmaxY1 = log(1/2) + sum1;
            argmaxY0 = log(1/2) + sum2;
            % assign the class
            if argmaxY1 > argmaxY0
                pred(k, 1) = 1;
            else
                pred(k, 1) = 0;
            end
        end

        % construct the confusion matrices
        confusion_student = confusionmat(y_test_categorical(1:200, 1),
pred(1:200, 1));
        confusion_faculty = confusionmat(y_test_categorical(201:400, 1),
pred(201:400, 1));

        accuracy_matrix(1, i) = ((confusion_faculty(1,1) +
confusion_student(2,2)) / (sum(sum(confusion_student)) +
sum(sum(confusion_faculty))));

    end
    % plot the accuracies
    plot(accuracy_matrix)
end
```