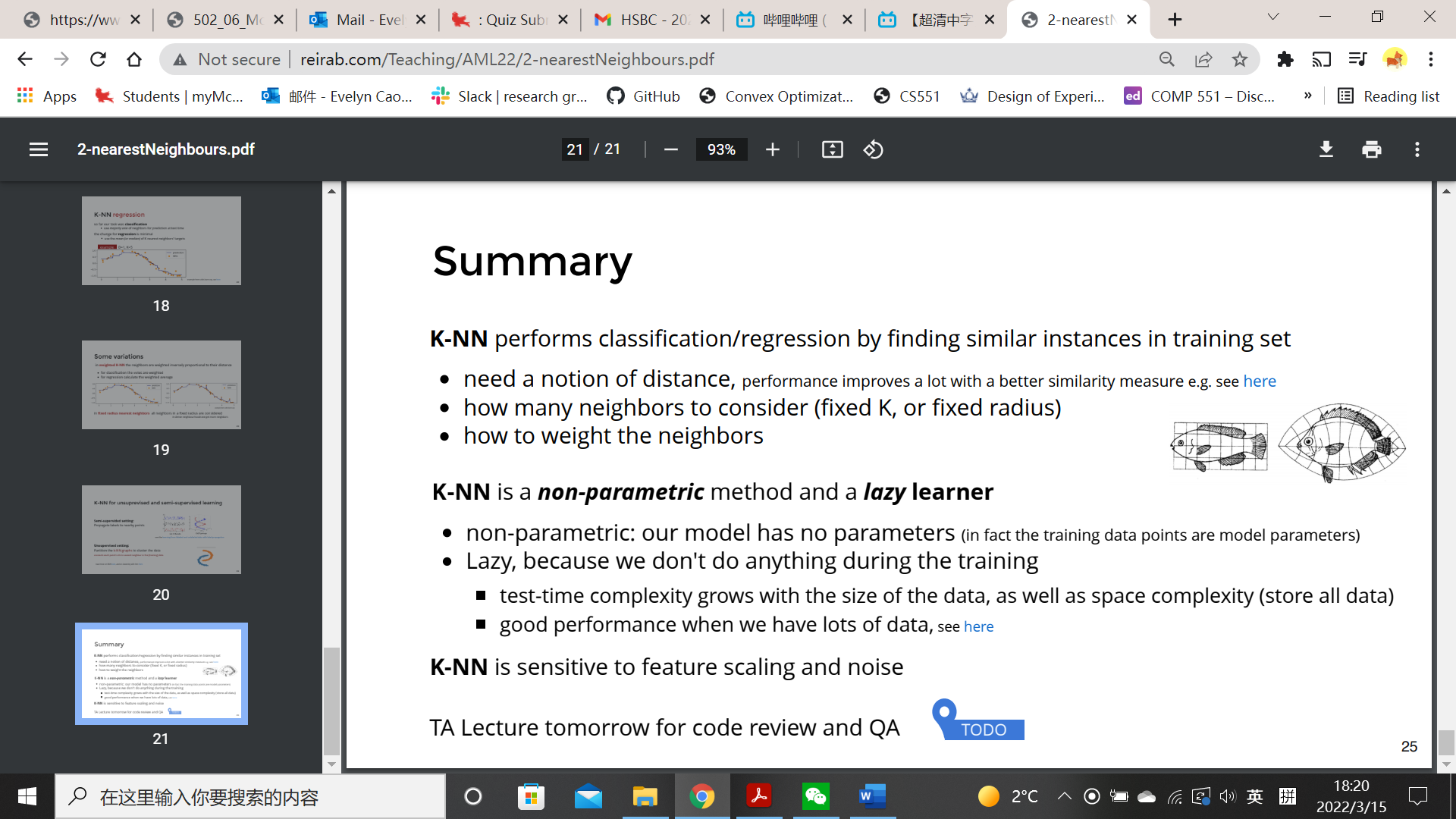
KNN

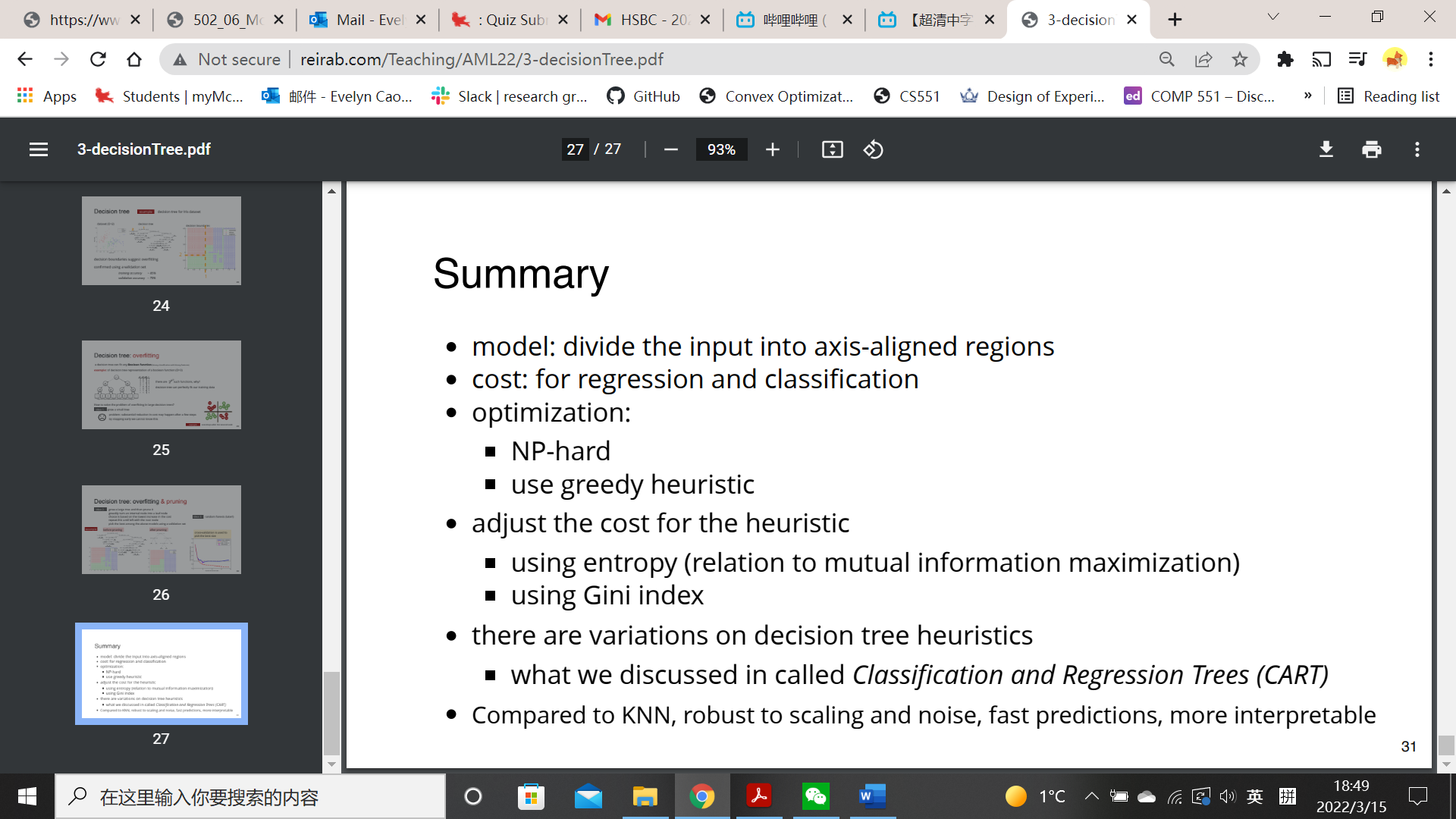


Probability of each class:

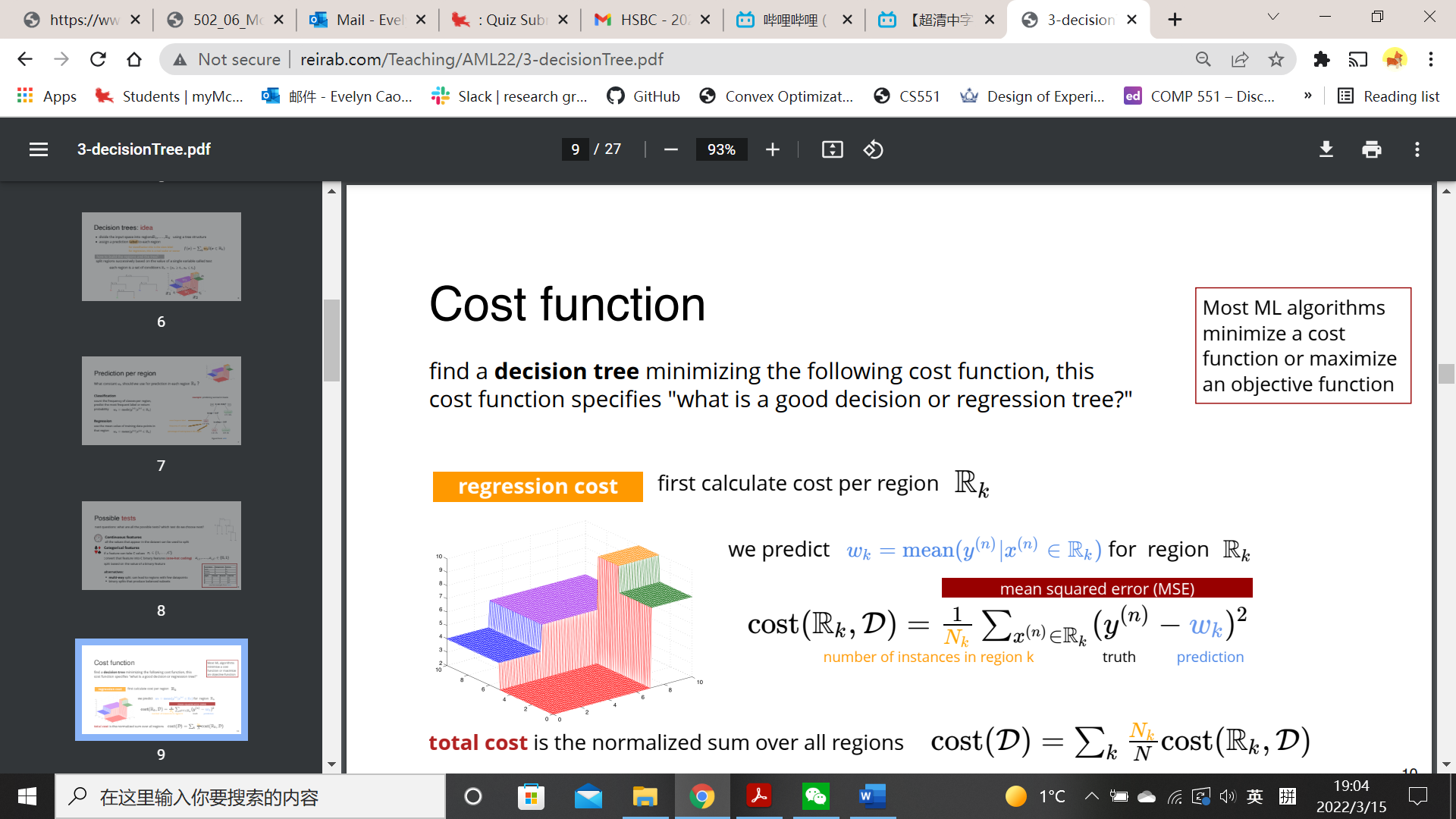
Small K overfits, large K underfits.

Computational complexity for a single test query is O(ND+NK)

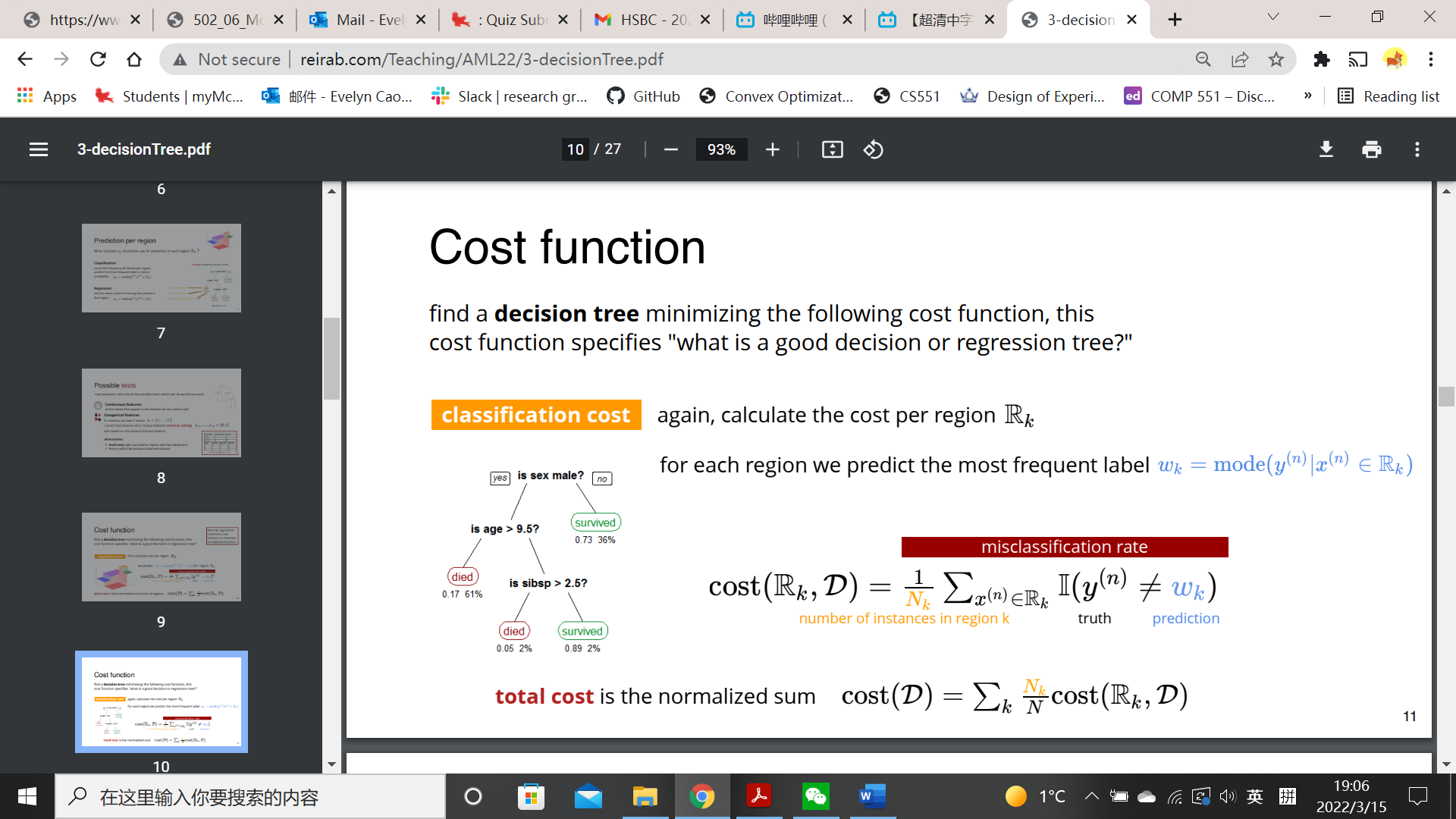
Decision Tree



One-hot coding: convert categorical feature into C binary features, split based on the value of a binary feature.

Cost function: where

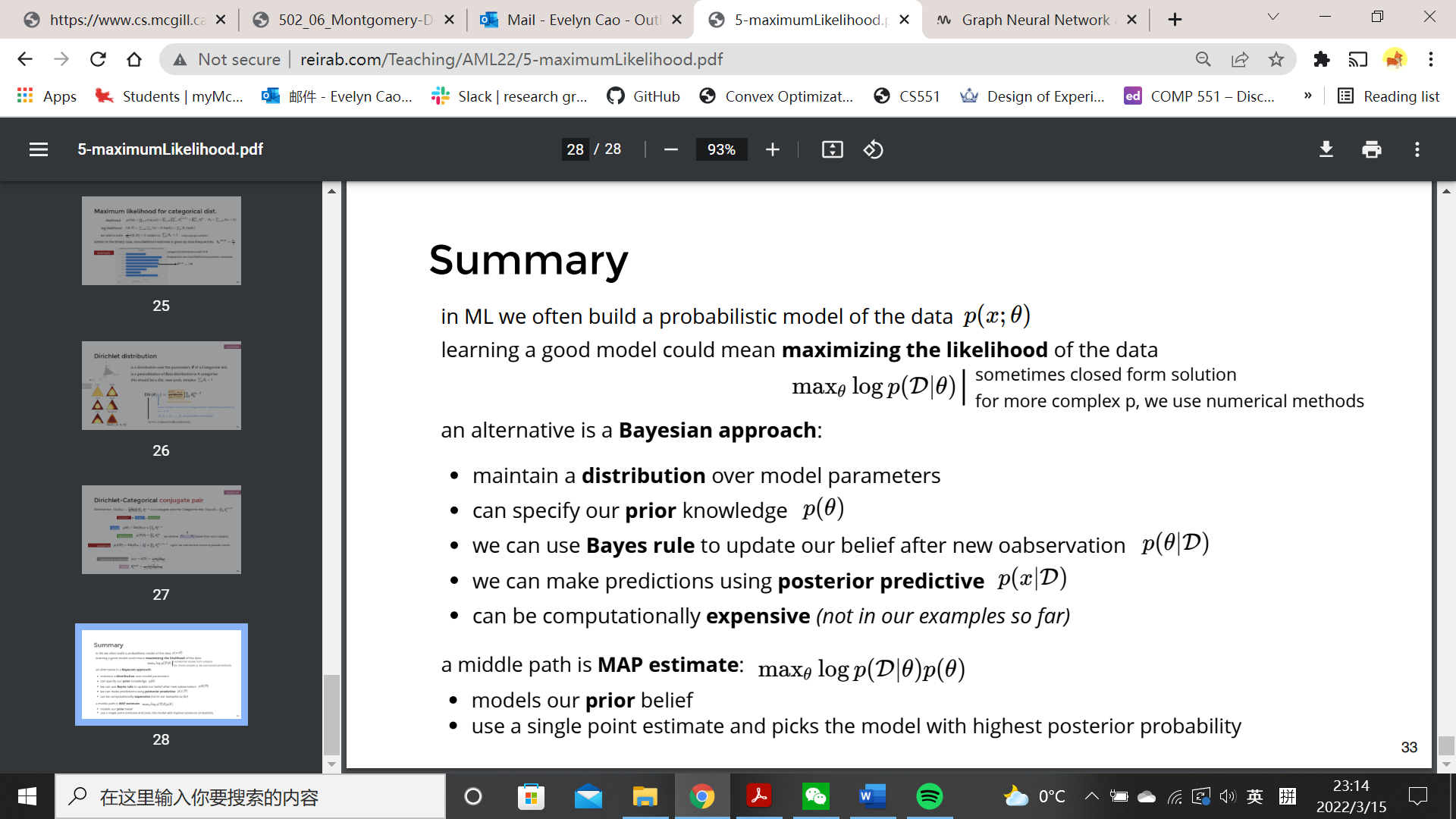
regression cost is

classification cost is

To avoid overfitting, build DT with at most K tests. K tests = K internal nodes = K+1 leaves

Entropy cost:

Naïve Bayes



Beta-Bernoulli:

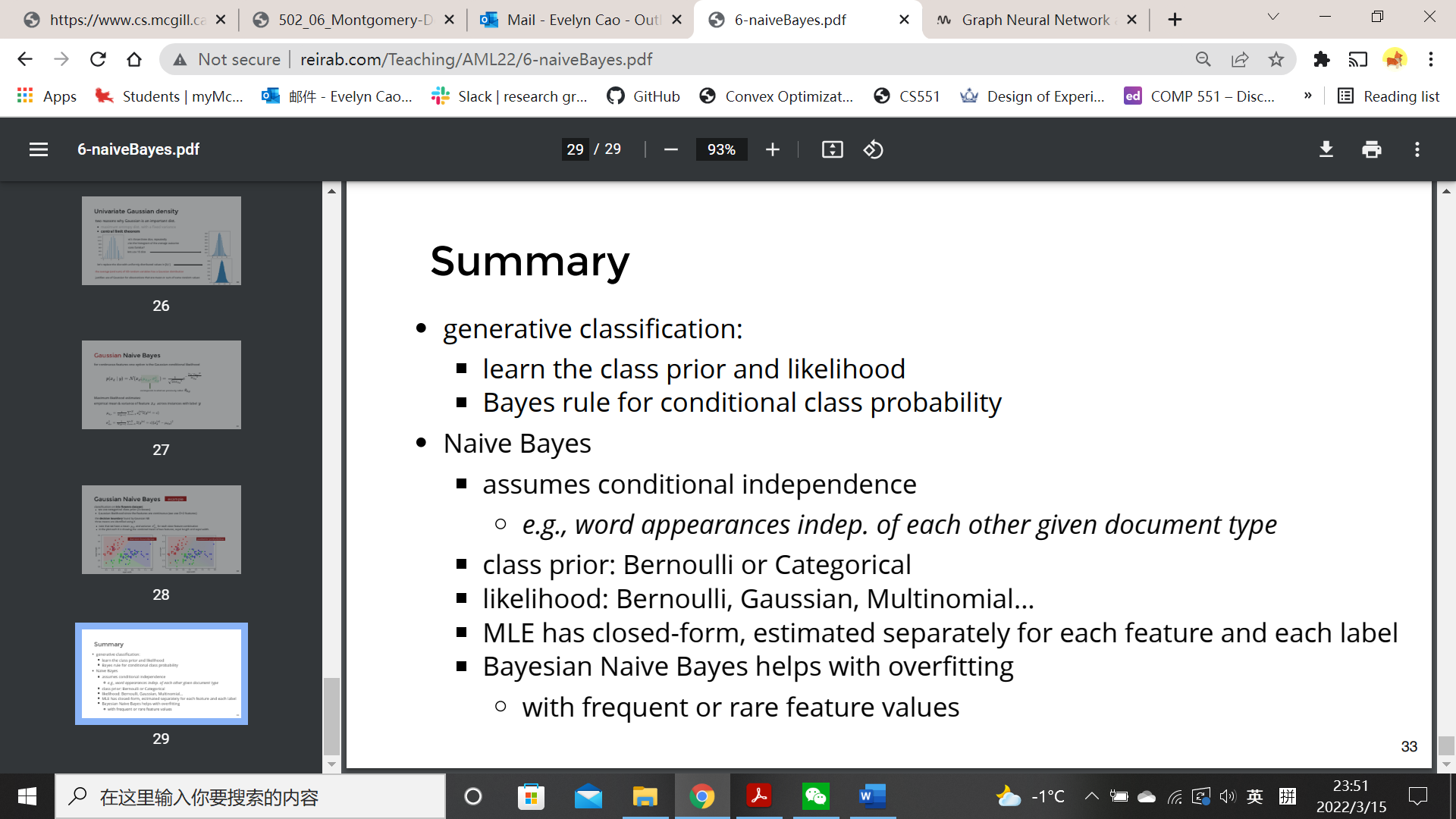
Prior

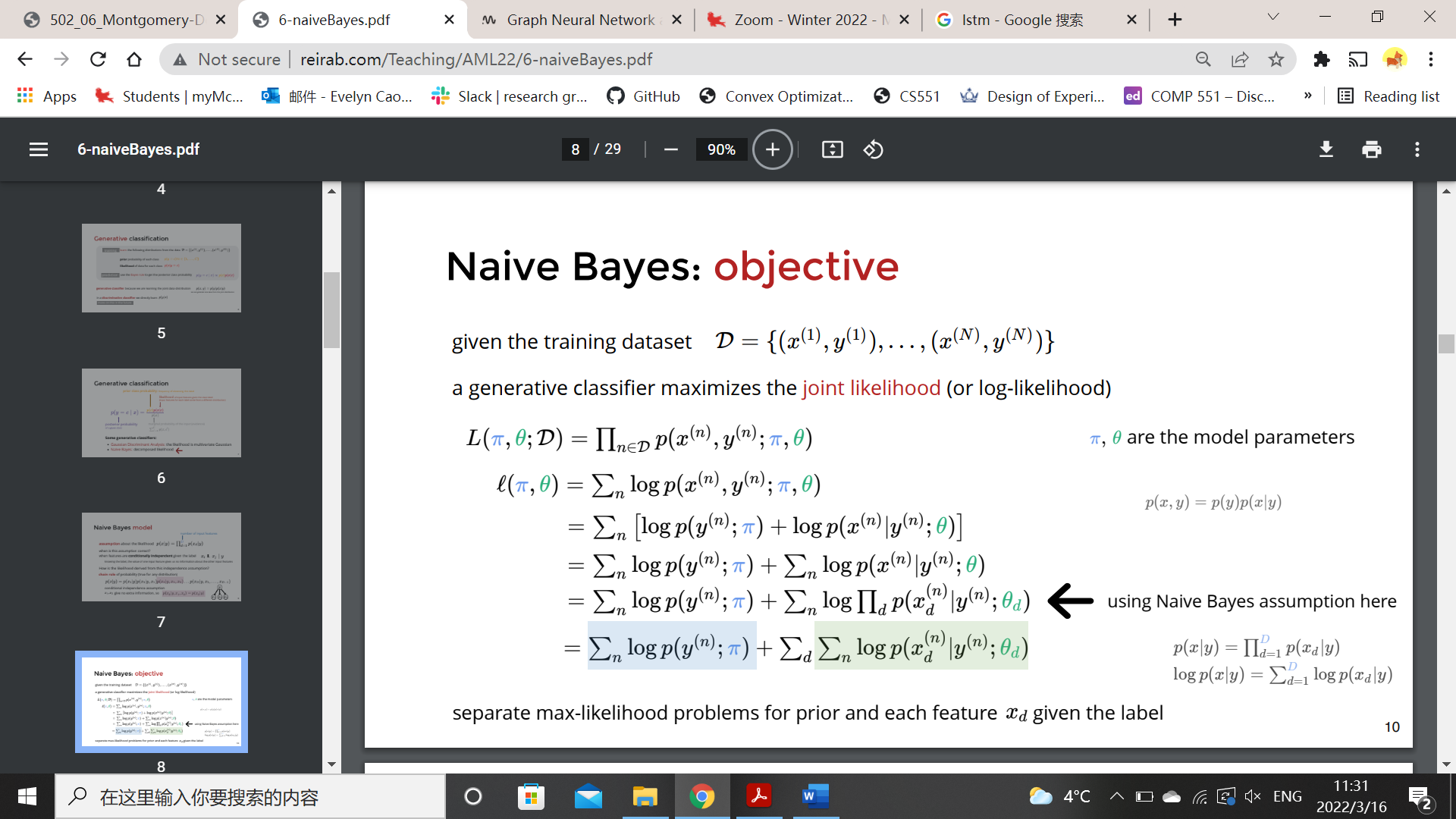
Posterior

Posterior predictive

MAP

Categorical distribution:





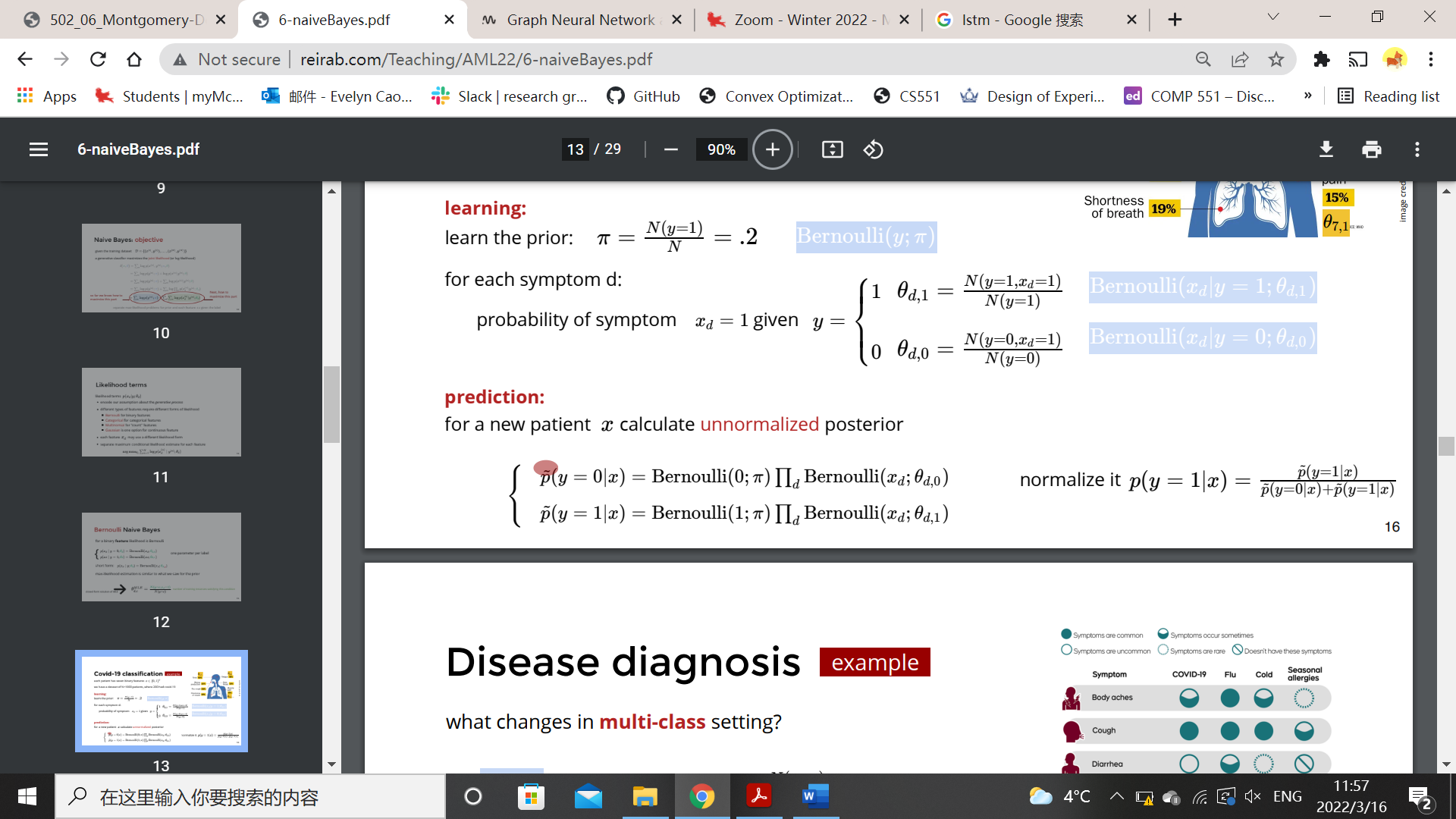
NB learns the joint distribution

is prior, we learn it as the class frequencies N(y=c)/N in the training data.

is likelihood, each feature may use a different likelihood form.

* Bernoulli NB: Ber likelihood for binary features

Point estimate by MLE:

Binary classification 🡪 1+2D parameters in model

Multi-class 🡪 C+CD parameters

* Bayesian NB: maintain distributions
* Multinomial NB for ‘count’ features
* Gaussian NB for continuous features

Linear Regression

Cost function: use L2 loss,

* Closed form solution , O(ND^2+D^3)

Probabilistic interpretation:

Learn and maximize the conditional likelihood.

When using square loss, we are assuming Gaussian noise.

Nonlinear basis

Logistic Regression

The linear decision boundary is

Cost function: use cross-entropy loss,

Probabilistic interpretation:

Interpret as class probability

Learn

Log-likelihood is the negative cost function 🡪 min J = max l

Softmax Regression

, where

Use one-hot encoding for labels

Cost function:use cross-entropy loss,

Gradient: C\*D parameters

Learn

Gradient descent

Convex function:

Convex if second derivative is positive everywhere

Sum of convex

Maximum of convex

g(f(x)) if f, g are convex and g is non-decreasing

Gradient for linear and logistic regression: is a D\*1 vector

in O(ND)

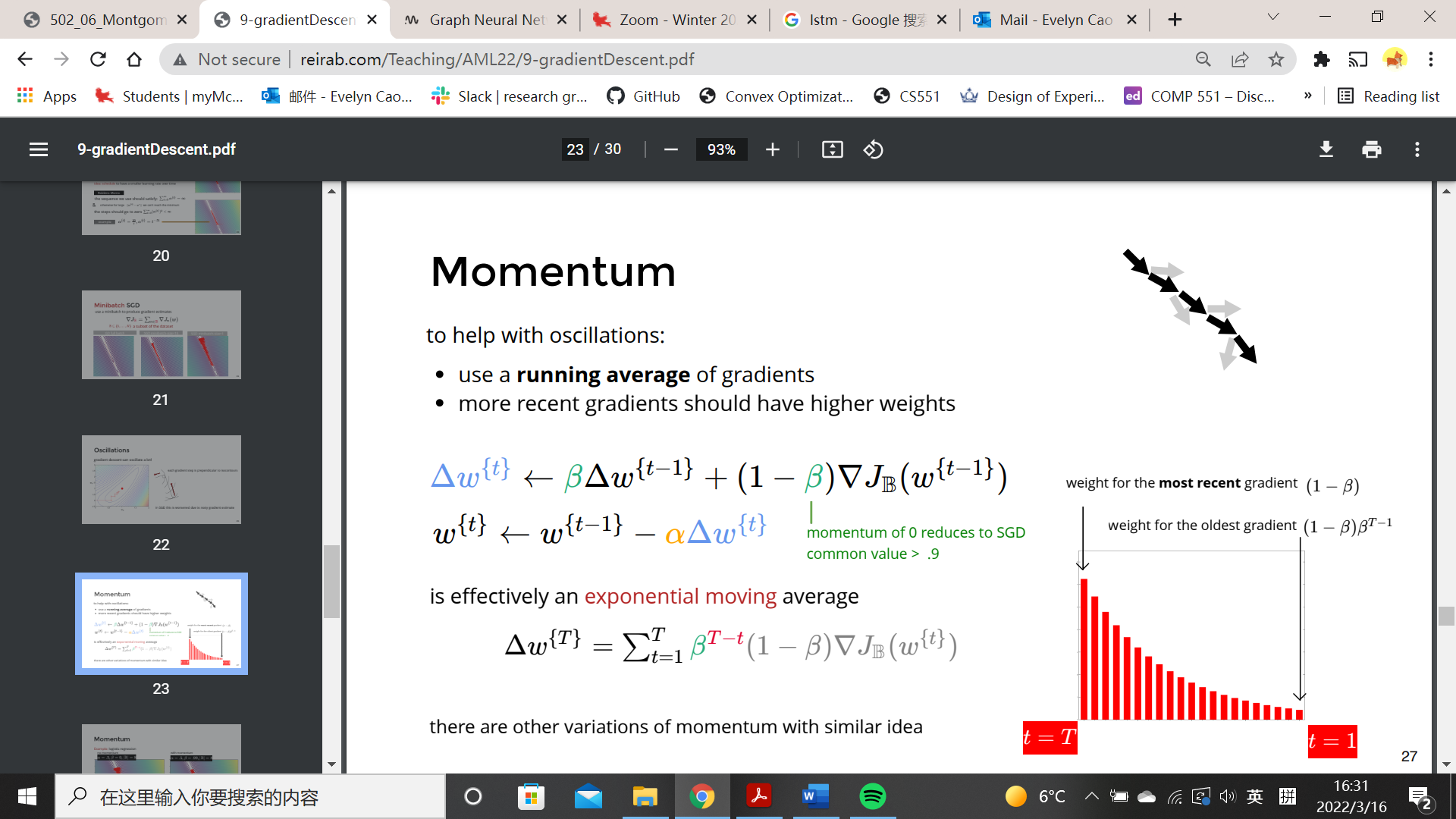
SGD:

For large dataset use minibatch for a noisy-fast estimate of gradient

* Robbins Monro condition 🡪 guarantee convergence

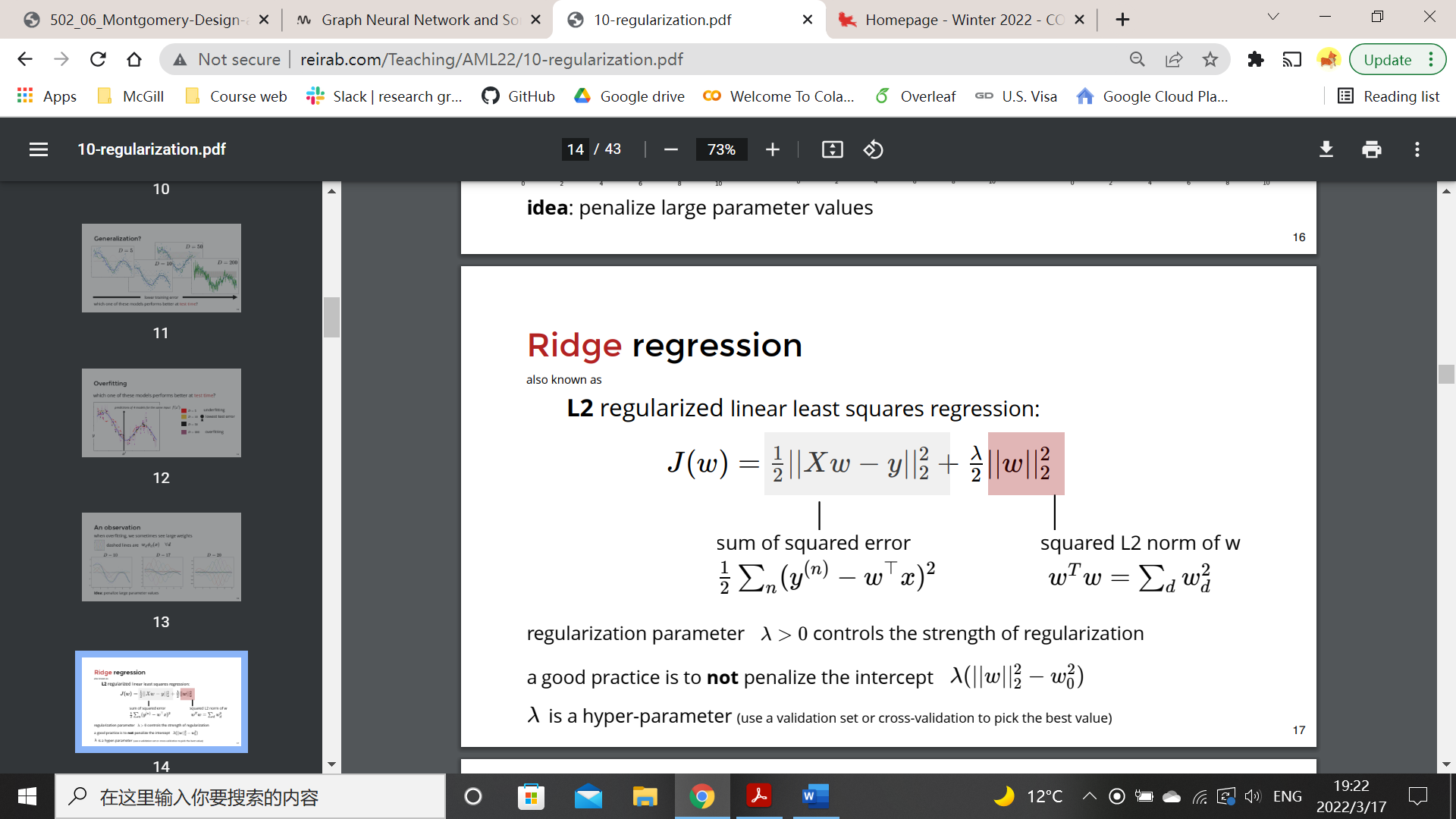
have a smaller learning rate over time

* Momentum 🡪 help with oscillations



Regularization

Ridge regression (L2 regularization)



weight decay when using gradient descent

🡪 unique

Probabilistic interpretation: Gaussian prior, smaller variance of the prior gives larger regularization

Lasso (L1 regularization):

Laplace prior

Produce sparse weights, useful for feature selection

Bias-variance tradeoff

Test error ~ bias(training error) + variance(complexity)

MLP

Perceptron:

If minimize

Use SGD, , learning rate is 1

Guaranteed to converge if linearly separable

Adaptive base: learn the parameters of the bases

Two-layer model:

g depends on the task regression/classification

activation functions: identity, logistic, hyperbolic tangent, ReLU

Regularization strategies

* Data augmentation: increase the size of dataset by adding reasonable transformations that change the label in predictive ways.
* Noise robustness: flat minimum generalizes better
* Early stopping: bound the region of parameter-space
* Dropout: Training: randomly dropout each unit with probability p

Test: average prediction of several passes; scale the weights by p

Gradient computation

