Introduction to Data Science Topic-7

- Instructor: Professor Henry Horng-Shing Lu,
  Institute of Statistics, National Yang Ming Chiao Tung University, Taiwan
  Email: henryhslu@nycu.edu.tw
- WWW: http://misg.stat.nctu.edu.tw/hslu/course/DataScience.htm
- Classroom: ED B27 (新竹市大學路1001號工程四館B27教室)
- References:
  M. A. Pathak, Beginning Data Science with R, 2014, Springer-Verlag.
  K.-T. Tsai, Machine Learning for Knowledge Discovery with R: Methodologies for Modeling, Inference, and Prediction, 2021, Chapman and Hall/CRC.
- Evaluation: Homework: 70%, Term Project: 30%
- Office hours: By appointment

Course Outline

10 Topics and 10 Homeworks:

- **Introduction of Data Science**
- **Introduction of R and Python**
- **Cleaning Data into R and Python**
- **Data Visualization**
- **Exploratory Data Analysis**
- **Regression (Supervised Learning)**
- **Classification (Supervised Learning)**
- **Text Mining**
- **Clustering (Unsupervised Learning)**
- **Neural Network and Deep Learning**

Classification with R

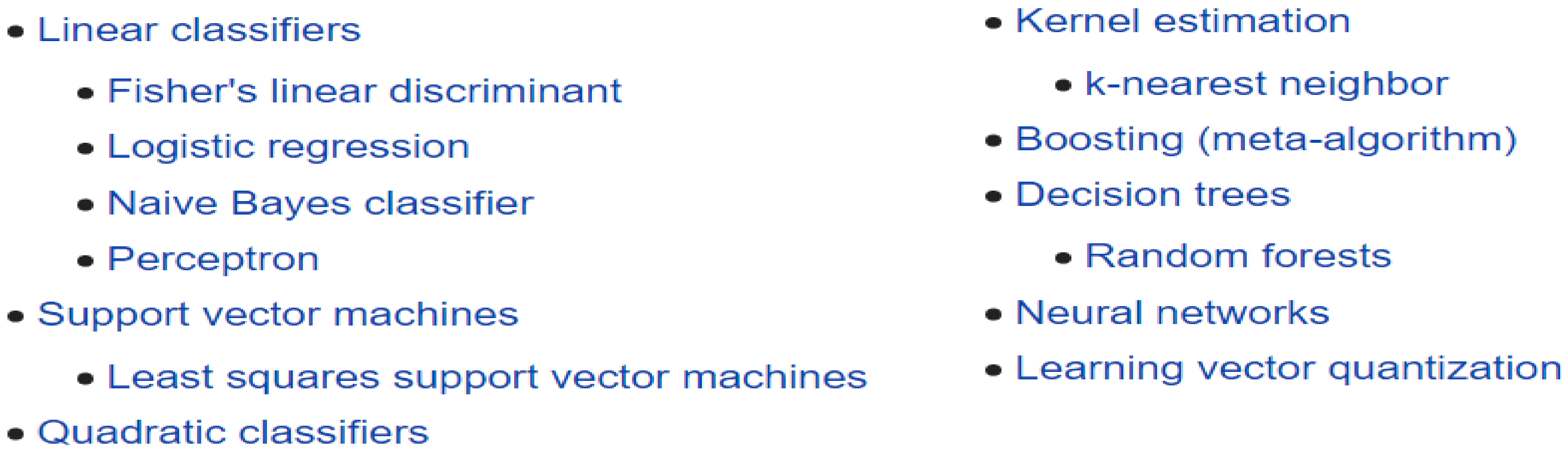
References:
Ch. 7, M. A. Pathak, Beginning Data Science with R, 2014, Springer-Verlag.
https://www.kaggle.com/code/royshih23/topic7-classification-in-python/notebook

Classification

- Why not linear regression?
- Logistic Regression
- Naïve Bayes
- SVM
- Discriminant Analysis
    - *Bayesian Theorem
    - *LDA
    - *QDA

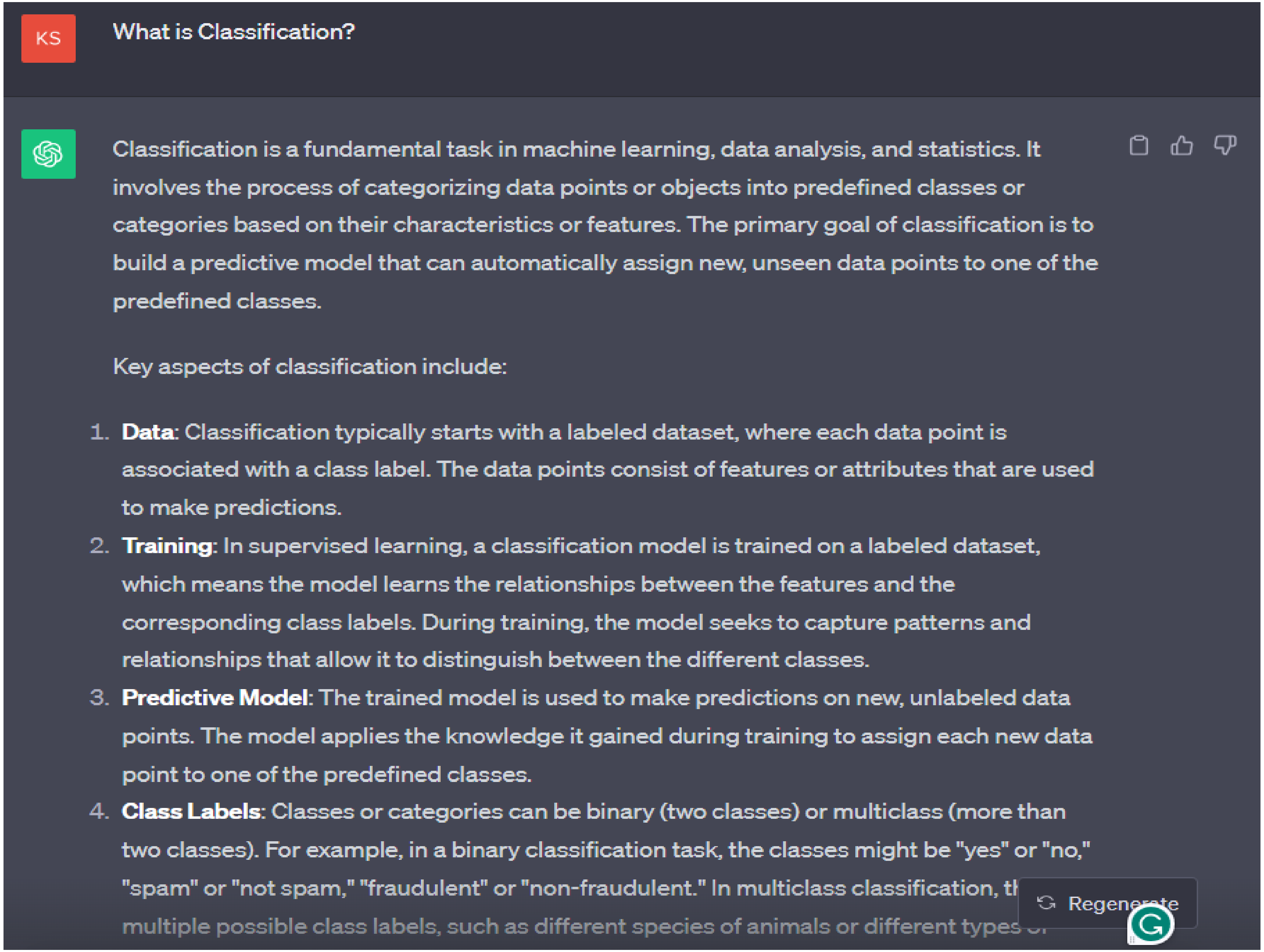
Classification

Classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.



- Linear classifiers
  - Fisher's linear discriminant
  - Logistic regression
  - Naive Bayes classifier
  - Perceptron
- Support vector machines
  - Least squares support vector machines
- Quadratic classifiers
- Kernel estimation
  - k-nearest neighbor
- Boosting (meta-algorithm)
- Decision trees
  - Random forests
- Neural networks
- Learning vector quantization

https://en.wikipedia.org/wiki/Statistical_classification

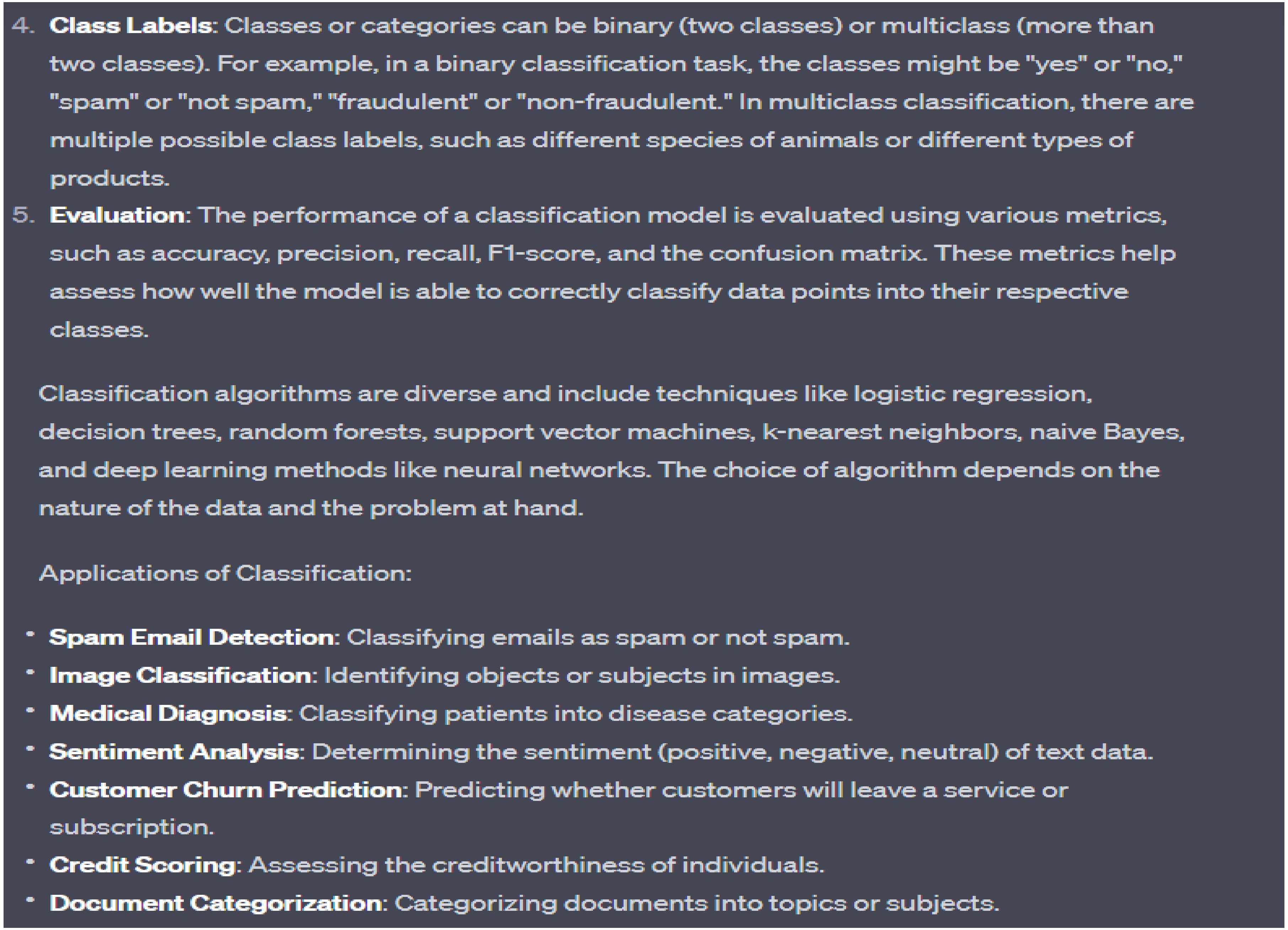


What is Classification?

Classification is a fundamental task in machine learning, data analysis, and statistics. It involves the process of categorizing data points or objects into predefined classes or categories based on their characteristics or features. The primary goal of classification is to build a predictive model that can automatically assign new, unseen data points to one of the predefined classes.

Key aspects of classification include:

1. **Data**: Classification typically starts with a labeled dataset, where each data point is associated with a class label. The data points consist of features or attributes that are used to make predictions.
2. **Training**: In supervised learning, a classification model is trained on a labeled dataset, which means the model learns the relationships between the features and the corresponding class labels. During training, the model seeks to capture patterns and relationships that allow it to distinguish between the different classes.
3. **Predictive Model**: The trained model is used to make predictions on new, unlabeled data points. The model applies the knowledge it gained during training to assign each new data point to one of the predefined classes.
4. **Class Labels**: Classes or categories can be binary (two classes) or multiclass (more than two classes). For example, in a binary classification task, the classes might be "yes" or "no," "spam" or "not spam," "fraudulent" or "non-fraudulent." In multiclass classification, th
multiple possible class labels, such as different species of animals or different types o



4. **Class Labels**: Classes or categories can be binary (two classes) or multiclass (more than two classes). For example, in a binary classification task, the classes might be "yes" or "no," "spam" or "not spam," "fraudulent" or "non-fraudulent." In multiclass classification, there are multiple possible class labels, such as different species of animals or different types of products.
5. **Evaluation**: The performance of a classification model is evaluated using various metrics, such as accuracy, precision, recall, F1-score, and the confusion matrix. These metrics help assess how well the model is able to correctly classify data points into their respective classes.

Classification algorithms are diverse and include techniques like logistic regression, decision trees, random forests, support vector machines, k-nearest neighbors, naive Bayes, and deep learning methods like neural networks. The choice of algorithm depends on the nature of the data and the problem at hand.

Applications of Classification:

- **Spam Email Detection**: Classifying emails as spam or not spam.
- **Image Classification**: Identifying objects or subjects in images.
- **Medical Diagnosis**: Classifying patients into disease categories.
- **Sentiment Analysis**: Determining the sentiment (positive, negative, neutral) of text data.
- **Customer Churn Prediction**: Predicting whether customers will leave a service or subscription.
- **Credit Scoring**: Assessing the creditworthiness of individuals.
- **Document Categorization**: Categorizing documents into topics or subjects.

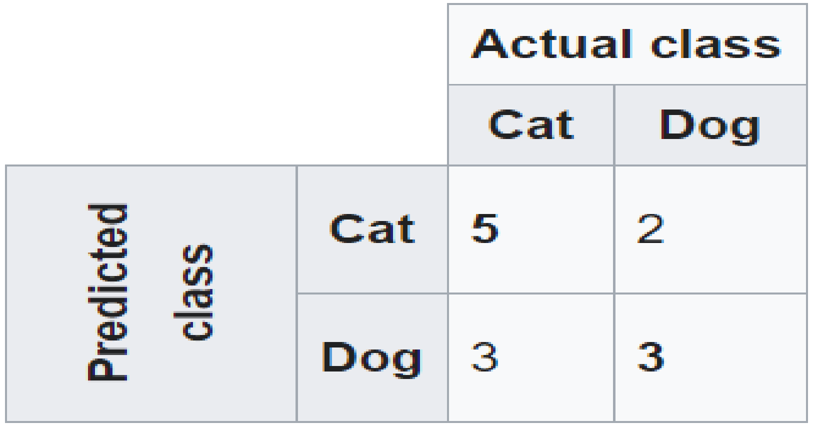
Confusion matrix

- In the field statistical classification, a confusion matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one.

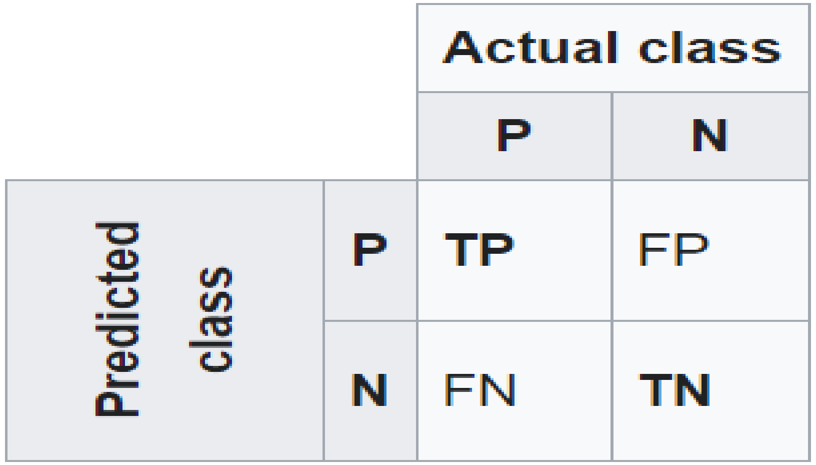
- Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).

https://en.wikipedia.org/wiki/Confusion_matrix

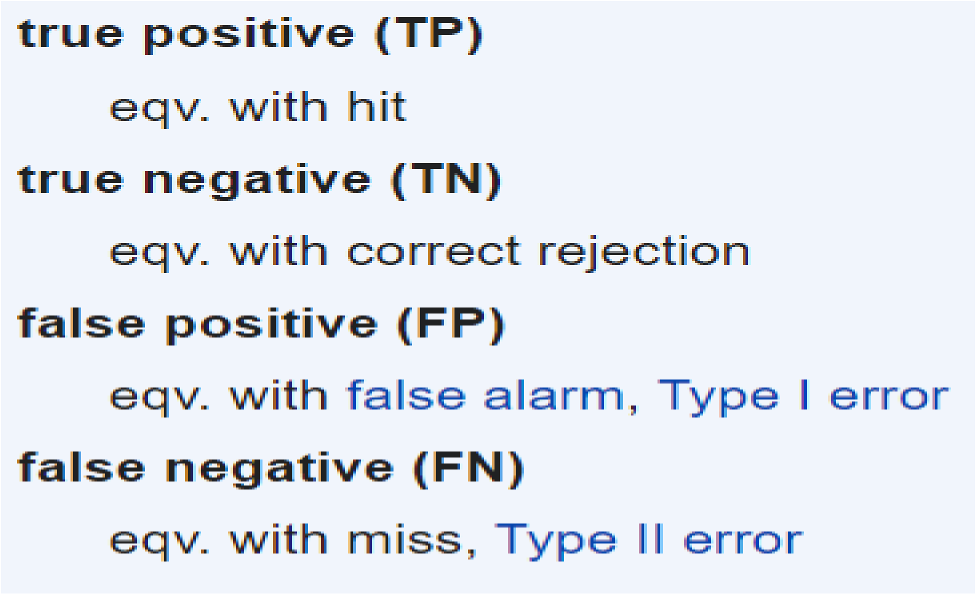
Confusion matrix



		Actual class	
		Cat	Dog
Predicted class	Cat	5	2
	Dog	3	3



		Actual class	
		P	N
Predicted class	P	TP	FP
	N	FN	TN



true positive (TP)

eqv. with hit

true negative (TN)

eqv. with correct rejection

false positive (FP)

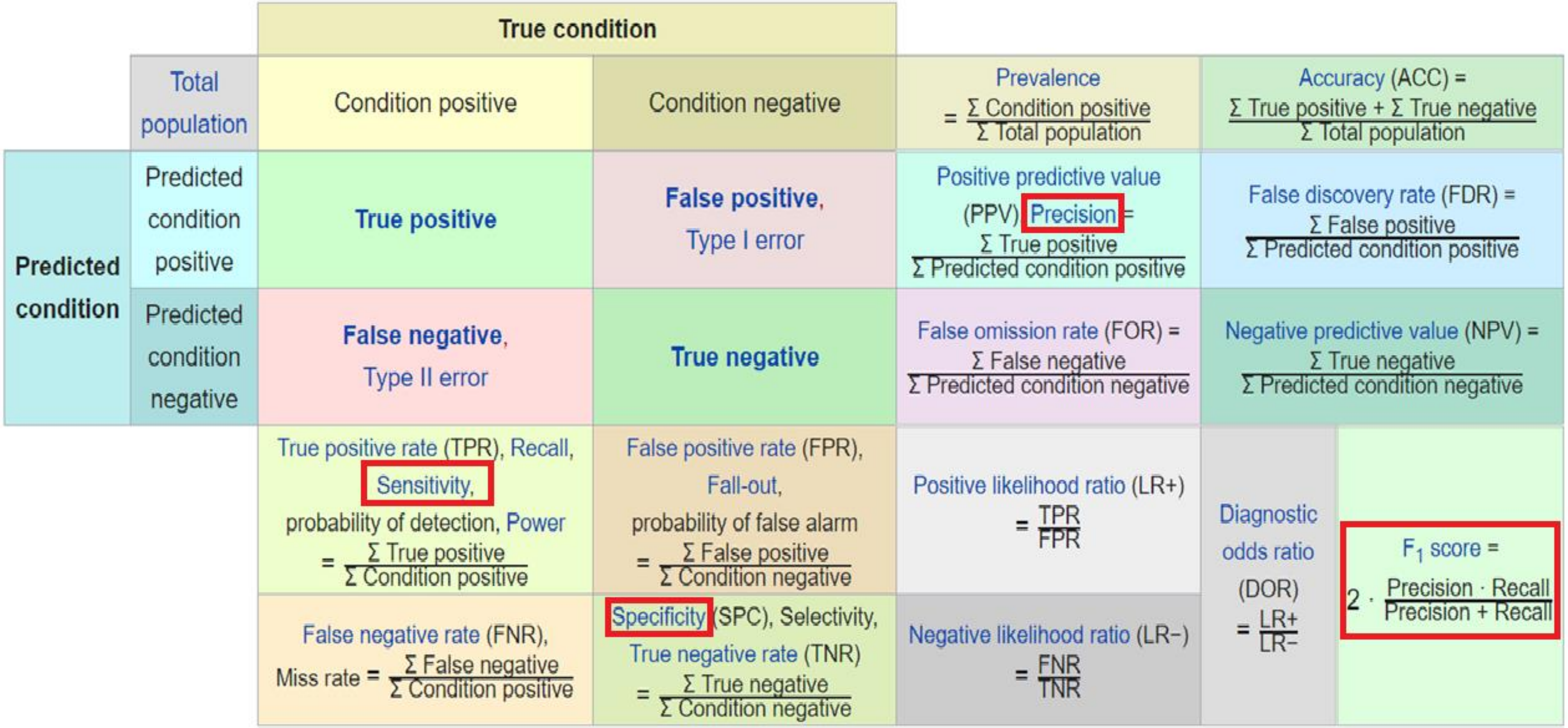
eqv. with false alarm, Type I error

false negative (FN)

eqv. with miss, Type II error

https://en.wikipedia.org/wiki/Confusion_matrix

Confusion matrix



<table>
<tr><td></td><td></td><td colspan="2"><b>True condition</b></td><td></td><td colspan="2"></td></tr>
<tr><td></td><td>Total population</td><td>Condition positive</td><td>Condition negative</td><td>Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$</td><td colspan="2">Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$</td></tr>
<tr><td rowspan="2"><b>Predicted condition</b></td><td>Predicted condition positive</td><td><b>True positive</b></td><td><b>False positive</b>,
Type I error</td><td>Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$</td><td colspan="2">False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$</td></tr>
<tr><td>Predicted condition negative</td><td><b>False negative</b>,
Type II error</td><td><b>True negative</b></td><td>False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$</td><td colspan="2">Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$</td></tr>
<tr><td></td><td></td><td>True positive rate (TPR), Recall, Sensitivity, probability of detection, Power $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$</td><td>False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$</td><td>Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$</td><td rowspan="2">Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$</td><td rowspan="2">$F_1$ score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$</td></tr>
<tr><td></td><td></td><td>False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$</td><td>Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$</td><td>Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$</td></tr>
</table>

https://en.wikipedia.org/wiki/Confusion_matrix

Receiver operating characteristic

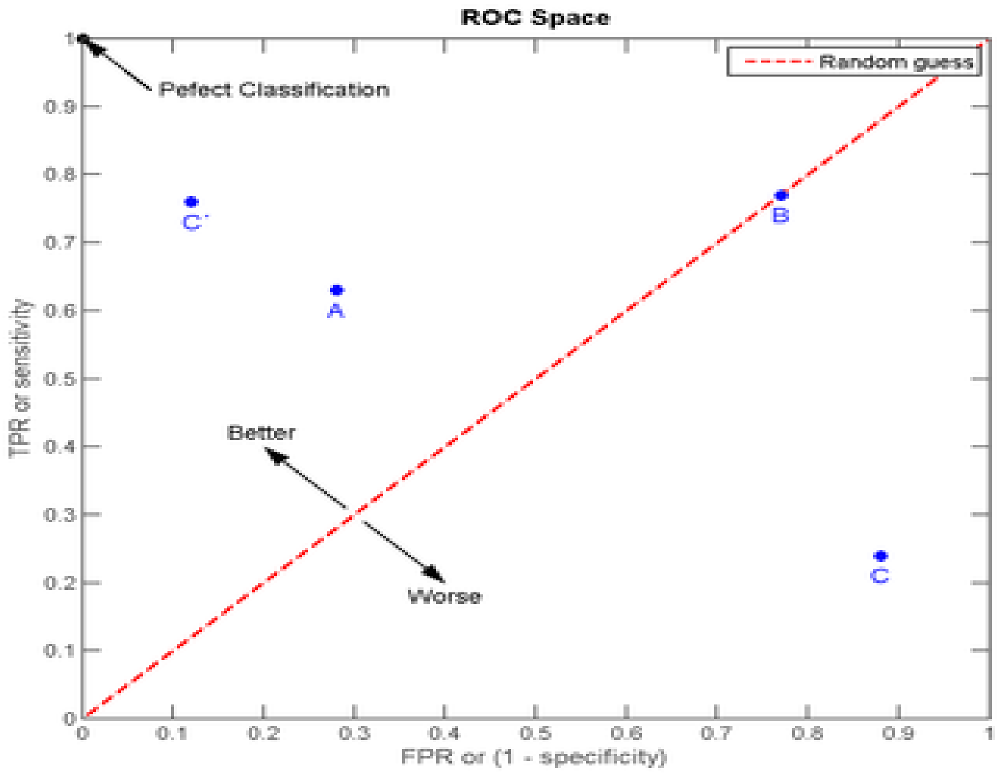
- A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

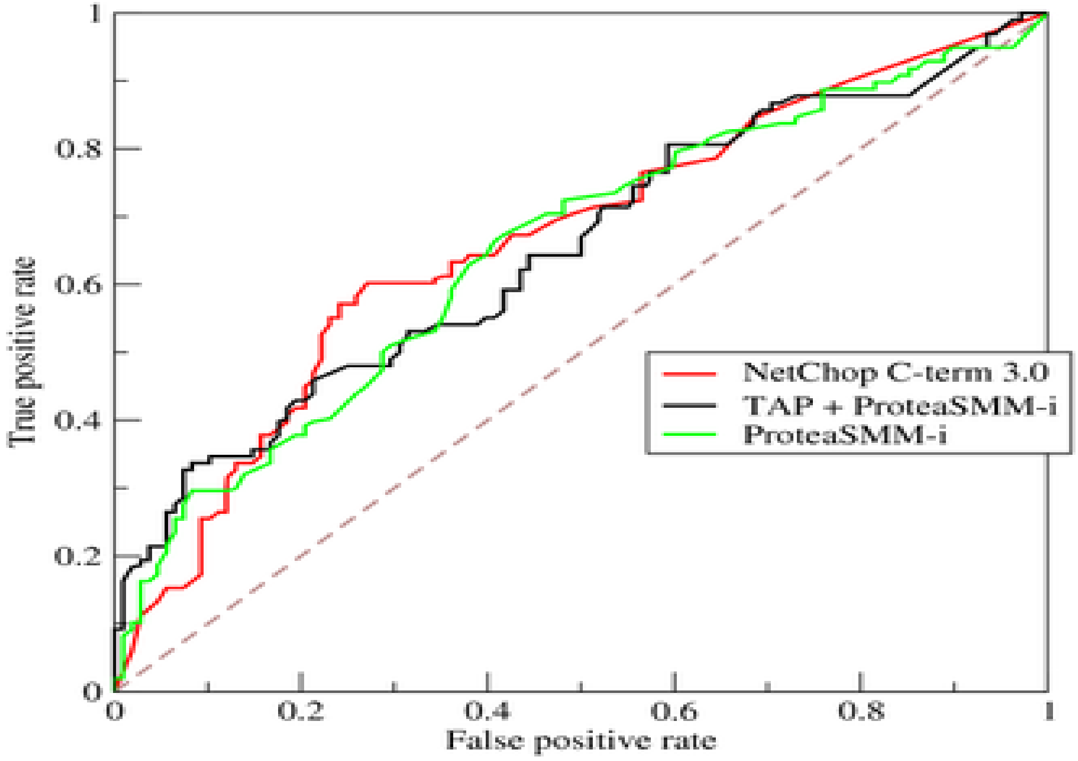
- The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

- The ROC curve is thus the sensitivity or recall as a function of fall-out.

https://en.wikipedia.org/wiki/Receiver_operating_characteristic

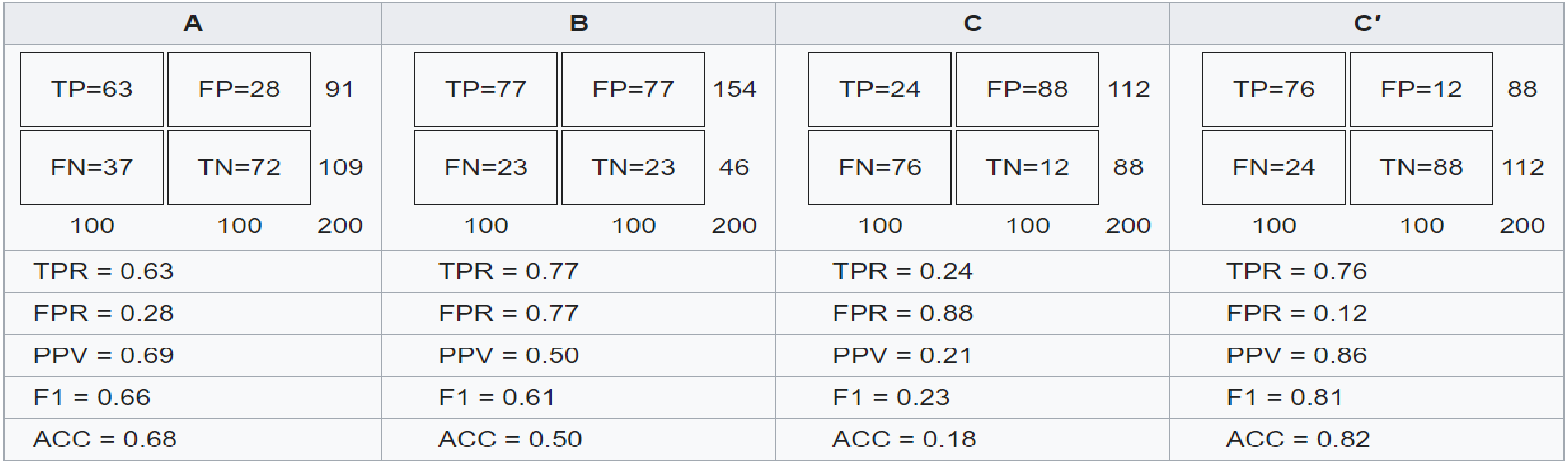
Receiver operating characteristic





https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Receiver operating characteristic



A	B	C	C′

TP=63 FP=28 91
FN=37 TN=72 109
100 100 200
	TP=77 FP=77 154
	FN=23 TN=23 46
	100 100 200
		TP=24 FP=88 112
		FN=76 TN=12 88
		100 100 200
			TP=76 FP=12 88
			FN=24 TN=88 112
			100 100 200

TPR = 0.63	TPR = 0.77	TPR = 0.24	TPR = 0.76
FPR = 0.28	FPR = 0.77	FPR = 0.88	FPR = 0.12
PPV = 0.69	PPV = 0.50	PPV = 0.21	PPV = 0.86
F1 = 0.66	F1 = 0.61	F1 = 0.23	F1 = 0.81
ACC = 0.68	ACC = 0.50	ACC = 0.18	ACC = 0.82

https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Receiver operating characteristic

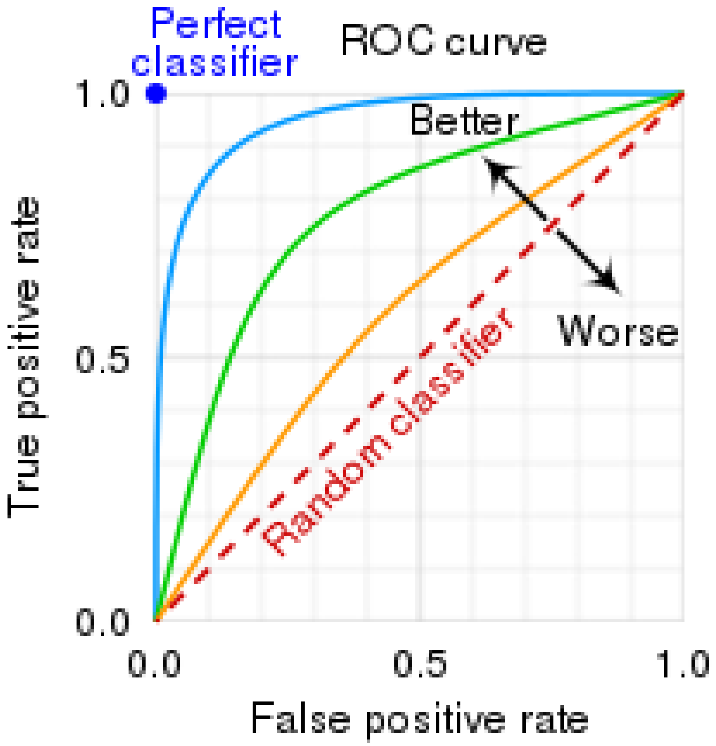
- The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% sensitivity and 100% specificity.

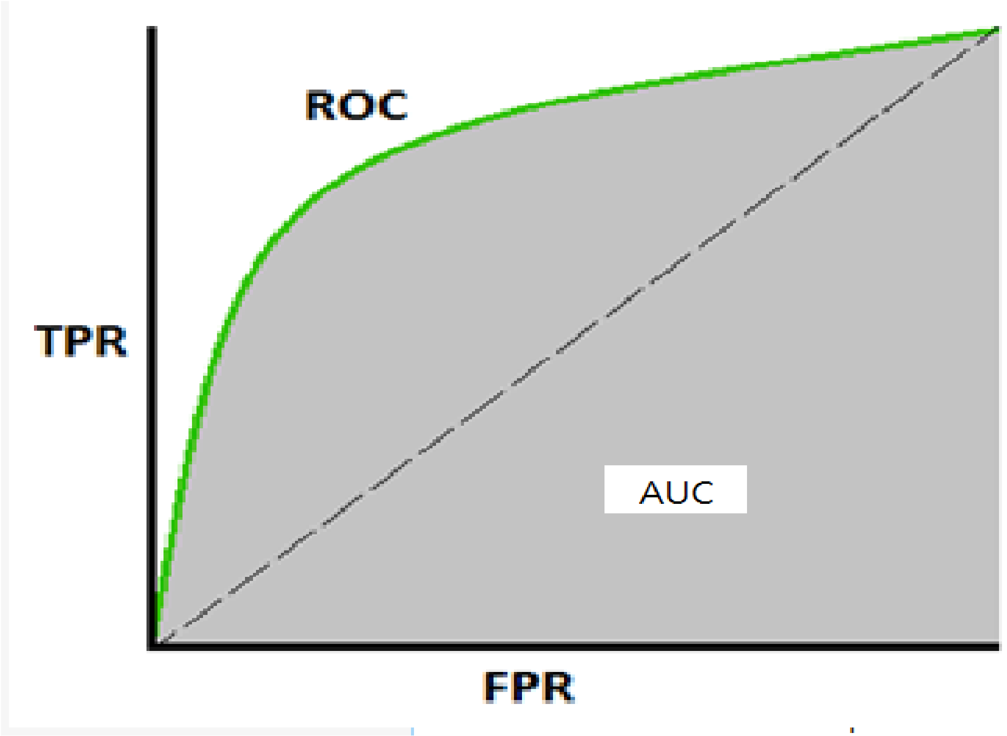
- A random guess would give a point along a diagonal line (the so-called *line of no-discrimination*) from the left bottom to the top right corners.

https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Receiver operating characteristic

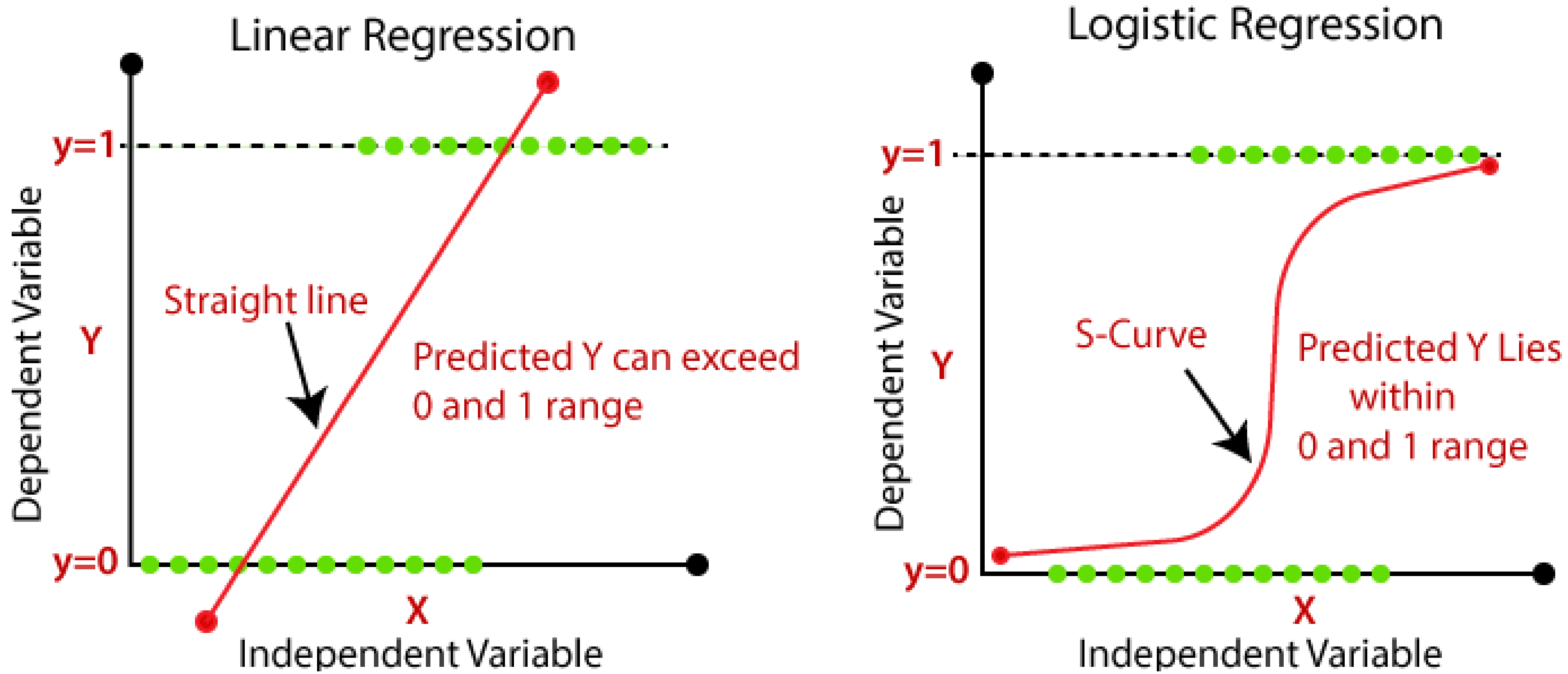
When using normalized units, the area under the curve (often referred to as simply the AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.





https://a7744hsc.github.io/machine/learning/2019/07/12/AUC-ROC.html

Why not linear regression?



https://www.javatpoint.com/linear-regression-vs-logistic-regression-in-machine-learning

Linear Regression vs. Logistic Regression

- **Linear Regression**

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots. + \beta_p x_p = \boldsymbol{\beta}^{\boldsymbol{T}} \boldsymbol{X}$$

- **Logistic Regression**

$$P(y = 1 \,|\, X = x) = \frac{e^y}{1+e^y} = \frac{e^{\boldsymbol{\beta}^{\boldsymbol{T}} \mathbf{X}}}{1+e^{\boldsymbol{\beta}^{\boldsymbol{T}} \mathbf{X}}}$$

This is classification rather than regression, so we use MLE.

Logistic Regression

If two classes Logistic Regression, it is Bernoulli distribution.

$$P(y=1|X=\mathbf{x}_i)=p, \quad p(y=0|X=\mathbf{x}_i)=1-p,$$

x_i

$$f(\mathbf{x}_i)=p^y(1-p)^{1-y}= \begin{cases} p, y=1 \\ 1-p, y=0 \end{cases}$$ where p is a function of $\mathbf{x}_i \cdot \beta$

$$L(\beta)=\prod_{i=1}^{n} f(\mathbf{x}_i)=\prod_{i=1}^{n}[p^y(1-p)^{1-y}]$$

$$log(L(\beta))=\Sigma_{i=1}^{n}[ylog(p)+(1-y)log(1-p)]=\Sigma_{i=1}^{n}[ylog(\frac{e^{\beta^T\mathbf{x}_i}}{1+e^{\beta^T\mathbf{x}_i}})+(1-y)log(\frac{1}{1+e^{\beta^T\mathbf{x}_i}})]$$

$$=\Sigma_{i=1}^{n}[y[\beta^T\mathbf{x}_i \ -log(1+e^{\beta^T\mathbf{x}_i})]+(1-y)[-log(1+e^{\beta^T\mathbf{x}_i})]]=\Sigma_{i=1}^{n}[y\beta^T\mathbf{x}_i \ -log(1+e^{\beta^T\mathbf{x}_i})]$$

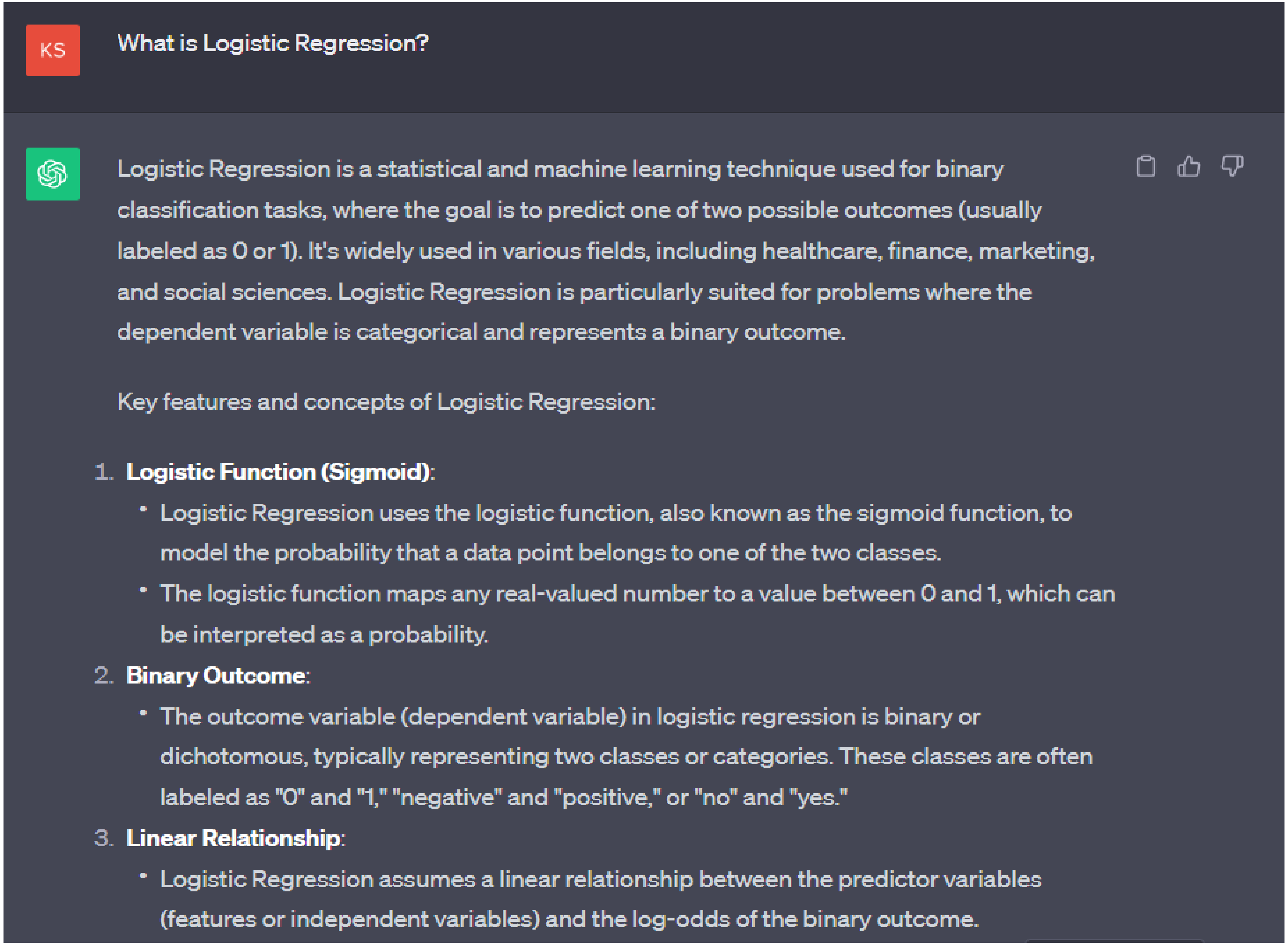
$$\frac{\partial logL(\beta)}{\partial\beta}=\Sigma_{i=1}^{n} x_i \, [y-\frac{e^{\beta^T\mathbf{x}_i}}{1+e^{\beta^T\mathbf{x}_i}}]$$

$$\beta^{(t+1)}=\beta^{(t)}+\alpha\frac{\partial logL(\beta)}{\partial\beta}=\beta^{(t)}+\alpha\Sigma_{i=1}^{n}[\, x_i \,(y_i-\frac{e^{\beta^T\mathbf{x}_i}}{1+e^{\beta^T\mathbf{x}_i}})]$$

If multi-classes Logistic Regreesion $$P(Y=k|X=\mathbf{x}_i)=\frac{e^{\beta_k^T\mathbf{x}_i}}{\Sigma_{k=1}^{K}e^{\beta_k^T\mathbf{x}_i}}$$

To see which probability of class is the largest, that is that class. The way of solving beta is as same as two classes Logistic Regression.

https://en.wikipedia.org/wiki/Logistic_regression

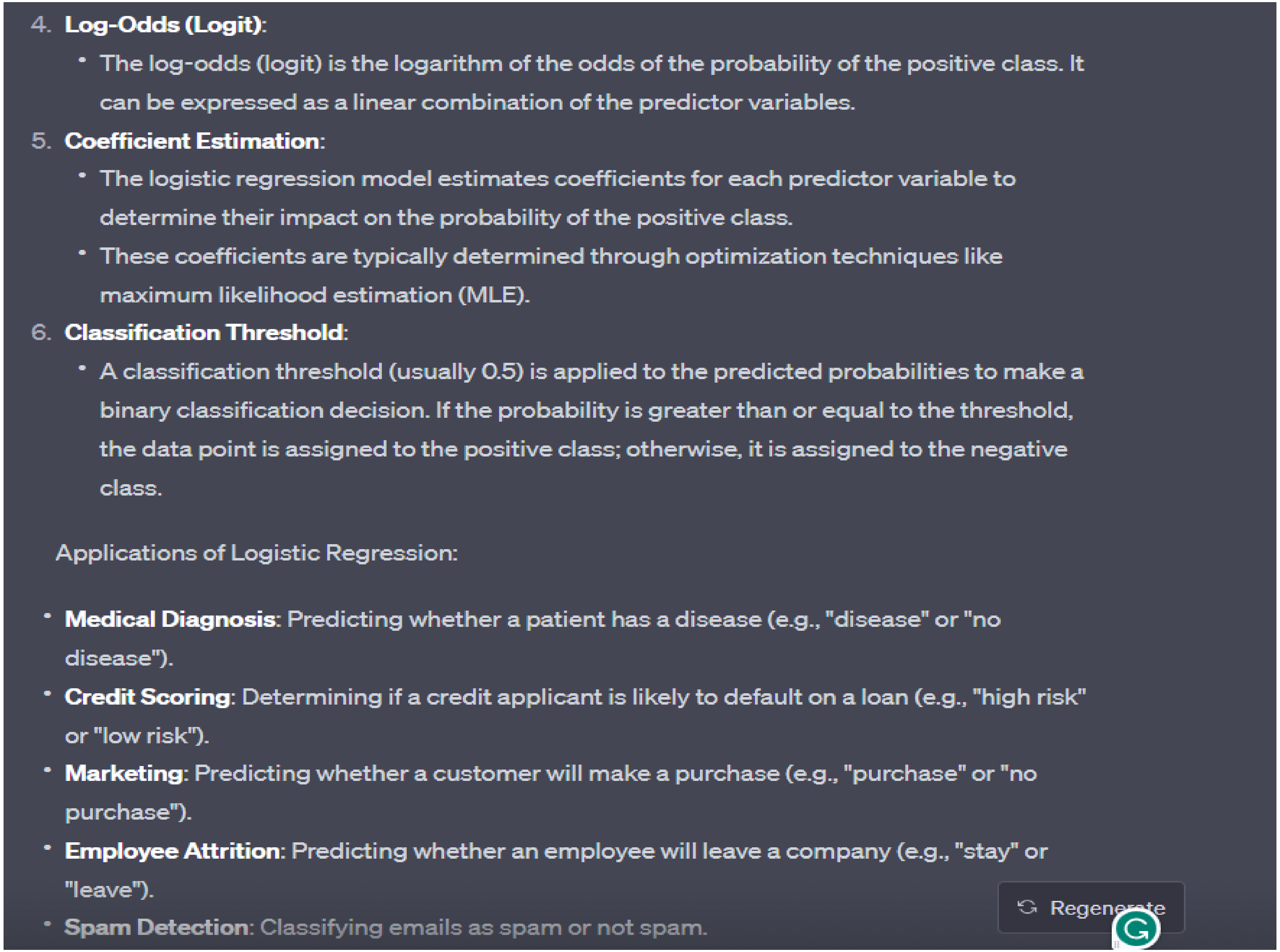


What is Logistic Regression?

Logistic Regression is a statistical and machine learning technique used for binary classification tasks, where the goal is to predict one of two possible outcomes (usually labeled as 0 or 1). It's widely used in various fields, including healthcare, finance, marketing, and social sciences. Logistic Regression is particularly suited for problems where the dependent variable is categorical and represents a binary outcome.

Key features and concepts of Logistic Regression:

1. **Logistic Function (Sigmoid)**:
   - Logistic Regression uses the logistic function, also known as the sigmoid function, to model the probability that a data point belongs to one of the two classes.
   - The logistic function maps any real-valued number to a value between 0 and 1, which can be interpreted as a probability.
2. **Binary Outcome**:
   - The outcome variable (dependent variable) in logistic regression is binary or dichotomous, typically representing two classes or categories. These classes are often labeled as "0" and "1," "negative" and "positive," or "no" and "yes."
3. **Linear Relationship**:
   - Logistic Regression assumes a linear relationship between the predictor variables (features or independent variables) and the log-odds of the binary outcome.



4. **Log-Odds (Logit)**:
    - The log-odds (logit) is the logarithm of the odds of the probability of the positive class. It can be expressed as a linear combination of the predictor variables.
5. **Coefficient Estimation**:
    - The logistic regression model estimates coefficients for each predictor variable to determine their impact on the probability of the positive class.
    - These coefficients are typically determined through optimization techniques like maximum likelihood estimation (MLE).
6. **Classification Threshold**:
    - A classification threshold (usually 0.5) is applied to the predicted probabilities to make a binary classification decision. If the probability is greater than or equal to the threshold, the data point is assigned to the positive class; otherwise, it is assigned to the negative class.

Applications of Logistic Regression:

- **Medical Diagnosis**: Predicting whether a patient has a disease (e.g., "disease" or "no disease").
- **Credit Scoring**: Determining if a credit applicant is likely to default on a loan (e.g., "high risk" or "low risk").
- **Marketing**: Predicting whether a customer will make a purchase (e.g., "purchase" or "no purchase").
- **Employee Attrition**: Predicting whether an employee will leave a company (e.g., "stay" or "leave").
- **Spam Detection**: Classifying emails as spam or not spam.

Multinomial logistic regression

To arrive at the multinomial logit model, one can imagine, for *K* possible outcomes, running *K*-1 independent binary logistic regression models, in which one outcome is chosen as a "pivot" and then the other *K*-1 outcomes are separately regressed against the pivot outcome. This would proceed as follows, if outcome *K* (the last outcome) is chosen as the pivot:

$$
\begin{aligned}
\ln \frac{\Pr(Y_i = 1)}{\Pr(Y_i = K)} &= \boldsymbol\beta_1 \cdot \mathbf{X}_i \\
\ln \frac{\Pr(Y_i = 2)}{\Pr(Y_i = K)} &= \boldsymbol\beta_2 \cdot \mathbf{X}_i \\
&\cdots\cdots \\
\ln \frac{\Pr(Y_i = K-1)}{\Pr(Y_i = K)} &= \boldsymbol\beta_{K-1} \cdot \mathbf{X}_i
\end{aligned}
$$

$$\Longrightarrow$$

$$
\begin{aligned}
\Pr(Y_i = 1) &= \Pr(Y_i = K)e^{\boldsymbol\beta_1 \cdot \mathbf{X}_i} \\
\Pr(Y_i = 2) &= \Pr(Y_i = K)e^{\boldsymbol\beta_2 \cdot \mathbf{X}_i} \\
&\cdots\cdots \\
\Pr(Y_i = K-1) &= \Pr(Y_i = K)e^{\boldsymbol\beta_{K-1} \cdot \mathbf{X}_i}
\end{aligned}
$$

Using the fact that all *K* of the probabilities must sum to one, we find:

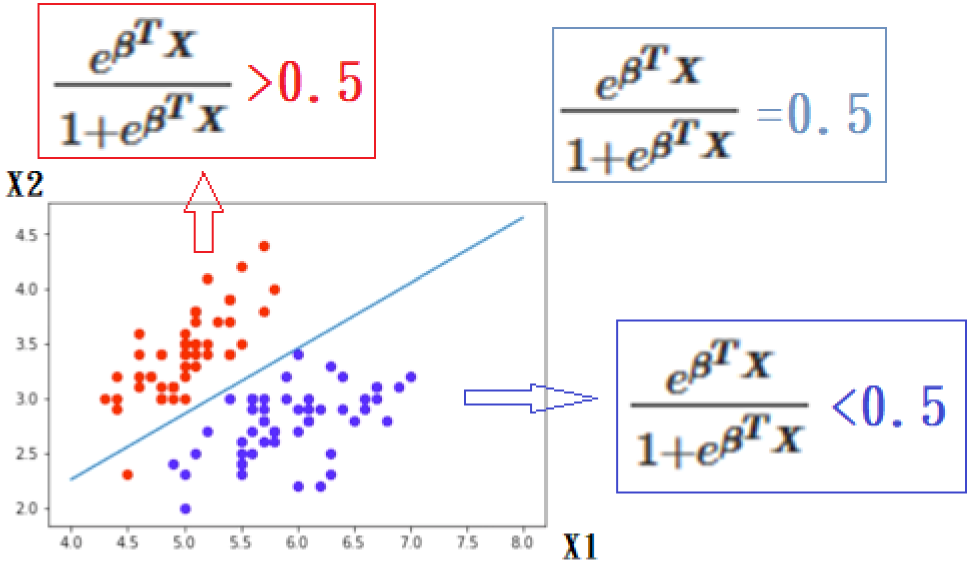
$$\Pr(Y_i = K) = 1 - \sum_{k=1}^{K-1} \Pr(Y_i = k) = 1 - \sum_{k=1}^{K-1} \Pr(Y_i = K)e^{\boldsymbol\beta_k \cdot \mathbf{X}_i} \Rightarrow \Pr(Y_i = K) = \frac{1}{1 + \sum_{k=1}^{K-1} e^{\boldsymbol\beta_k \cdot \mathbf{X}_i}}$$

We can use this to find the other probabilities:

$$
\begin{aligned}
\Pr(Y_i = 1) &= \frac{e^{\boldsymbol\beta_1 \cdot \mathbf{X}_i}}{1 + \sum_{k=1}^{K-1} e^{\boldsymbol\beta_k \cdot \mathbf{X}_i}} \\
\Pr(Y_i = 2) &= \frac{e^{\boldsymbol\beta_2 \cdot \mathbf{X}_i}}{1 + \sum_{k=1}^{K-1} e^{\boldsymbol\beta_k \cdot \mathbf{X}_i}} \\
&\cdots\cdots \\
\Pr(Y_i = K-1) &= \frac{e^{\boldsymbol\beta_{K-1} \cdot \mathbf{X}_i}}{1 + \sum_{k=1}^{K-1} e^{\boldsymbol\beta_k \cdot \mathbf{X}_i}}
\end{aligned}
$$

https://en.wikipedia.org/wiki/Multinomial_logistic_regression

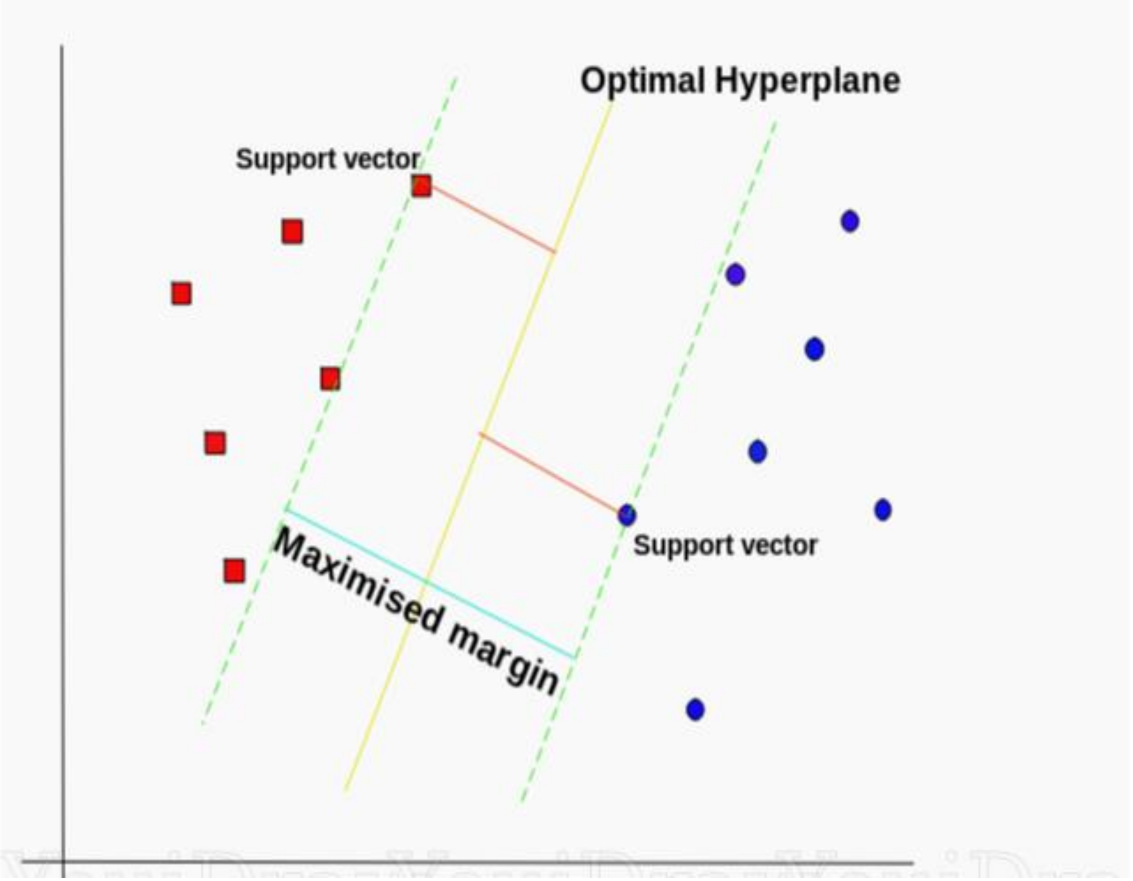
Logistic Regression vs SVM



$$\frac{e^{\beta^T X}}{1+e^{\beta^T X}} > 0.5$$

$$\frac{e^{\beta^T X}}{1+e^{\beta^T X}} = 0.5$$

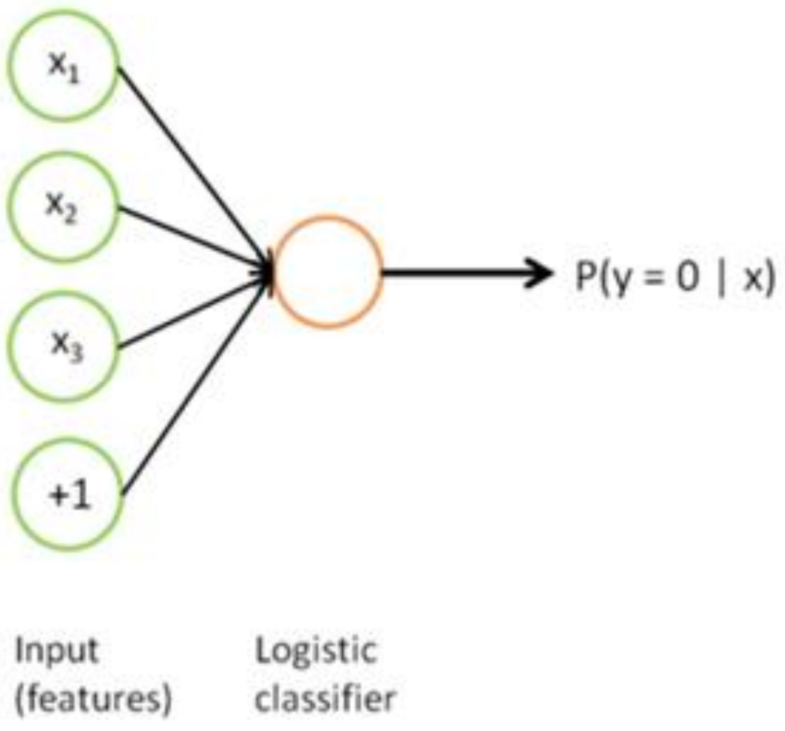
$$\frac{e^{\beta^T X}}{1+e^{\beta^T X}} < 0.5$$



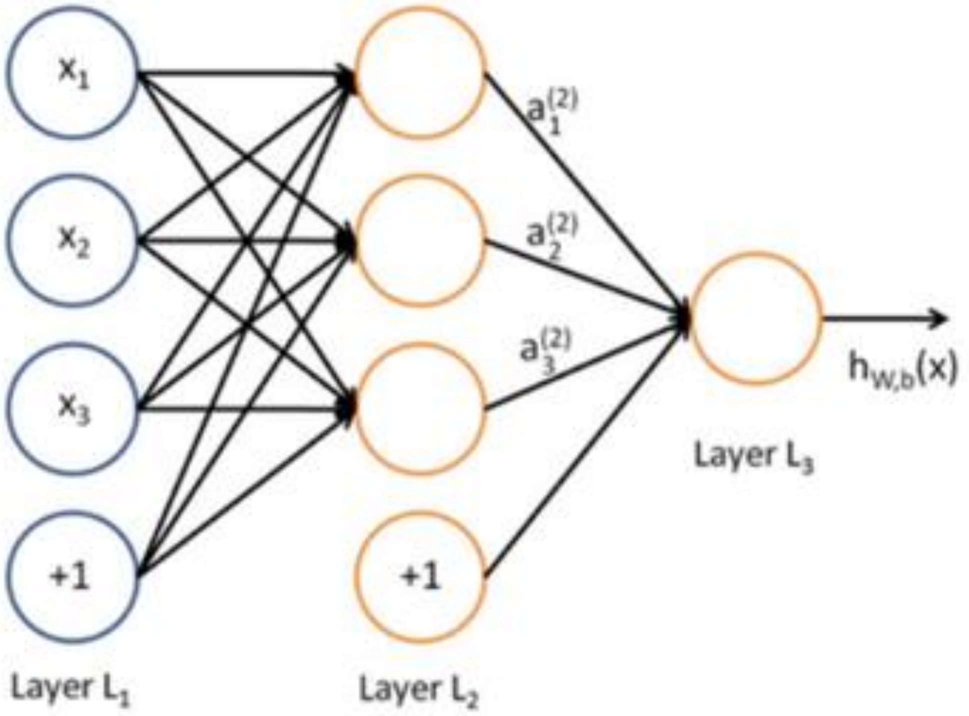
https://www.gushiciku.cn/pl/gWEr/zh-tw

https://gdcoder.com/support-vector-machine-vs-logistic-regression/

Logistic Regression vs Neural Network



Logistic Regression

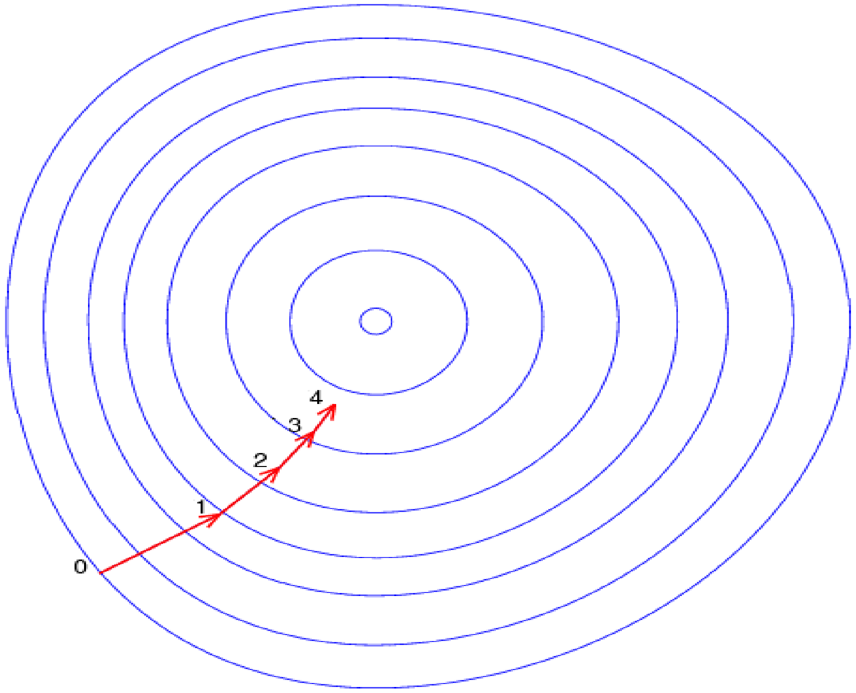


Neural Network

https://stats.stackexchange.com/questions/366707/a-logistic-regression-with-neural-network-mindset-vs-a-shallow-neural-network

General gradient Ascent/Descent

When we solve coefficient β ,we have to make likelihood function $LE = \prod_{i=1}^{n} f(x_i)$. Let the likelihood function be maximum, that is MLE, so that we can get β.
However, LE is difficult to solve, so we add log in front of LE



This is a function space of beta, we randomly choose a point and move. Use gradient ascent to find the highest location:

$$\beta^{(t+1)} = \beta^{(t)} + \alpha \frac{\partial logL(\beta)}{\partial \beta} \qquad \alpha > 0$$

similar how many steps for each epoch

similar how far for each step

Or use gradient descent to find the lowest location.

$$\beta^{(t+1)} = \beta^{(t)} - \alpha \frac{\partial logL(\beta)}{\partial \beta} \qquad \alpha > 0$$

We can casually choose the learning rate (alpha), but there is a better way.

https://zh.wikipedia.org/wiki/%E6%A2%AF%E5%BA%A6%E4%B8%8B%E9%99%8D%E6%B3%95

Other Gradient Ascent/Descent

1. Vanilla Gradient:

2. Stochastic Gradient Ascent/Descent:

3. Momentum Gradient Ascent/Descent:

4. Adagrad Gradient Ascent/Descent:

5. Adam:

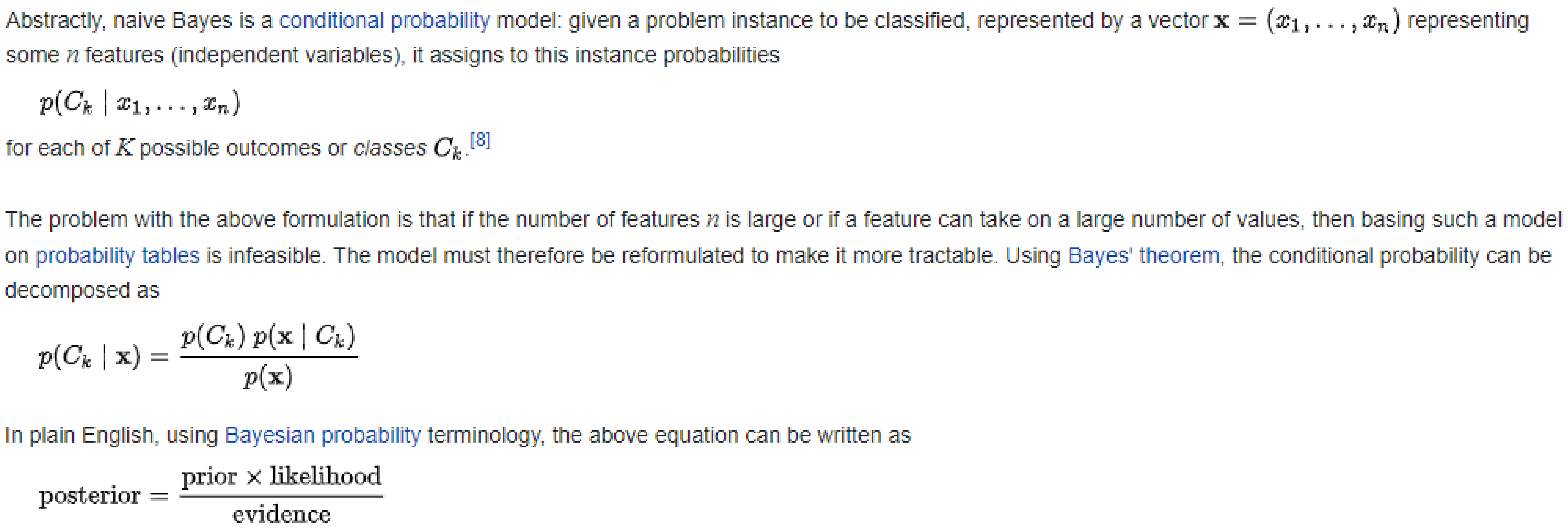
6. RMSProp:

- Ascent/Descenthttps://spinningup.openai.com/en/latest/algorithms/vpg.html
- https://medium.com/%E9%9B%9E%E9%9B%9E%E8%88%87%E5%85%94%E5%85%94%E7%9A%84%E5%B7%A5%E7%A8%8B%E4%B8%96%E7%95%8C/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92ml-note-sgd-momentum-adagrad-adam-optimizer-f20568c968db

- https://medium.com/chung-yi/ml%E5%85%A5%E9%96%80-%E5%8D%81%E4%BA%8C-sgd-adagrad-momentum-rmsprop-adam-optimizer-e331ef3cf5cf

Naive Bayes classifier

In statistics, **naive Bayes classifiers** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features



Abstractly, naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$ representing some n features (independent variables), it assigns to this instance probabilities

$$p(C_k \mid x_1, \ldots, x_n)$$

for each of K possible outcomes or *classes* C_k.[8]

The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. The model must therefore be reformulated to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as

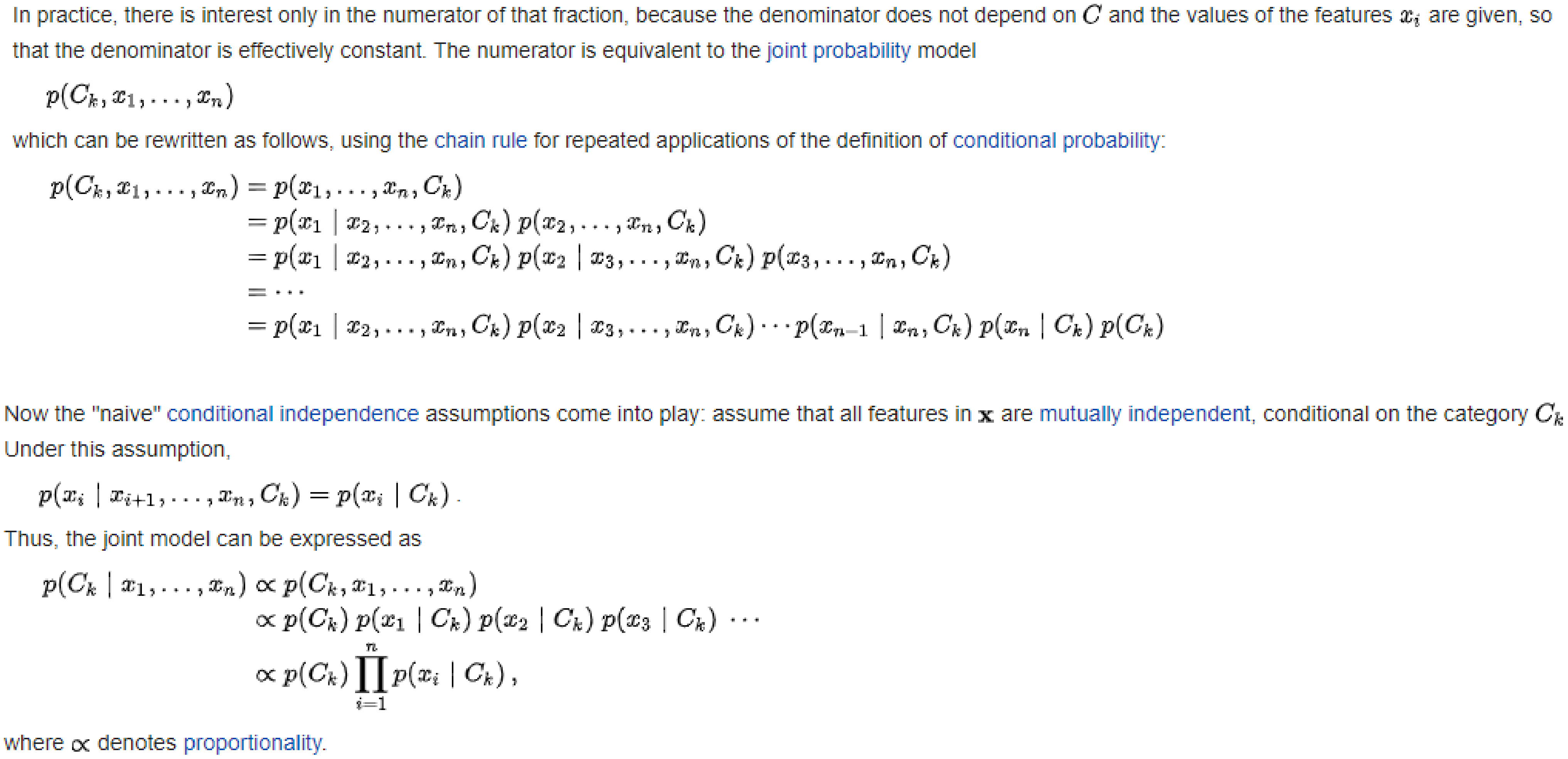
$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\ p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

In plain English, using Bayesian probability terminology, the above equation can be written as

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

Naive Bayes classifier



In practice, there is interest only in the numerator of that fraction, because the denominator does not depend on C and the values of the features x_i are given, so that the denominator is effectively constant. The numerator is equivalent to the joint probability model

$$p(C_k, x_1, \dots, x_n)$$

which can be rewritten as follows, using the chain rule for repeated applications of the definition of conditional probability:

$$\begin{aligned} p(C_k, x_1, \dots, x_n) &= p(x_1, \dots, x_n, C_k) \\ &= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2, \dots, x_n, C_k) \\ &= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2 \mid x_3, \dots, x_n, C_k)\, p(x_3, \dots, x_n, C_k) \\ &= \cdots \\ &= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2 \mid x_3, \dots, x_n, C_k) \cdots p(x_{n-1} \mid x_n, C_k)\, p(x_n \mid C_k)\, p(C_k) \end{aligned}$$

Now the "naive" conditional independence assumptions come into play: assume that all features in $\mathbf{x}$ are mutually independent, conditional on the category C_k. Under this assumption,

$$p(x_i \mid x_{i+1}, \dots, x_n, C_k) = p(x_i \mid C_k)\,.$$

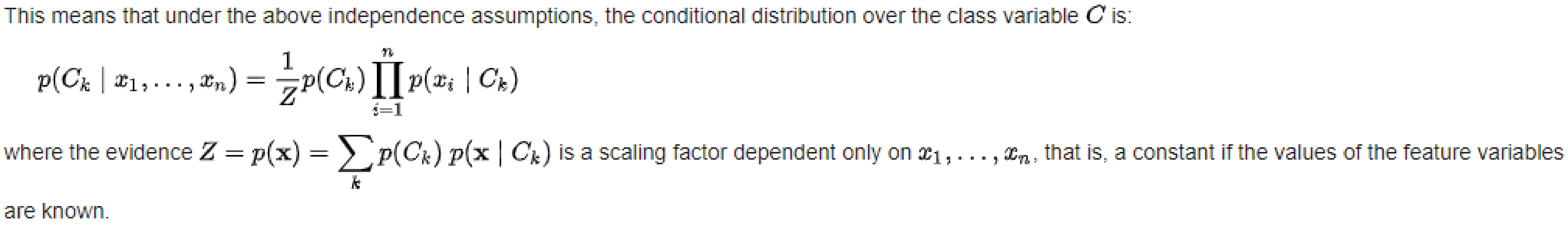
Thus, the joint model can be expressed as

$$\begin{aligned} p(C_k \mid x_1, \dots, x_n) &\propto p(C_k, x_1, \dots, x_n) \\ &\propto p(C_k)\, p(x_1 \mid C_k)\, p(x_2 \mid C_k)\, p(x_3 \mid C_k)\, \cdots \\ &\propto p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,, \end{aligned}$$

where $\propto$ denotes proportionality.

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

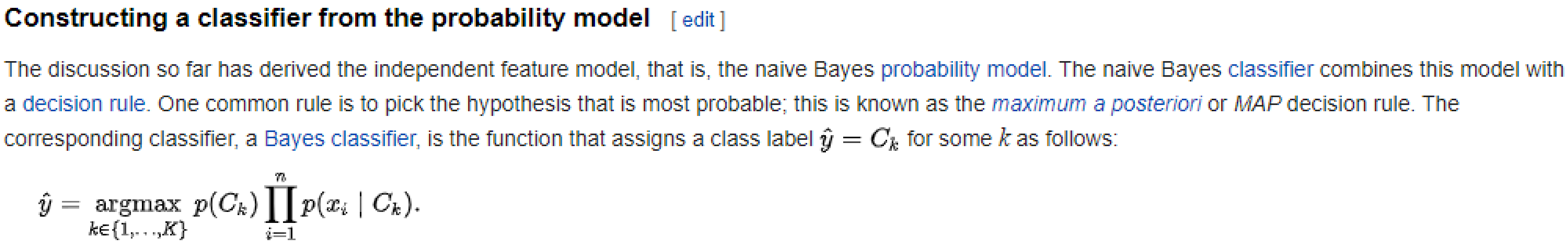
Naive Bayes classifier



This means that under the above independence assumptions, the conditional distribution over the class variable C is:

$$p(C_k \mid x_1, \dots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

where the evidence $Z = p(\mathbf{x}) = \sum_k p(C_k)\, p(\mathbf{x} \mid C_k)$ is a scaling factor dependent only on $x_1, \dots, x_n$, that is, a constant if the values of the feature variables are known.

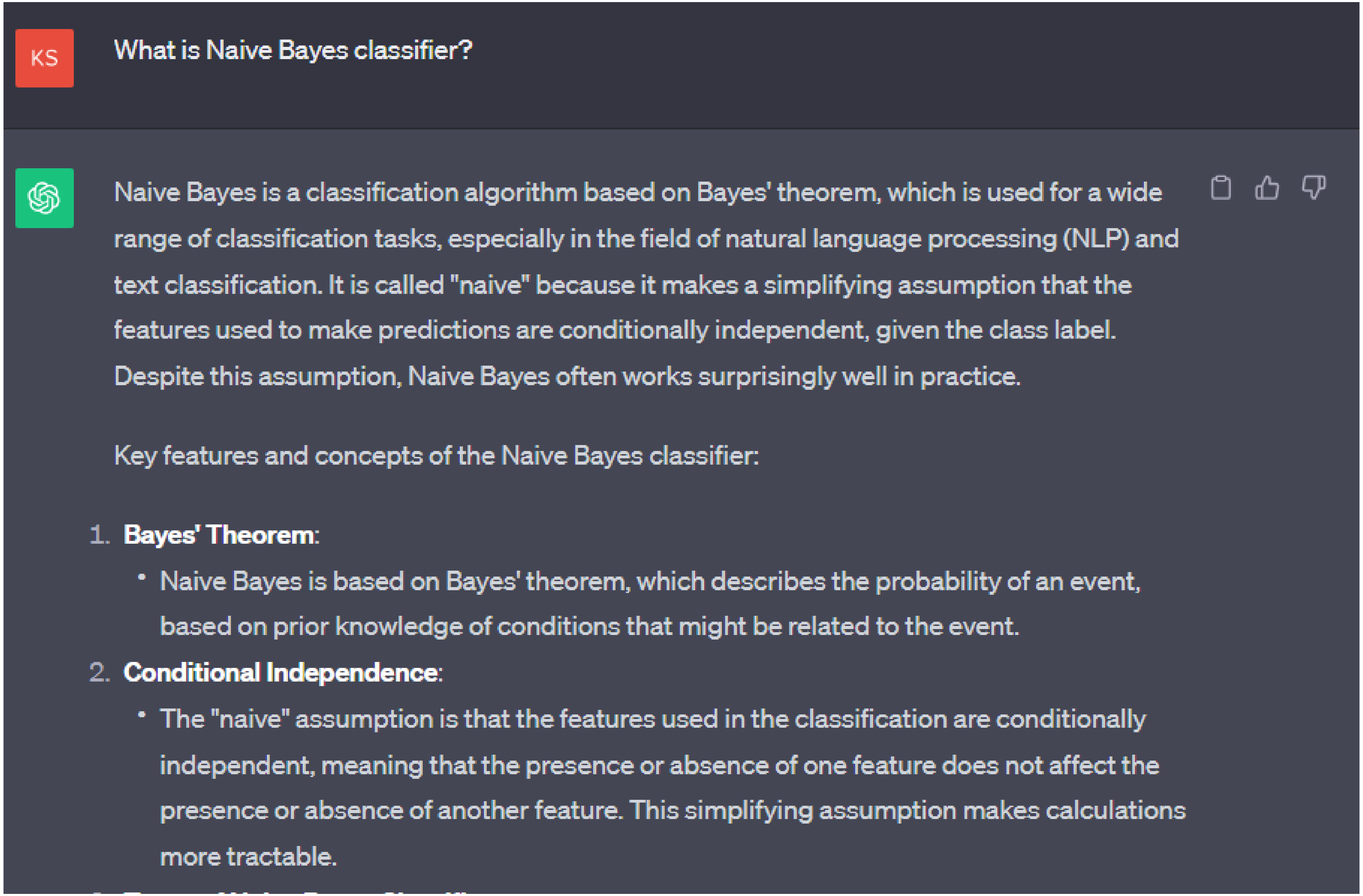


Constructing a classifier from the probability model [edit]

The discussion so far has derived the independent feature model, that is, the naive Bayes probability model. The naive Bayes classifier combines this model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the *maximum a posteriori* or *MAP* decision rule. The corresponding classifier, a Bayes classifier, is the function that assigns a class label $\hat{y} = C_k$ for some k as follows:

$$\hat{y} = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k).$$

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

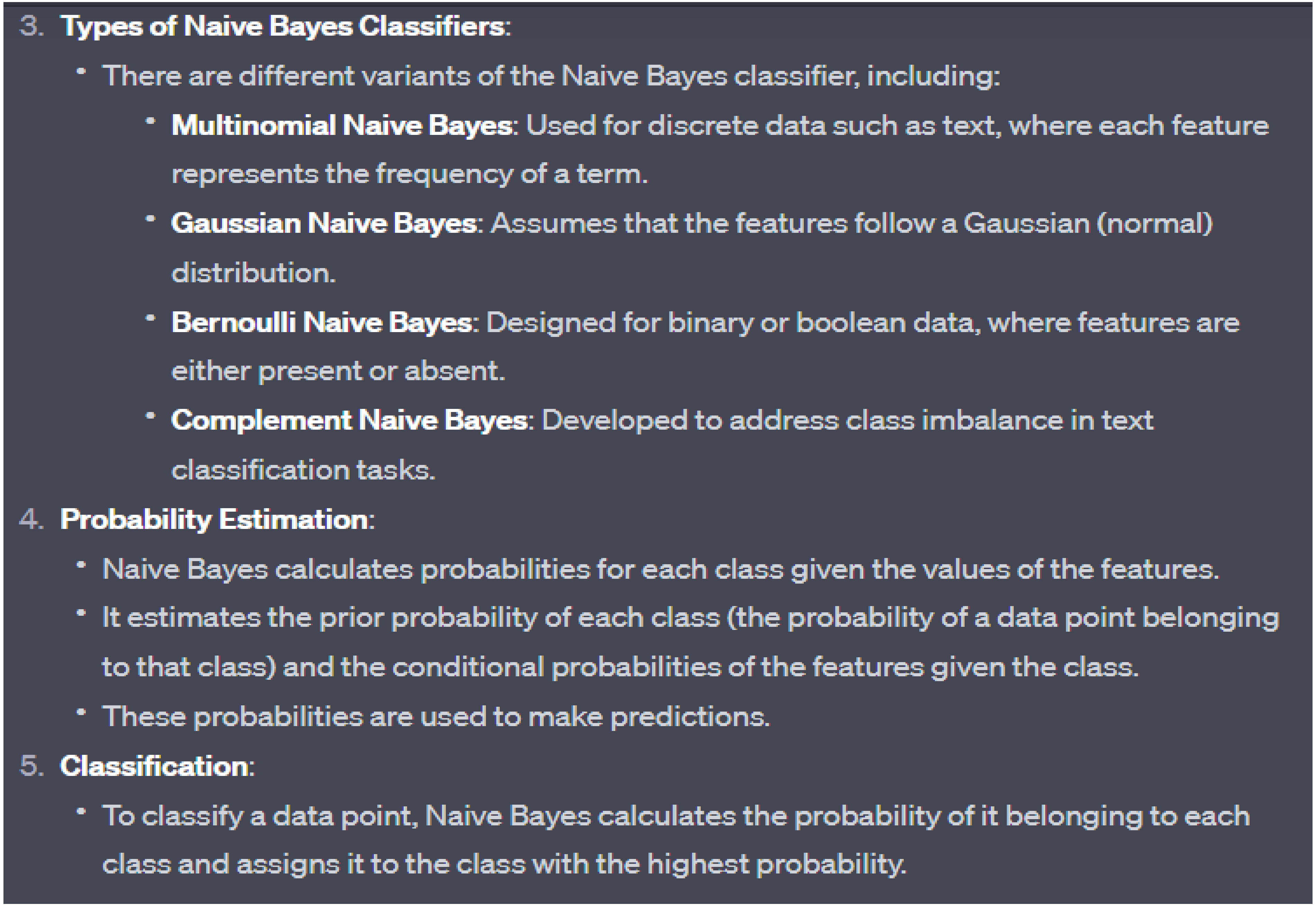


What is Naive Bayes classifier?

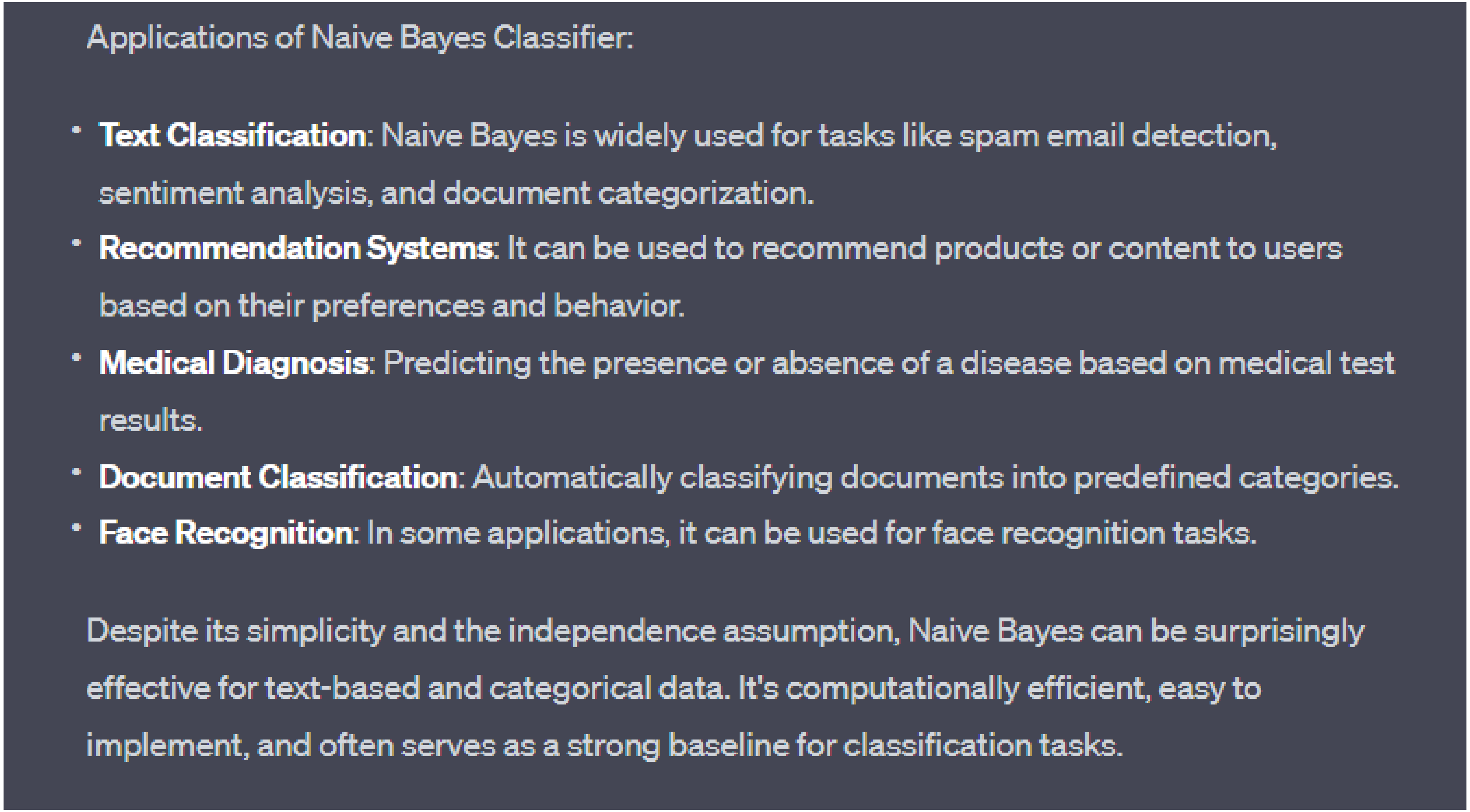
Naive Bayes is a classification algorithm based on Bayes' theorem, which is used for a wide range of classification tasks, especially in the field of natural language processing (NLP) and text classification. It is called "naive" because it makes a simplifying assumption that the features used to make predictions are conditionally independent, given the class label. Despite this assumption, Naive Bayes often works surprisingly well in practice.

Key features and concepts of the Naive Bayes classifier:

1. **Bayes' Theorem**:
    - Naive Bayes is based on Bayes' theorem, which describes the probability of an event, based on prior knowledge of conditions that might be related to the event.
2. **Conditional Independence**:
    - The "naive" assumption is that the features used in the classification are conditionally independent, meaning that the presence or absence of one feature does not affect the presence or absence of another feature. This simplifying assumption makes calculations more tractable.



3. **Types of Naive Bayes Classifiers**:
    - There are different variants of the Naive Bayes classifier, including:
        - **Multinomial Naive Bayes**: Used for discrete data such as text, where each feature represents the frequency of a term.
        - **Gaussian Naive Bayes**: Assumes that the features follow a Gaussian (normal) distribution.
        - **Bernoulli Naive Bayes**: Designed for binary or boolean data, where features are either present or absent.
        - **Complement Naive Bayes**: Developed to address class imbalance in text classification tasks.
4. **Probability Estimation**:
    - Naive Bayes calculates probabilities for each class given the values of the features.
    - It estimates the prior probability of each class (the probability of a data point belonging to that class) and the conditional probabilities of the features given the class.
    - These probabilities are used to make predictions.
5. **Classification**:
    - To classify a data point, Naive Bayes calculates the probability of it belonging to each class and assigns it to the class with the highest probability.



Applications of Naive Bayes Classifier:

- **Text Classification**: Naive Bayes is widely used for tasks like spam email detection, sentiment analysis, and document categorization.
- **Recommendation Systems**: It can be used to recommend products or content to users based on their preferences and behavior.
- **Medical Diagnosis**: Predicting the presence or absence of a disease based on medical test results.
- **Document Classification**: Automatically classifying documents into predefined categories.
- **Face Recognition**: In some applications, it can be used for face recognition tasks.

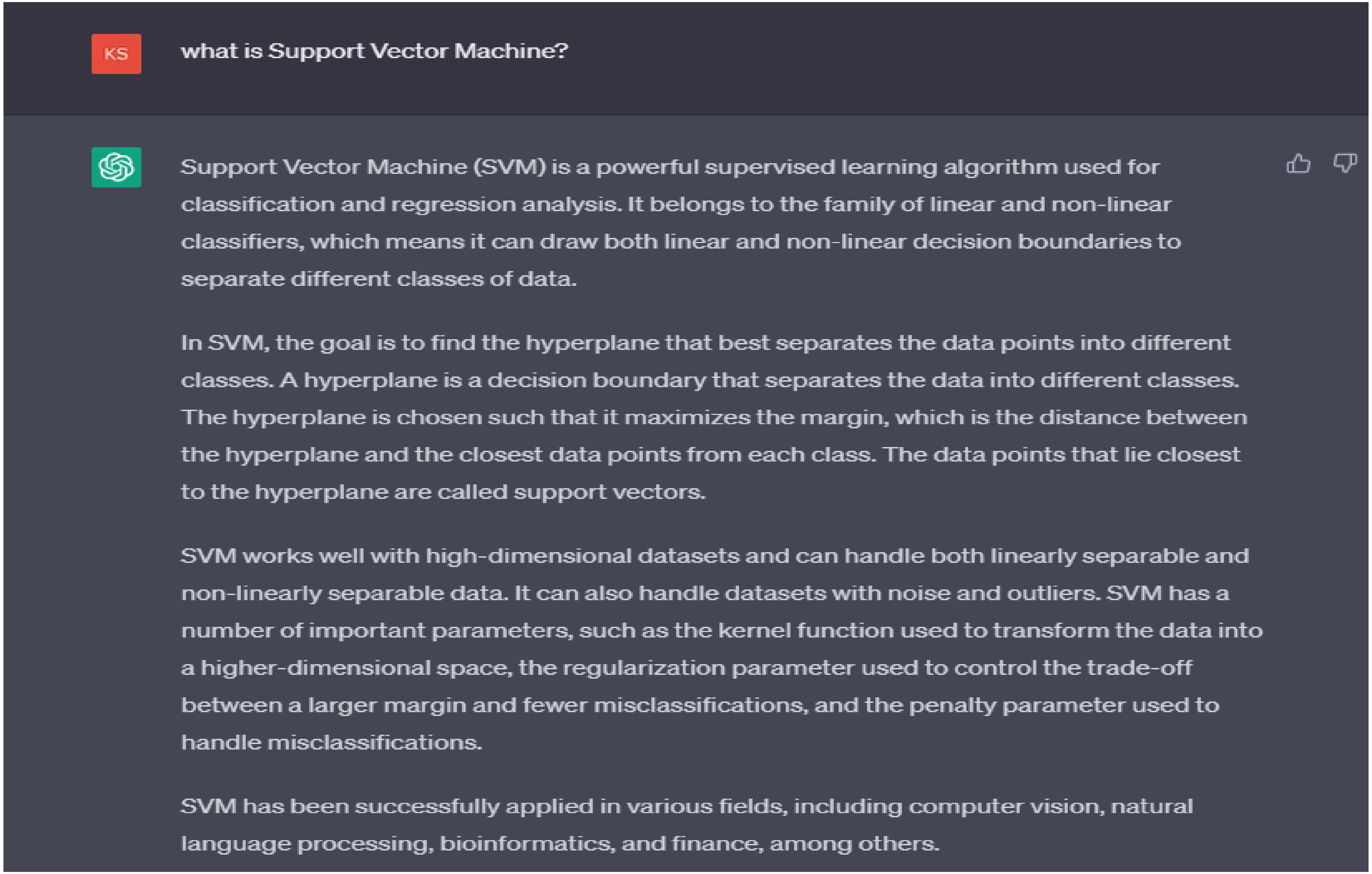
Despite its simplicity and the independence assumption, Naive Bayes can be surprisingly effective for text-based and categorical data. It's computationally efficient, easy to implement, and often serves as a strong baseline for classification tasks.

Support Vector Machine

- A classifier derived from statistical learning theory by Vapnik, et al. in 1992.

- SVM became famous when using images as input, it gave accuracy comparable to a neural network with hand-designed features in a handwriting recognition task.

- Currently, SVM is widely used in object detection & recognition, content-based image retrieval, text recognition, biometrics, speech recognition, etc.



what is Support Vector Machine?

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression analysis. It belongs to the family of linear and non-linear classifiers, which means it can draw both linear and non-linear decision boundaries to separate different classes of data.

In SVM, the goal is to find the hyperplane that best separates the data points into different classes. A hyperplane is a decision boundary that separates the data into different classes. The hyperplane is chosen such that it maximizes the margin, which is the distance between the hyperplane and the closest data points from each class. The data points that lie closest to the hyperplane are called support vectors.

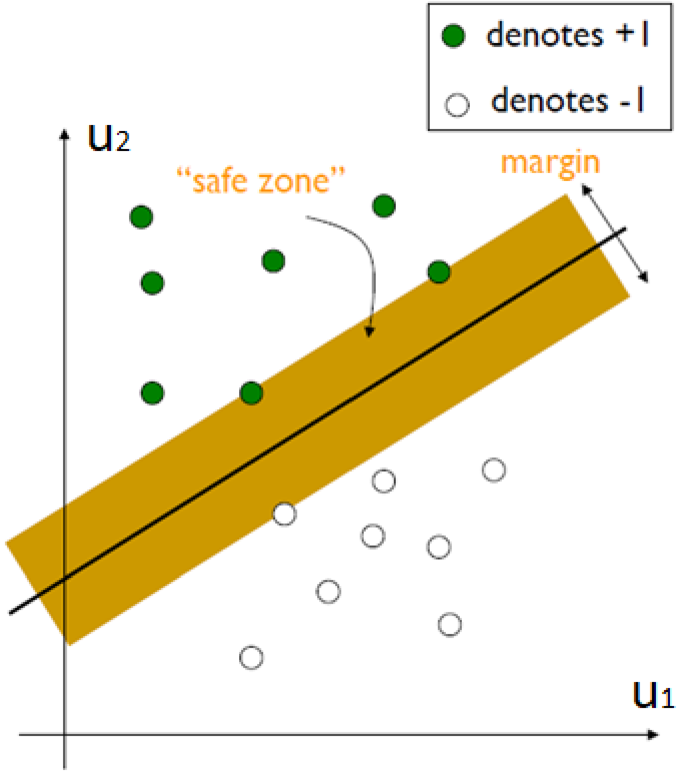
SVM works well with high-dimensional datasets and can handle both linearly separable and non-linearly separable data. It can also handle datasets with noise and outliers. SVM has a number of important parameters, such as the kernel function used to transform the data into a higher-dimensional space, the regularization parameter used to control the trade-off between a larger margin and fewer misclassifications, and the penalty parameter used to handle misclassifications.

SVM has been successfully applied in various fields, including computer vision, natural language processing, bioinformatics, and finance, among others.

https://chat.openai.com/

Large margin linear classifier

- The linear discriminant function (classifier) with the maximum margin is the best.
- Margin is defined as the width that the boundary could be increased by before hitting a data point.
- Robust to outliners and thus strong generalization ability is the reason for the best.



Given a set of data points: $\{(\boldsymbol{x}_i, y_i)\}, i = 1, \cdots, n,$ where

For $y_i = +1, \boldsymbol{w}^T \boldsymbol{x}_i + b > 0$

For $y_i = -1, \boldsymbol{w}^T \boldsymbol{x}_i + b < 0$

For margin = $2M$, then we have

For $y_i = +1,$

$$\frac{1}{\| \boldsymbol{w} \|}(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq M$$

For $y_i = -1,$

$$\frac{1}{\| \boldsymbol{w} \|}(\boldsymbol{w}^T \boldsymbol{x}_i + b) \leq -M$$

Since for any $\boldsymbol{w}$ and b satisfying these inequalities, any positively scaled multiple satisfies them too, we can arbitrarily set $\| \boldsymbol{w} \| = 1/M$

Thus above inequalities is equivalent to

For $y_i = +1, \boldsymbol{w}^T \boldsymbol{x}_i + b \geq 1$

For $y_i = -1, \boldsymbol{w}^T \boldsymbol{x}_i + b \leq -1$

Large margin linear classifier

We know that

$$\boldsymbol{w}^T\boldsymbol{x}^+ + b = 1$$

$$\boldsymbol{w}^T\boldsymbol{x}^- + b = -1$$

The margin width is:

$$2M = (\boldsymbol{x}^+ - \boldsymbol{x}^-)\cdot\boldsymbol{n}$$

$$= (\boldsymbol{x}^+ - \boldsymbol{x}^-)\cdot\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$$

The margin is the projection of (x^+-x^-) along the direction of **w**

Formulation:

$$\text{maximize } \frac{2}{\|\boldsymbol{w}\|}$$

such that

$$\text{For } y_i = +1, \boldsymbol{w}^T\boldsymbol{x}_i + b \geq 1$$

$$\text{For } y_i = -1, \boldsymbol{w}^T\boldsymbol{x}_i + b \leq -1$$

Formulation:

$$\text{minimize } \frac{1}{2}\|\boldsymbol{w}\|^2$$

such that

$$\text{For } y_i = +1, \boldsymbol{w}^T\boldsymbol{x}_i + b \geq 1$$

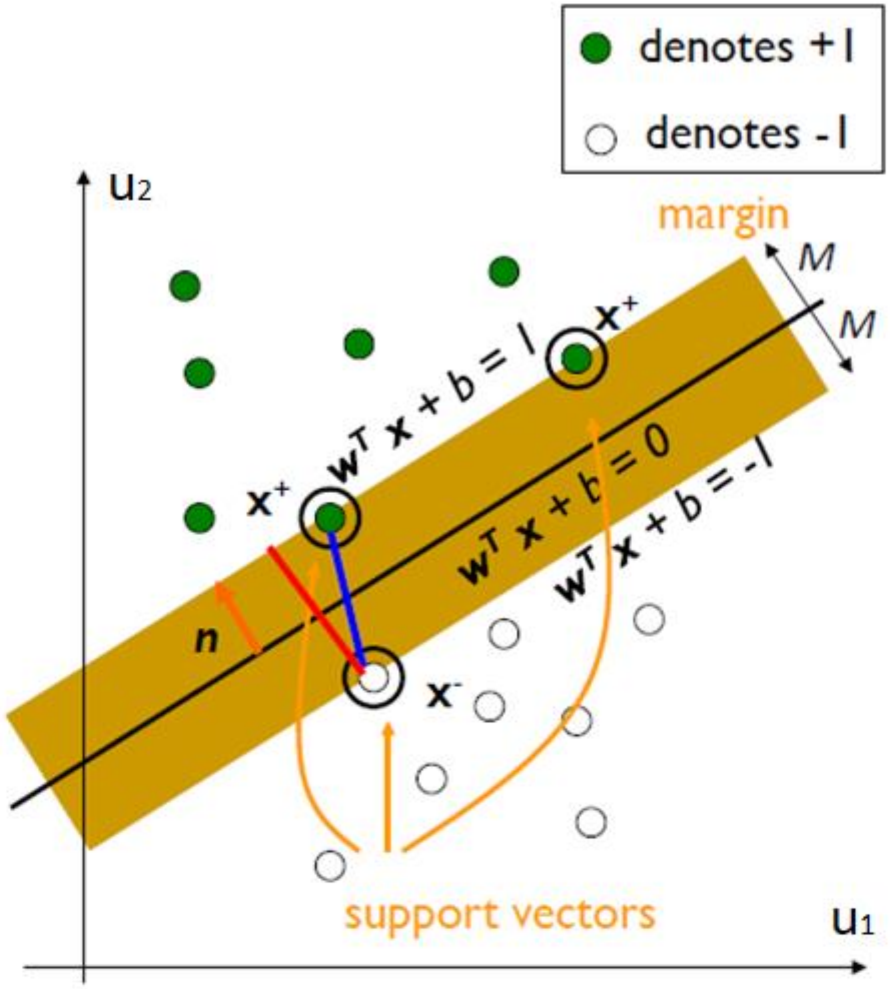
$$\text{For } y_i = -1, \boldsymbol{w}^T\boldsymbol{x}_i + b \leq -1$$

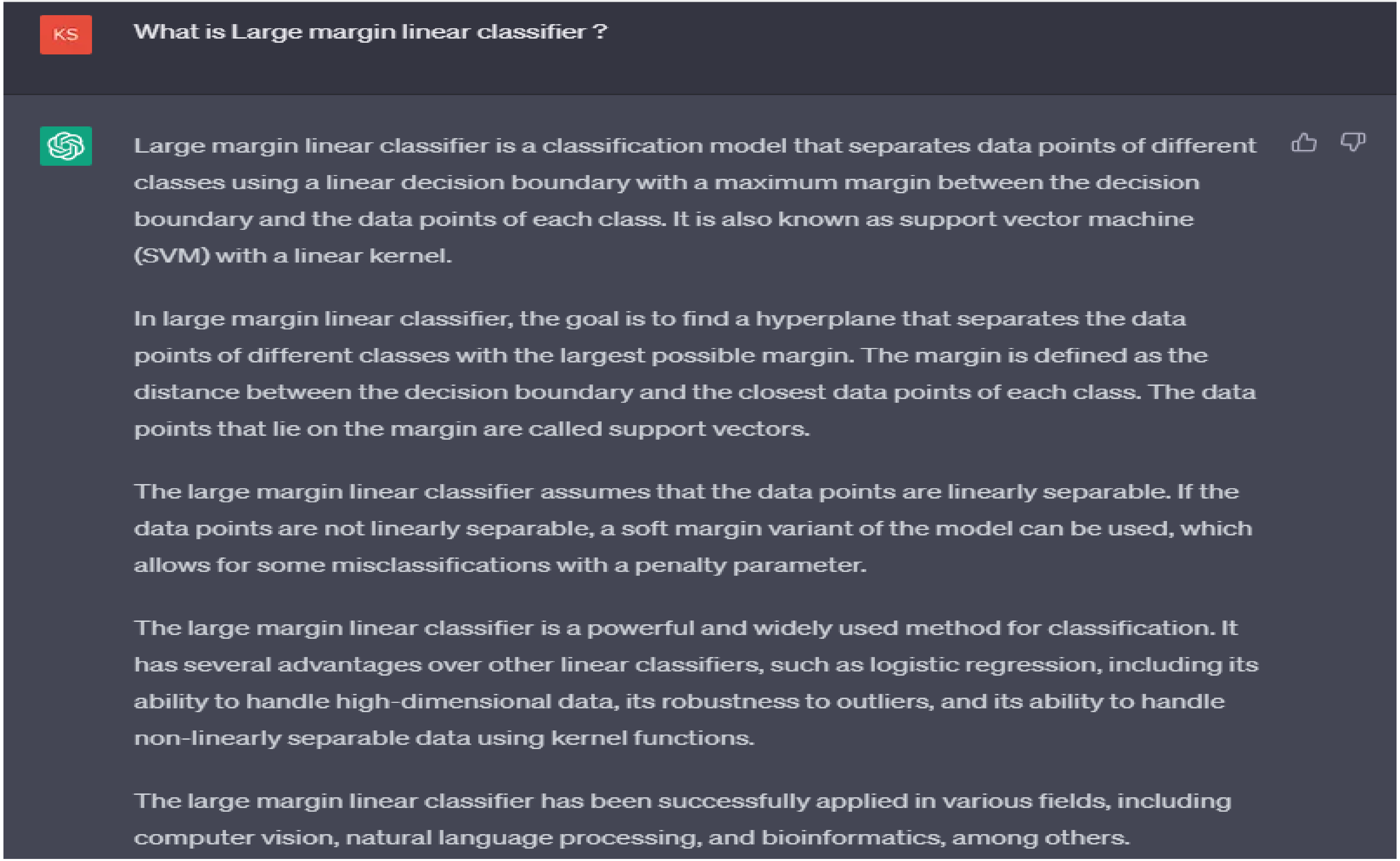
Formulation:

$$\text{minimize } \frac{1}{2}\|\boldsymbol{w}\|^2$$

such that

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1$$





What is Large margin linear classifier ?

Large margin linear classifier is a classification model that separates data points of different classes using a linear decision boundary with a maximum margin between the decision boundary and the data points of each class. It is also known as support vector machine (SVM) with a linear kernel.

In large margin linear classifier, the goal is to find a hyperplane that separates the data points of different classes with the largest possible margin. The margin is defined as the distance between the decision boundary and the closest data points of each class. The data points that lie on the margin are called support vectors.

The large margin linear classifier assumes that the data points are linearly separable. If the data points are not linearly separable, a soft margin variant of the model can be used, which allows for some misclassifications with a penalty parameter.

The large margin linear classifier is a powerful and widely used method for classification. It has several advantages over other linear classifiers, such as logistic regression, including its ability to handle high-dimensional data, its robustness to outliers, and its ability to handle non-linearly separable data using kernel functions.

The large margin linear classifier has been successfully applied in various fields, including computer vision, natural language processing, and bioinformatics, among others.

https://chat.openai.com/

Solving the optimization problem

Quadratic programming with linear constraints

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$

Lagrangian function

$$\text{minimize} \quad L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \right)$$

$$\text{s.t.} \quad \alpha_i \geq 0$$

$$\text{minimize} \quad L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \right)$$

$$\text{s.t.} \quad \alpha_i \geq 0$$

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L_p}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Solving the optimization problem

$$\text{minimize} \quad L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \right)$$

$$\text{s.t.} \qquad \alpha_i \geq 0$$

Lagrangian dual problem

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{s.t.} \qquad \alpha_i \geq 0 \text{ , and} \qquad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Solving the optimization problem

From KKT condition, we know:

$$\alpha_i(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1) = 0$$

Thus, only support vectors have $\alpha_i \neq 0$

The solution has the form:

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i = \sum_{i \in \text{SV}} \alpha_i y_i \boldsymbol{x}_i$$

get b from $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1 = 0$, where $\boldsymbol{x}_i$ is a support vector

The linear discriminant function is:

$$g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = \sum_{i \in \text{SV}} \alpha_i y_i \boldsymbol{x}_i{}^T \boldsymbol{x} + b$$

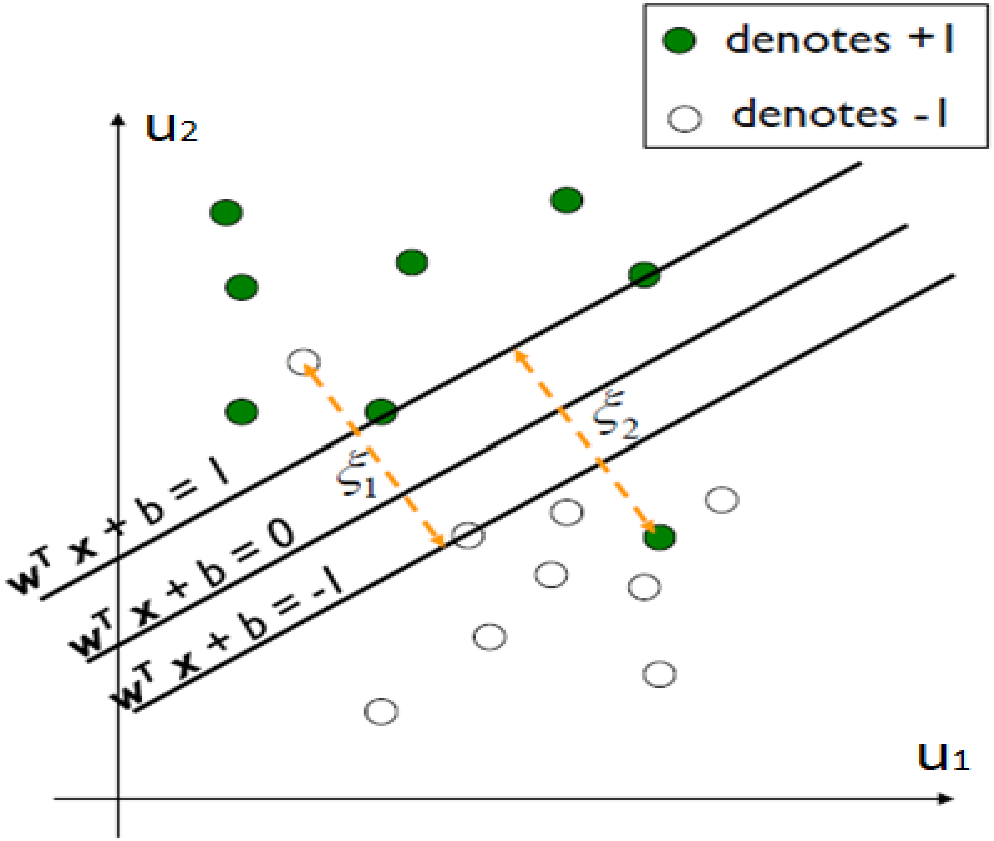
Notice it relies on a dot product between the test point $\boldsymbol{x}$ and the support vectors $\boldsymbol{x}_i$

Also keep in mind that solving the optimization problem involved computing the dot products $\boldsymbol{x}_i{}^T\boldsymbol{x}_i$ between all pairs of training points

Soft margin linear classifier

What if data is not linear separable? (noisy data, outliers, etc.)

Slack variables ξ_i can be added to allow mis-classification of difficult or noisy data points, resulting margins are called soft



Soft margin linear classifier

Formulation:

$$\text{minimize } \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n} \xi_i$$

such that

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

Parameter C can be viewed as a way to control over-fitting.

Soft margin linear classifier

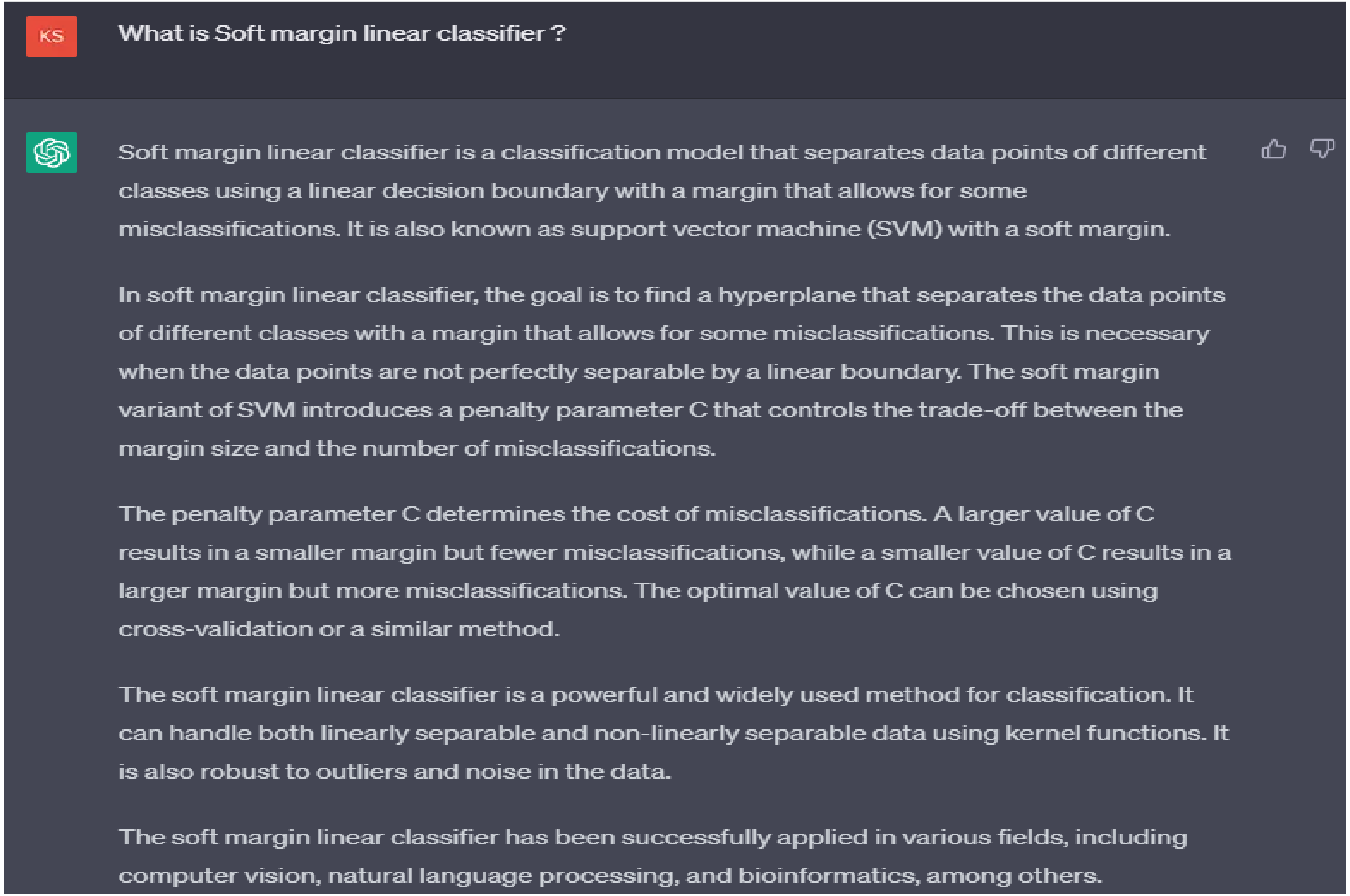
Formulation: (Lagrangian dual problem)

$$\text{maximize} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i{}^T \boldsymbol{x}_j$$

such that

$$0 \leq \alpha_i \leq C$$

$$\sum\nolimits_{i=1}^{n} \alpha_i y_i = 0$$



KS

What is Soft margin linear classifier ?

Soft margin linear classifier is a classification model that separates data points of different classes using a linear decision boundary with a margin that allows for some misclassifications. It is also known as support vector machine (SVM) with a soft margin.

In soft margin linear classifier, the goal is to find a hyperplane that separates the data points of different classes with a margin that allows for some misclassifications. This is necessary when the data points are not perfectly separable by a linear boundary. The soft margin variant of SVM introduces a penalty parameter C that controls the trade-off between the margin size and the number of misclassifications.

The penalty parameter C determines the cost of misclassifications. A larger value of C results in a smaller margin but fewer misclassifications, while a smaller value of C results in a larger margin but more misclassifications. The optimal value of C can be chosen using cross-validation or a similar method.

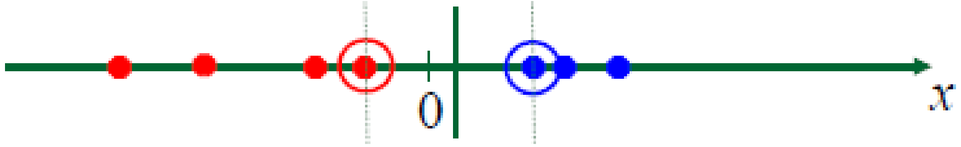
The soft margin linear classifier is a powerful and widely used method for classification. It can handle both linearly separable and non-linearly separable data using kernel functions. It is also robust to outliers and noise in the data.

The soft margin linear classifier has been successfully applied in various fields, including computer vision, natural language processing, and bioinformatics, among others.

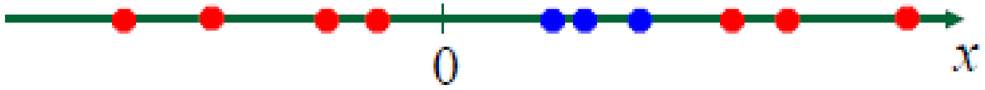
https://chat.openai.com/

Non-linear SVMs

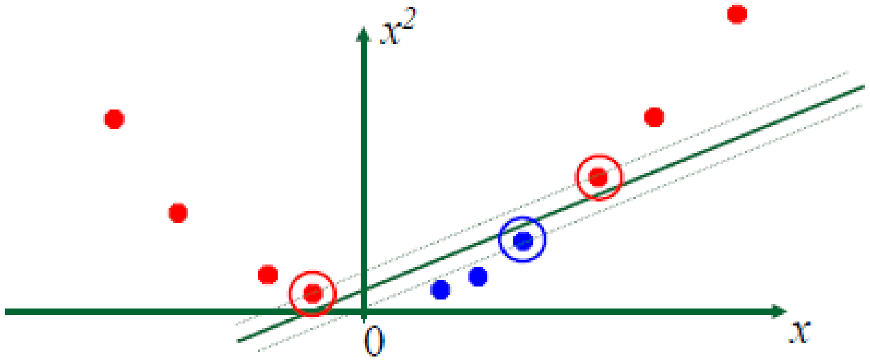
Datasets that are linearly separable with noise work out great:



But what are we going to do if the dataset is just too hard?

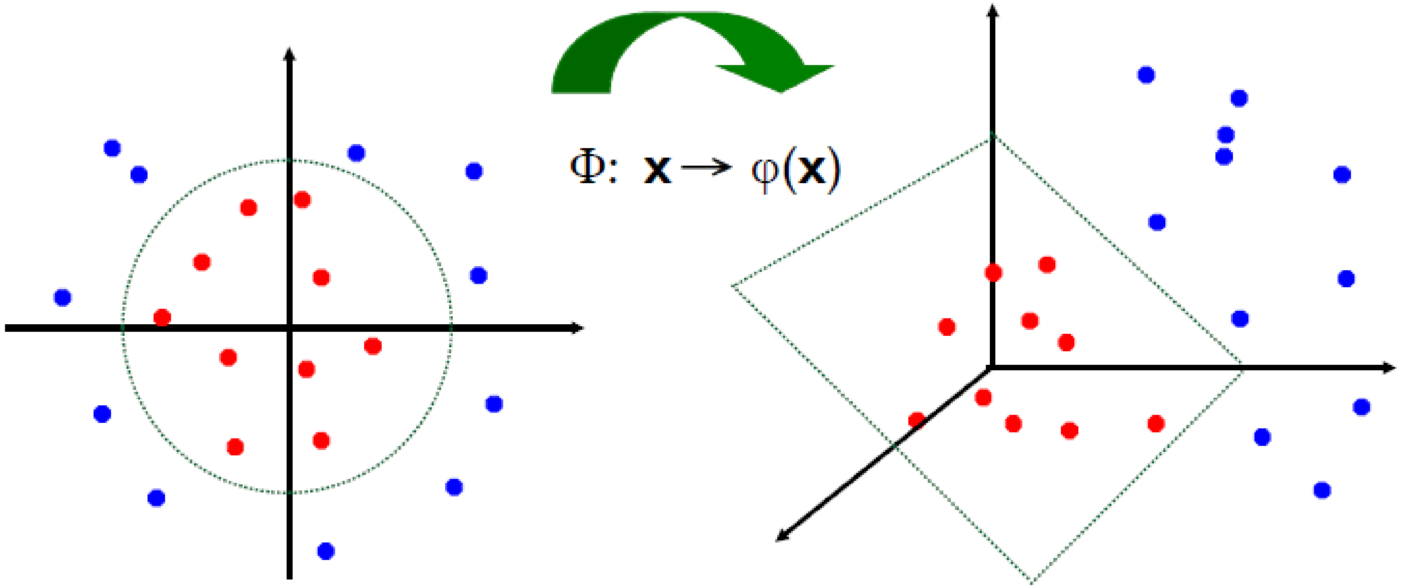


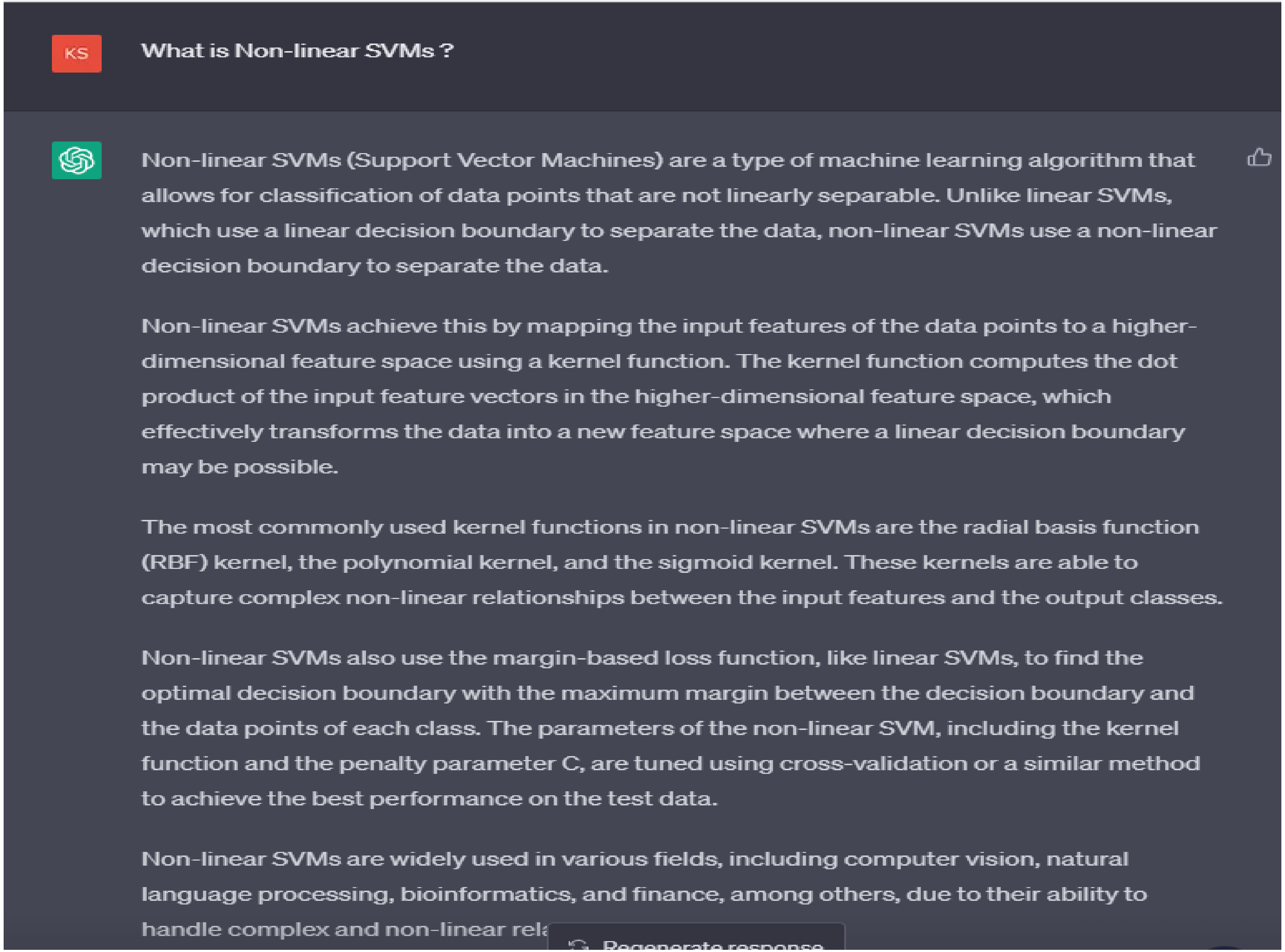
mapping data to a higher-dimensional space:



Non-linear SVMs: feature space

General idea: the original input space can be mapped to some higher-dimensional feature space where the training set is separable:





What is Non-linear SVMs ?

Non-linear SVMs (Support Vector Machines) are a type of machine learning algorithm that allows for classification of data points that are not linearly separable. Unlike linear SVMs, which use a linear decision boundary to separate the data, non-linear SVMs use a non-linear decision boundary to separate the data.

Non-linear SVMs achieve this by mapping the input features of the data points to a higher-dimensional feature space using a kernel function. The kernel function computes the dot product of the input feature vectors in the higher-dimensional feature space, which effectively transforms the data into a new feature space where a linear decision boundary may be possible.

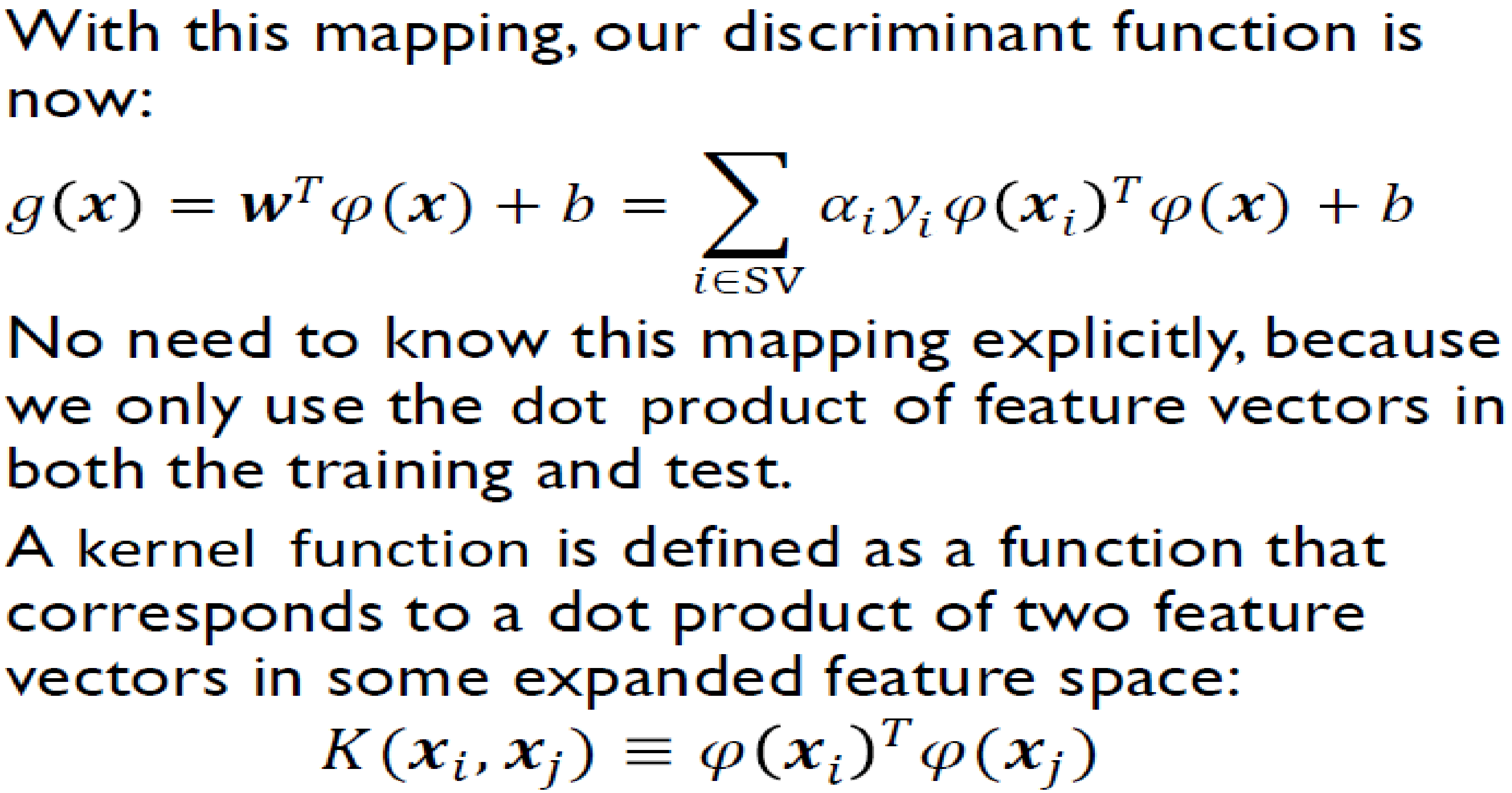
The most commonly used kernel functions in non-linear SVMs are the radial basis function (RBF) kernel, the polynomial kernel, and the sigmoid kernel. These kernels are able to capture complex non-linear relationships between the input features and the output classes.

Non-linear SVMs also use the margin-based loss function, like linear SVMs, to find the optimal decision boundary with the maximum margin between the decision boundary and the data points of each class. The parameters of the non-linear SVM, including the kernel function and the penalty parameter C, are tuned using cross-validation or a similar method to achieve the best performance on the test data.

Non-linear SVMs are widely used in various fields, including computer vision, natural language processing, bioinformatics, and finance, among others, due to their ability to handle complex and non-linear rela

https://chat.openai.com/

Nonlinear SVMs: the kernel trick



With this mapping, our discriminant function is now:

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \varphi(\boldsymbol{x}) + b = \sum_{i \in \mathrm{SV}} \alpha_i y_i \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}) + b$$

No need to know this mapping explicitly, because we only use the dot product of feature vectors in both the training and test.

A kernel function is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)$$

Nonlinear SVMs: the kernel trick

An example:

2-dimensional vectors $\mathbf{x}=[x_1 \;\; x_2]$;

let $K(\mathrm{x}_i,\mathrm{x}_j)=(1 + \mathrm{x}_i^{\mathrm{T}}\mathrm{x}_j)^2$,

Need to show that $K(\mathrm{x}_i,\mathrm{x}_j) = \varphi(\mathrm{x}_i)^{\mathrm{T}}\varphi(\mathrm{x}_j)$:

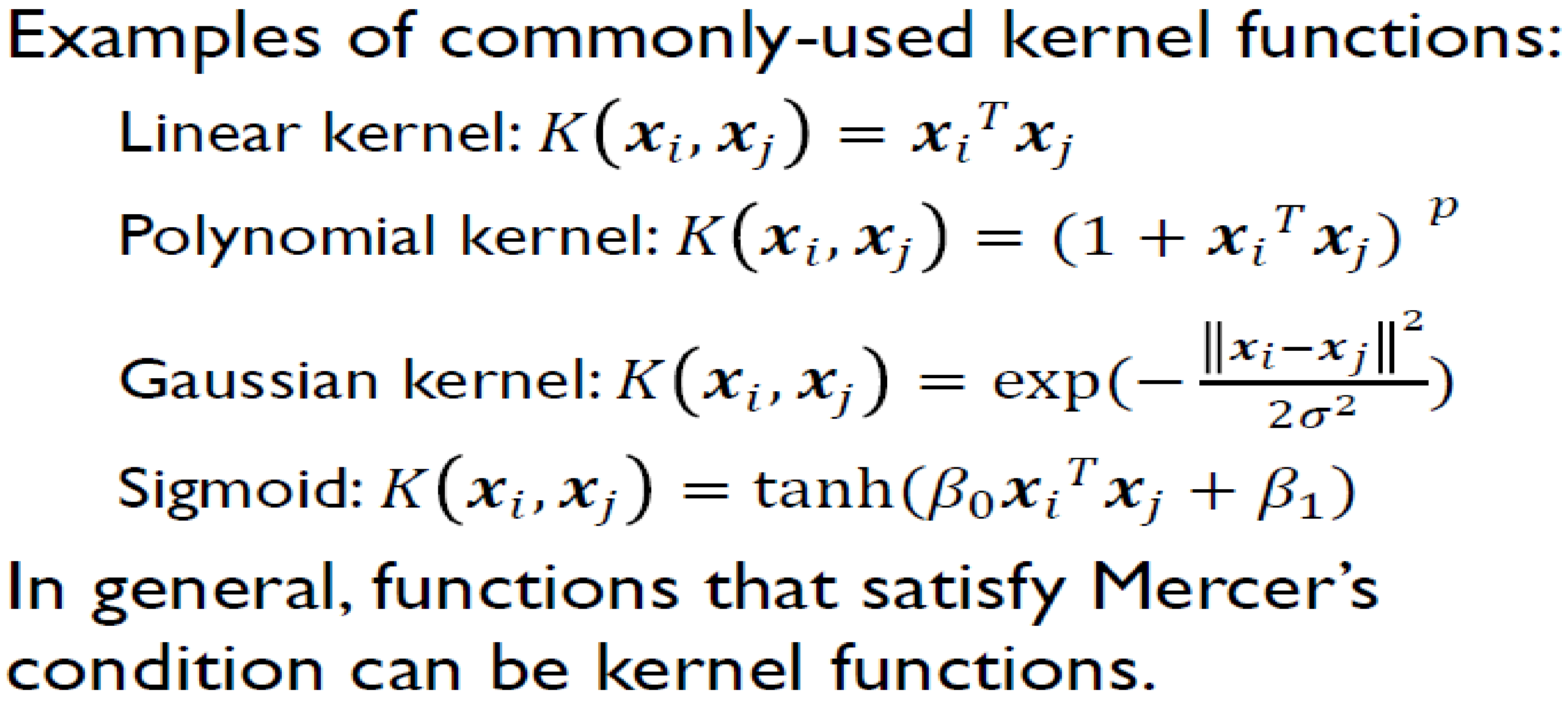
$$K(\mathrm{x}_i,\mathrm{x}_j)=(1 + \mathrm{x}_i^{\mathrm{T}}\mathrm{x}_j)^2$$

$$= 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$

$$=[1 \;\; x_{i1}^2 \;\; \sqrt{2}x_{i1}x_{i2} \;\; x_{i2}^2 \;\; \sqrt{2}x_{i1} \;\; \sqrt{2}x_{i2}]^{\mathrm{T}} [1 \;\; x_{j1}^2 \;\; \sqrt{2}x_{j1}x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2}x_{j1} \;\; \sqrt{2}x_{j2}]$$

$$= \varphi(\mathrm{x}_i)^{\mathrm{T}}\varphi(\mathrm{x}_j), \quad \text{where } \varphi(\mathrm{x}) = [1 \;\; x_1^2 \;\; \sqrt{2}\, x_1x_2 \;\; x_2^2 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2]$$

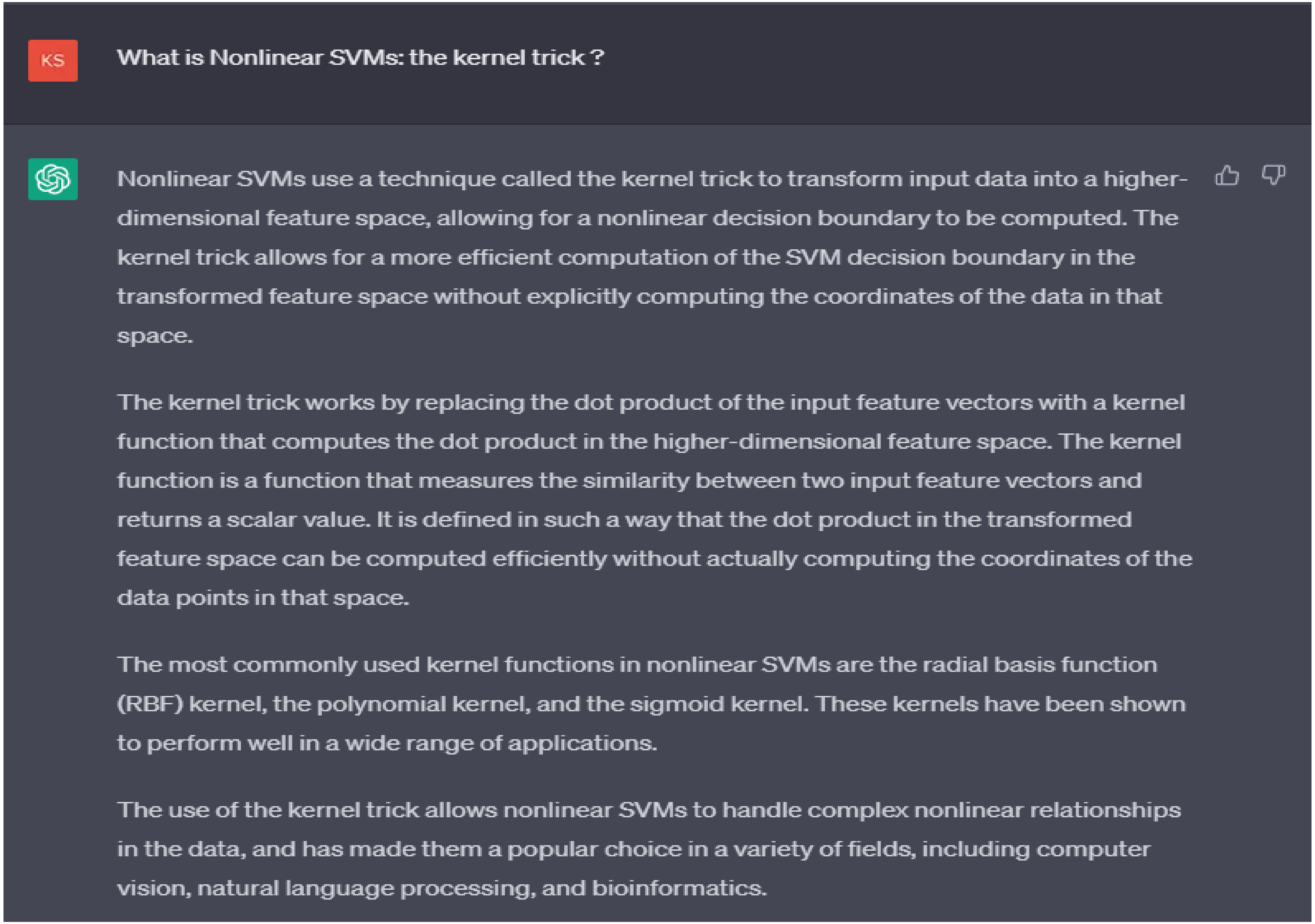
Nonlinear SVMs: the kernel trick



Examples of commonly-used kernel functions:

- Linear kernel: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i{}^T \boldsymbol{x}_j$
- Polynomial kernel: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i{}^T \boldsymbol{x}_j)^{\,p}$
- Gaussian kernel: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$
- Sigmoid: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\beta_0 \boldsymbol{x}_i{}^T \boldsymbol{x}_j + \beta_1)$

In general, functions that satisfy Mercer's condition can be kernel functions.



What is Nonlinear SVMs: the kernel trick ?

Nonlinear SVMs use a technique called the kernel trick to transform input data into a higher-dimensional feature space, allowing for a nonlinear decision boundary to be computed. The kernel trick allows for a more efficient computation of the SVM decision boundary in the transformed feature space without explicitly computing the coordinates of the data in that space.

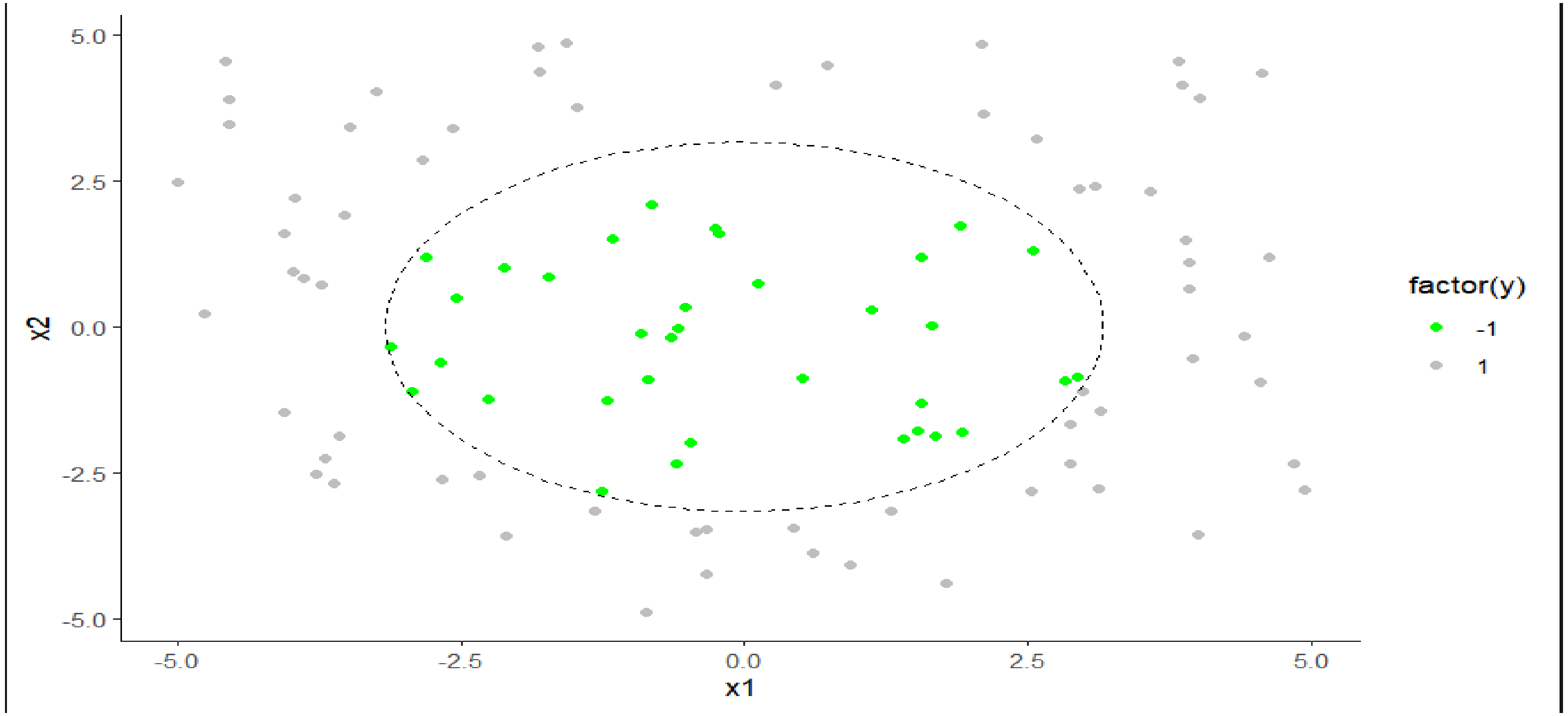
The kernel trick works by replacing the dot product of the input feature vectors with a kernel function that computes the dot product in the higher-dimensional feature space. The kernel function is a function that measures the similarity between two input feature vectors and returns a scalar value. It is defined in such a way that the dot product in the transformed feature space can be computed efficiently without actually computing the coordinates of the data points in that space.

The most commonly used kernel functions in nonlinear SVMs are the radial basis function (RBF) kernel, the polynomial kernel, and the sigmoid kernel. These kernels have been shown to perform well in a wide range of applications.

The use of the kernel trick allows nonlinear SVMs to handle complex nonlinear relationships in the data, and has made them a popular choice in a variety of fields, including computer vision, natural language processing, and bioinformatics.

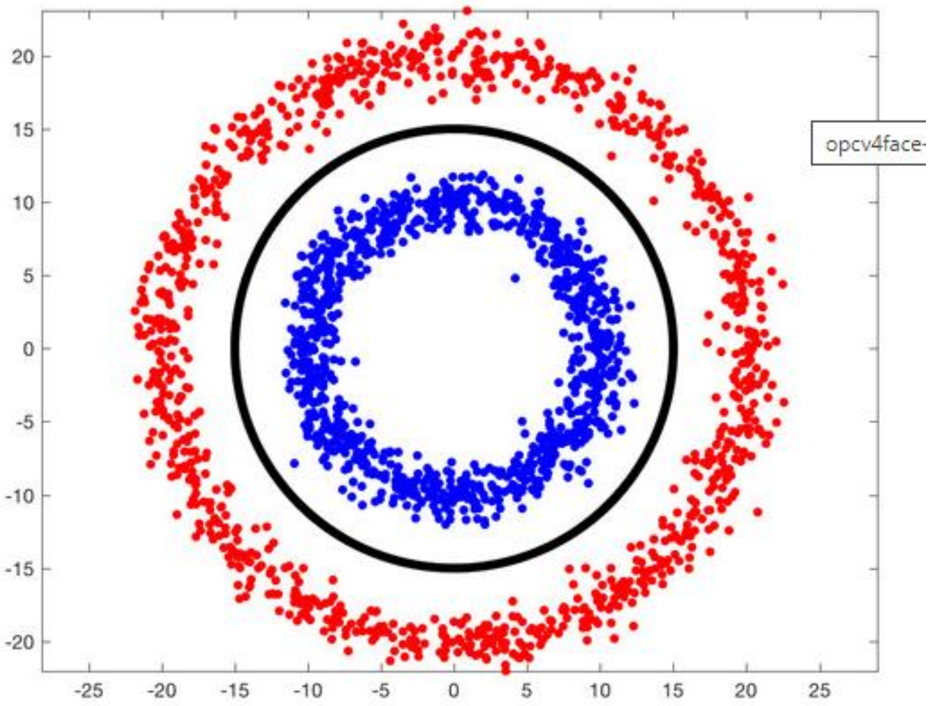
https://chat.openai.com/

Nonlinear SVMs: the kernel trick

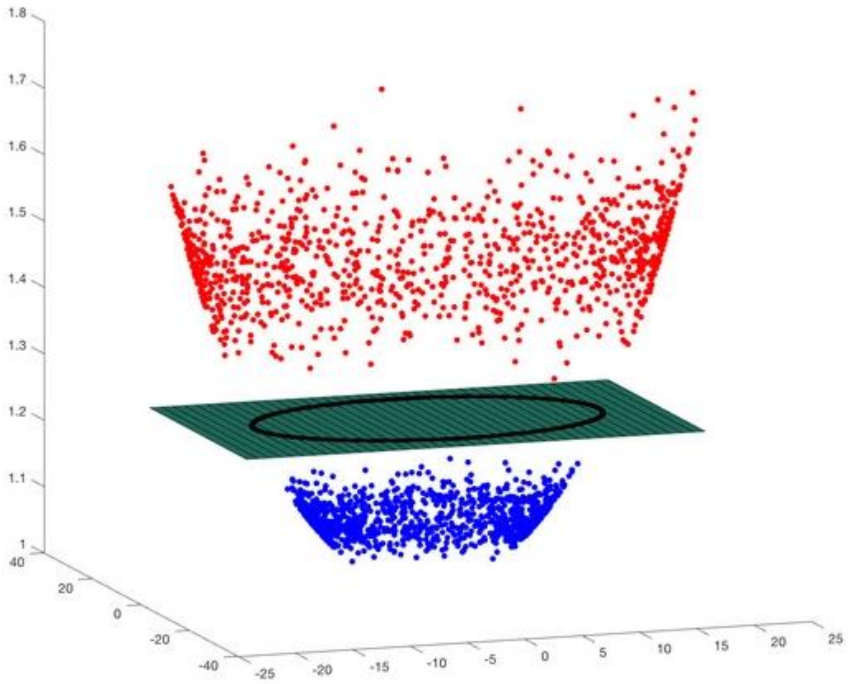


Demo code

Nonlinear SVMs: the kernel trick



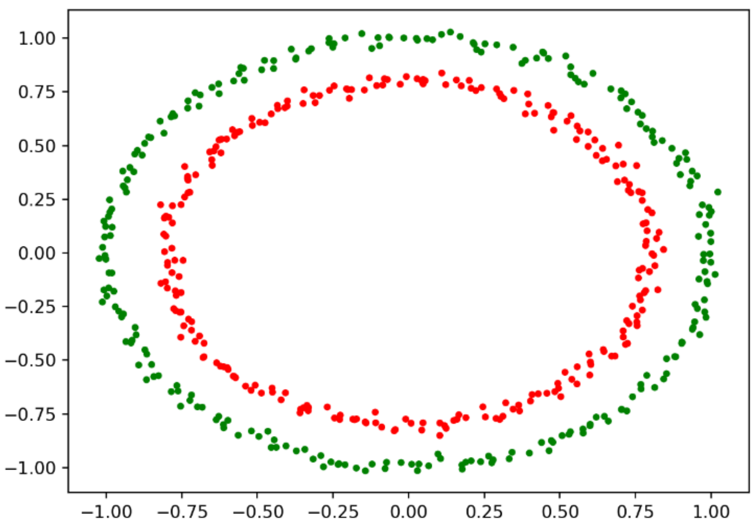
Non-Linearly Separable Data



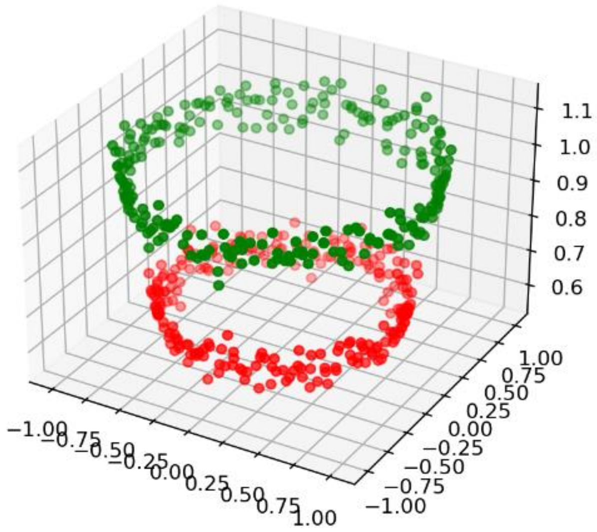
Kernel Trick to make data linearly separable.

https://learnopencv.com/support-vector-machines-svm/

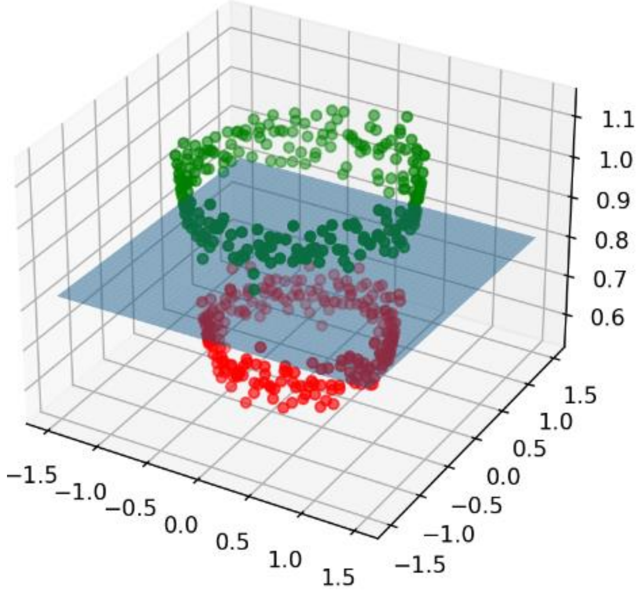
Nonlinear SVMs: the kernel trick



Original dataset



Transformed dataset

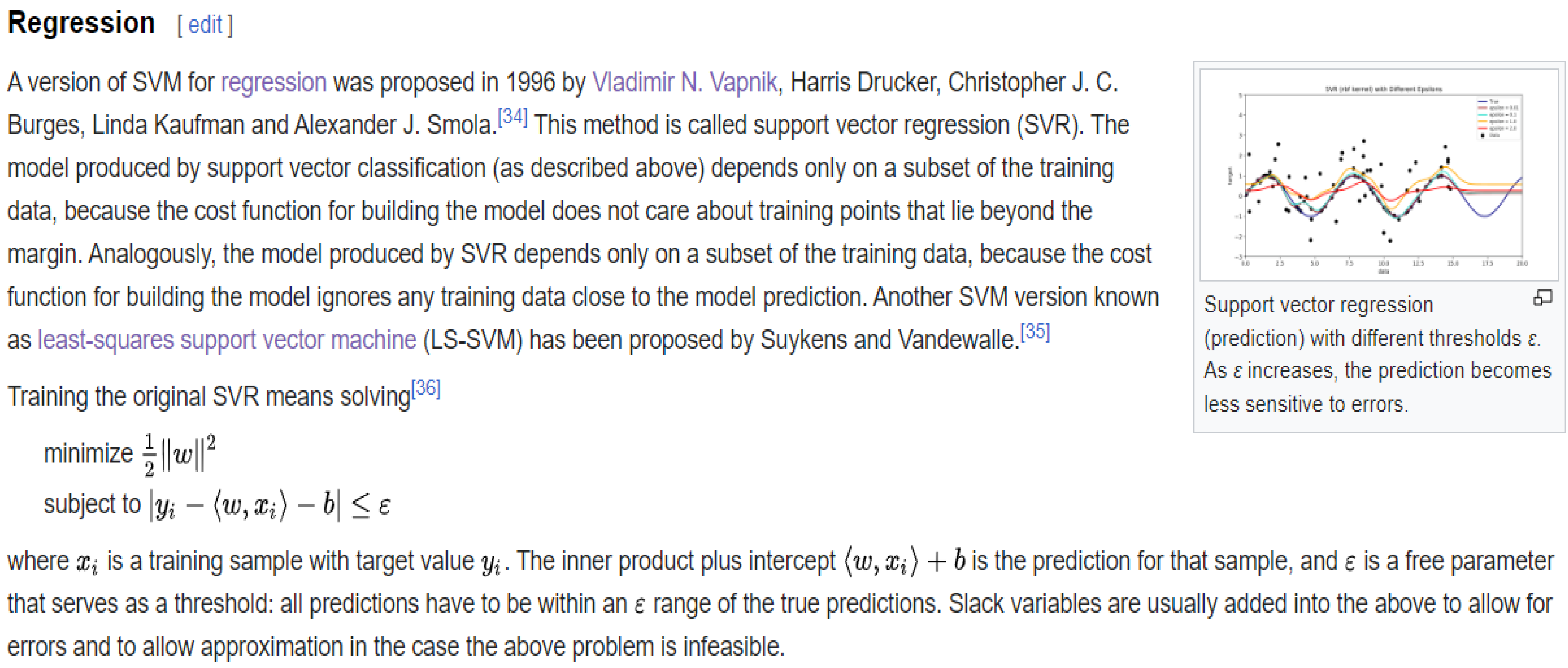


Separated by a Hyperplane

Demo code

https://prutor.ai/using-svm-to-perform-classification-on-a-non-linear-dataset/

SVM for Regression



Regression [edit]

A version of SVM for regression was proposed in 1996 by Vladimir N. Vapnik, Harris Drucker, Christopher J. C. Burges, Linda Kaufman and Alexander J. Smola.[34] This method is called support vector regression (SVR). The model produced by support vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction. Another SVM version known as least-squares support vector machine (LS-SVM) has been proposed by Suykens and Vandewalle.[35]

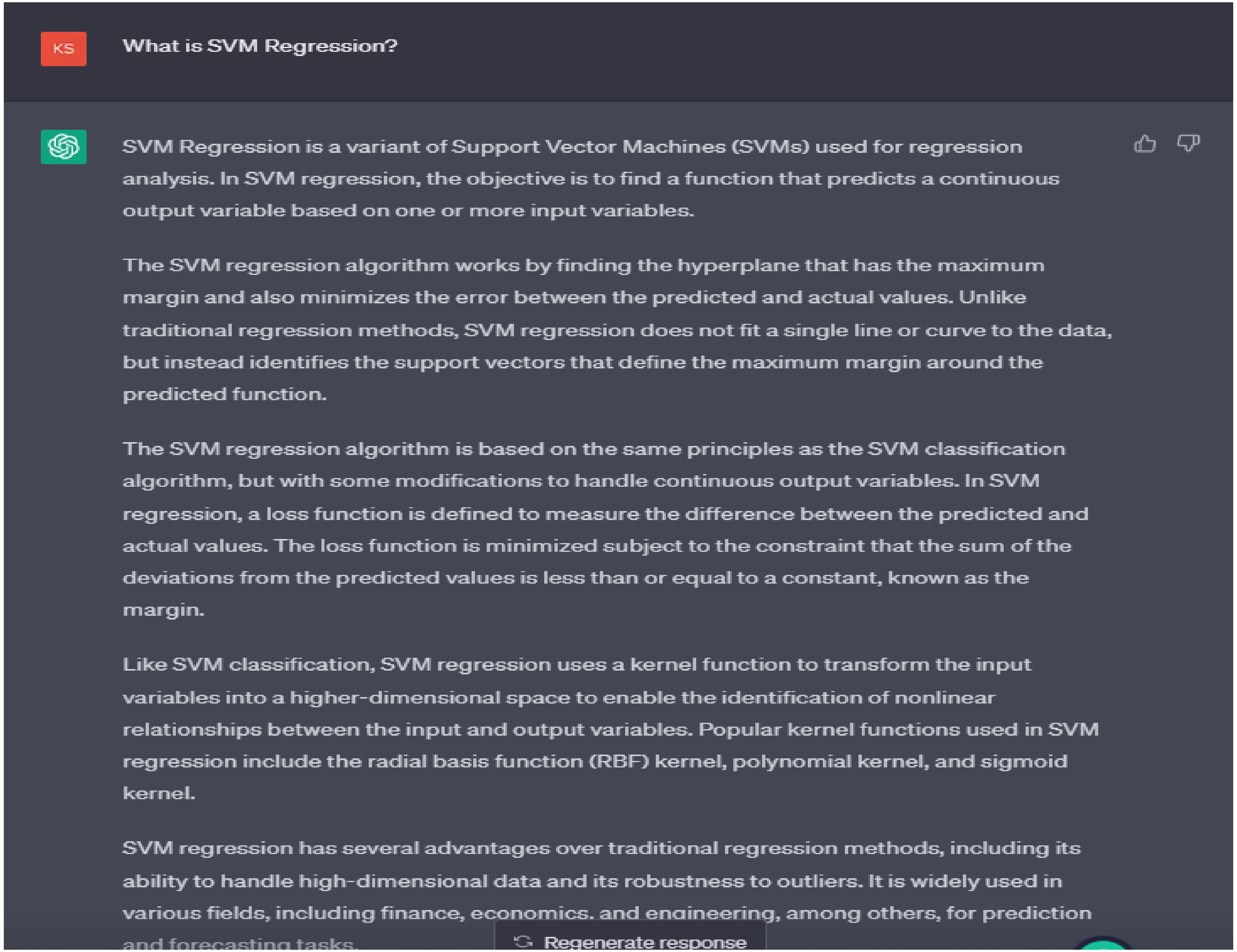
Support vector regression (prediction) with different thresholds ε. As ε increases, the prediction becomes less sensitive to errors.

Training the original SVR means solving[36]

$$\begin{aligned}&\text{minimize } \tfrac{1}{2}\|w\|^2\\ &\text{subject to } |y_i - \langle w, x_i\rangle - b| \le \varepsilon\end{aligned}$$

where x_i is a training sample with target value y_i. The inner product plus intercept $\langle w, x_i\rangle + b$ is the prediction for that sample, and ε is a free parameter that serves as a threshold: all predictions have to be within an ε range of the true predictions. Slack variables are usually added into the above to allow for errors and to allow approximation in the case the above problem is infeasible.

https://en.wikipedia.org/wiki/Support_vector_machine



What is SVM Regression?

SVM Regression is a variant of Support Vector Machines (SVMs) used for regression analysis. In SVM regression, the objective is to find a function that predicts a continuous output variable based on one or more input variables.

The SVM regression algorithm works by finding the hyperplane that has the maximum margin and also minimizes the error between the predicted and actual values. Unlike traditional regression methods, SVM regression does not fit a single line or curve to the data, but instead identifies the support vectors that define the maximum margin around the predicted function.

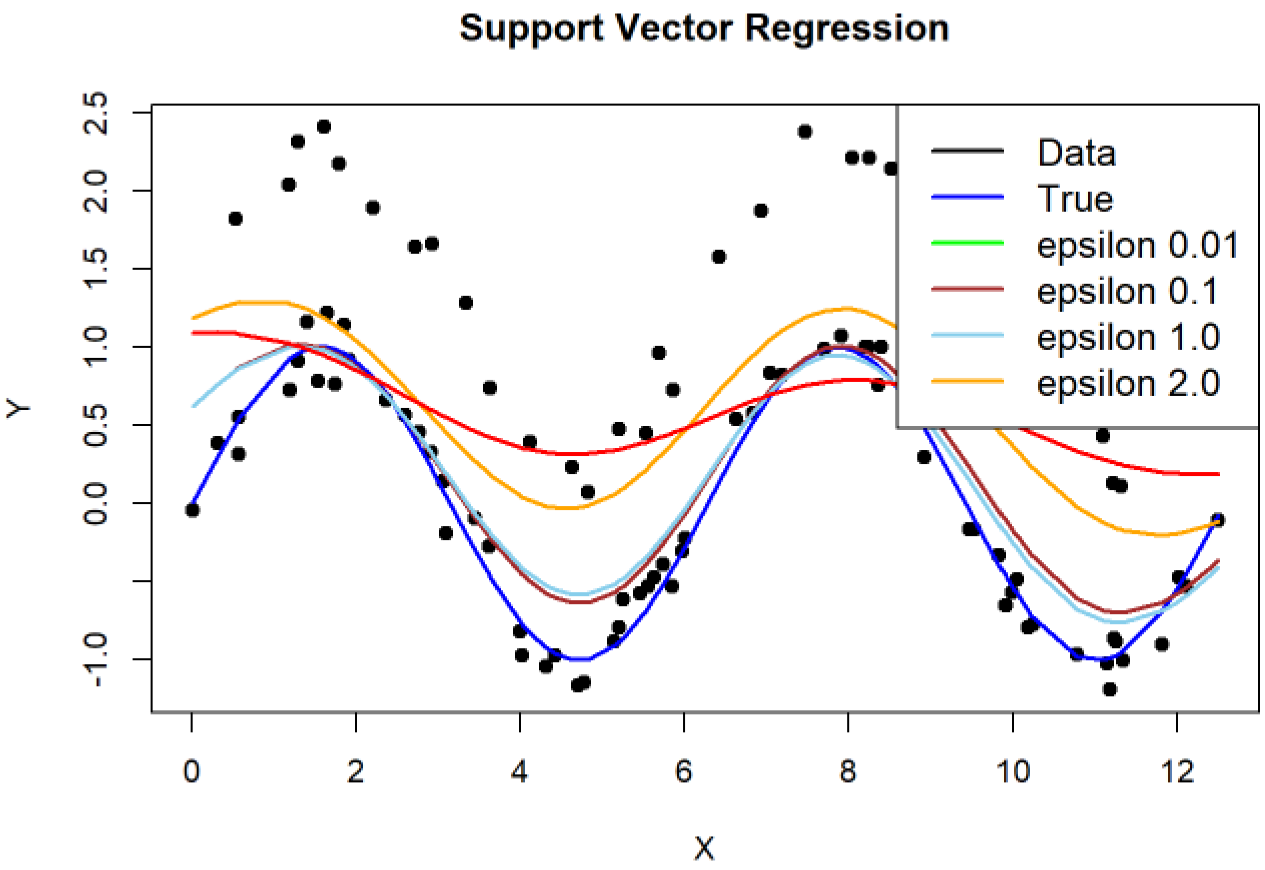
The SVM regression algorithm is based on the same principles as the SVM classification algorithm, but with some modifications to handle continuous output variables. In SVM regression, a loss function is defined to measure the difference between the predicted and actual values. The loss function is minimized subject to the constraint that the sum of the deviations from the predicted values is less than or equal to a constant, known as the margin.

Like SVM classification, SVM regression uses a kernel function to transform the input variables into a higher-dimensional space to enable the identification of nonlinear relationships between the input and output variables. Popular kernel functions used in SVM regression include the radial basis function (RBF) kernel, polynomial kernel, and sigmoid kernel.

SVM regression has several advantages over traditional regression methods, including its ability to handle high-dimensional data and its robustness to outliers. It is widely used in various fields, including finance, economics, and engineering, among others, for prediction and forecasting tasks.

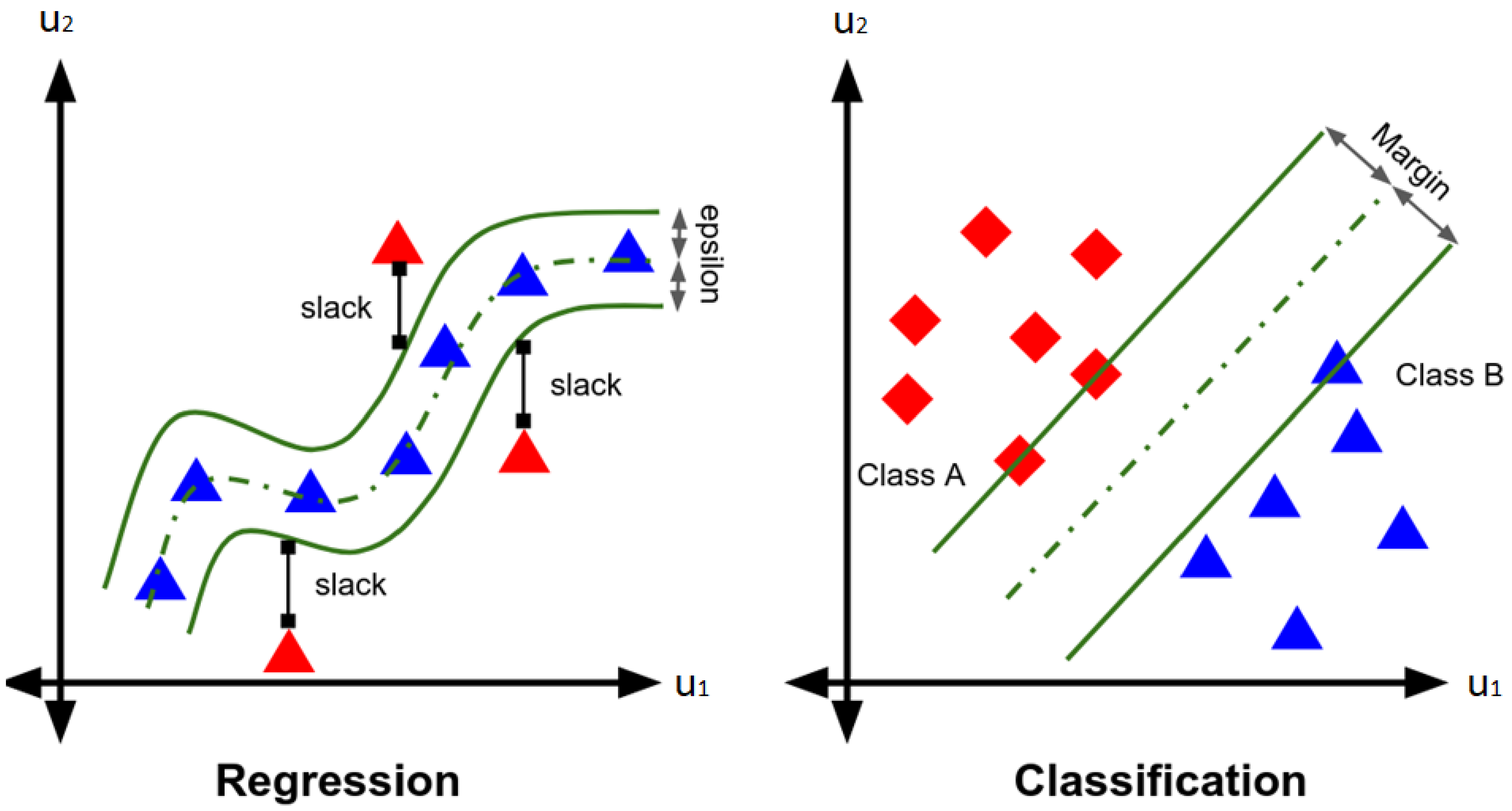
https://chat.openai.com/

SVM for Regression



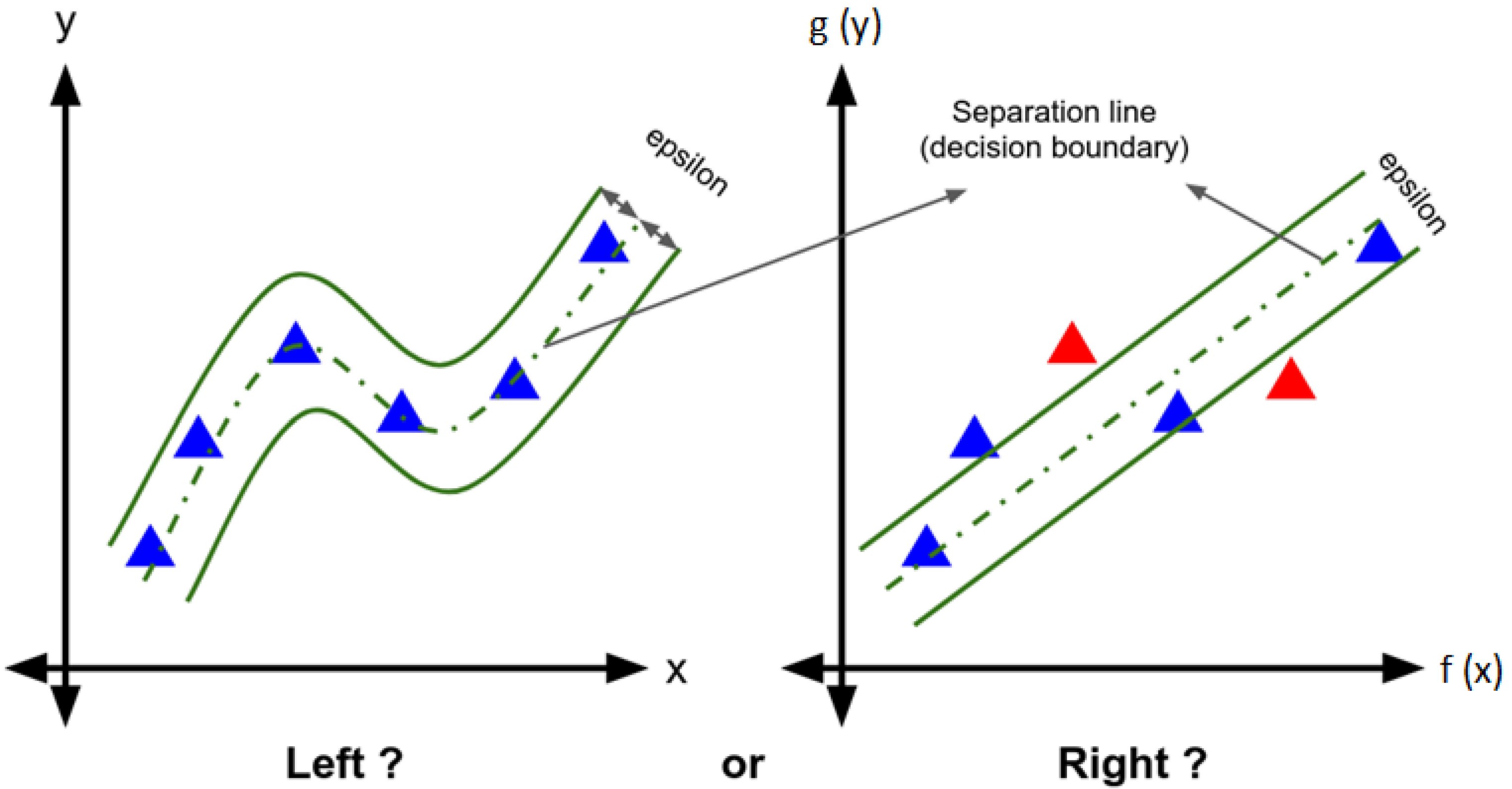
Demo code

SVM Regression



https://medium.com/it-paragon/support-vector-machine-regression-cf65348b6345

SVM Regression



Both of them are correct. The right one tolerates some outlier points but the left one is trying to achieve zero tolerance with perfect regression.

https://medium.com/it-paragon/support-vector-machine-regression-cf65348b6345

SVM Regression

Regularization

- The regularization parameter (C parameter in Python’s sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training sample.

- C *parameter balances the trade-off between the model complexity and empirical error*. *To simplify*, when C is large, the SVM tends to be overfitting while when C is small, the SVM tends to be underfitting.

https://medium.com/it-paragon/support-vector-machine-regression-cf65348b6345

SVM Regression

Gamma

- The gamma parameter defines how far the influence of a single training example reaches (low values mean far and a high value means close). With low gamma, points far away from plausible separation lines are considered in the calculation for the separation line. Whereas high gamma means the points close to a plausible line are considered in the calculation.

- When γ is large, the SVM tends to be overfitting. On the other hand, when γ is small, the SVM tends to be underfitting.

https://medium.com/it-paragon/support-vector-machine-regression-cf65348b6345

SVM Regression

Epsilon

- In the same way, as with a classification approach, there is a motivation to seek and optimize the generalization bounds given for regression. They relied on defining the loss function that ignores errors, which are situated within a certain distance of the true value. This type of function is often called ***epsilon intensive loss function***.

- *ε-intensive loss function affects the smoothness of the SVM's response.*

https://medium.com/it-paragon/support-vector-machine-regression-cf65348b6345

Nonlinear SVM: optimization

Formulation: (Lagrangian dual problem)

$$\text{maximize} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

such that

$$0 \le \alpha_i \le C$$

$$\sum\nolimits_{i=1}^{n} \alpha_i y_i = 0$$

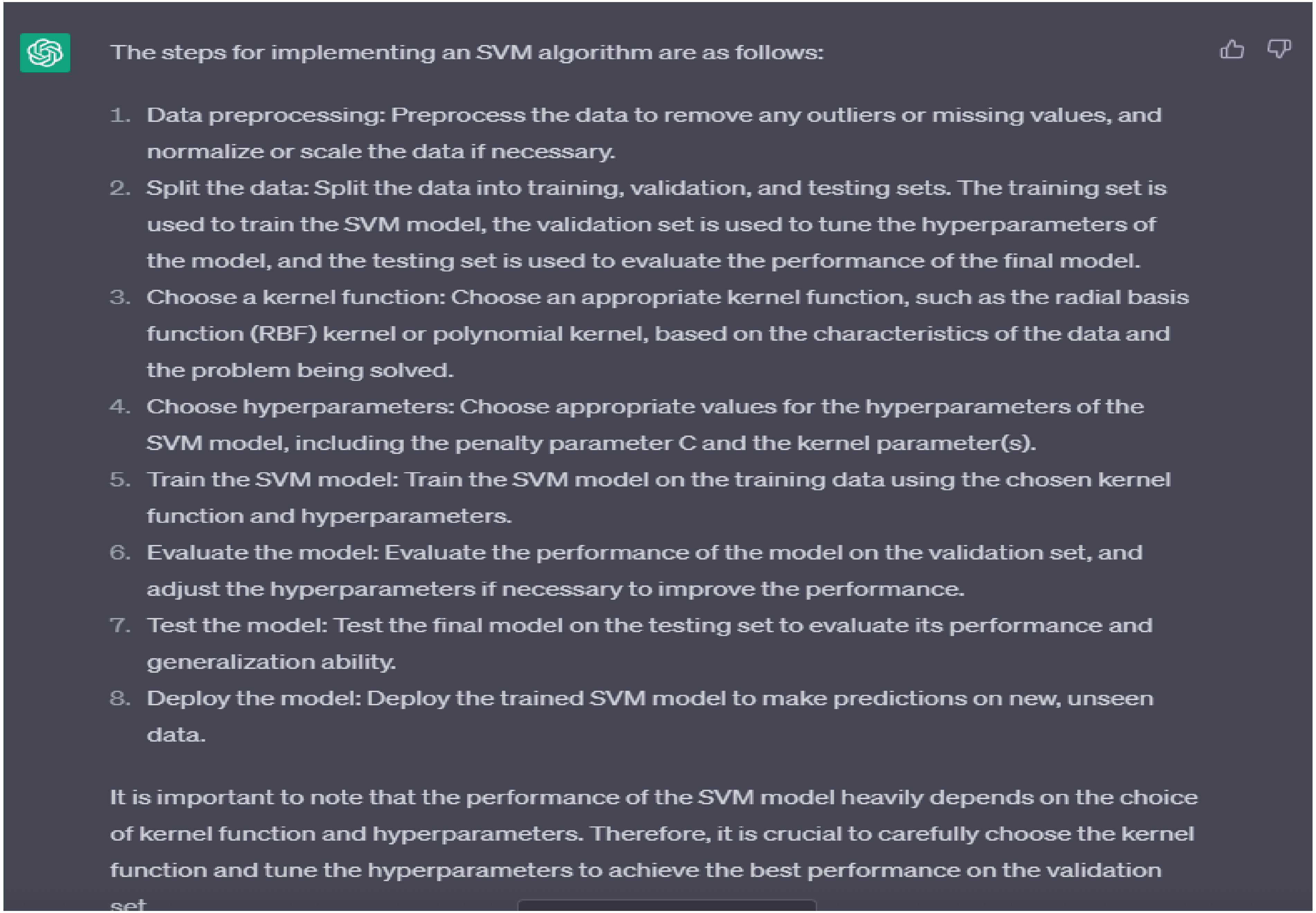
The solution of the discriminant function is

$$g(\boldsymbol{x}) = \sum\nolimits_{i \in \mathrm{SV}} \alpha_i \, y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

The optimization technique is the same.

SVM: algorithm

1. Choose a kernel function
2. Choose a value for C
3. Solve the quadratic programming problem (many software packages available)
4. Construct the discriminant function from the support vectors



The steps for implementing an SVM algorithm are as follows:

1. Data preprocessing: Preprocess the data to remove any outliers or missing values, and normalize or scale the data if necessary.
2. Split the data: Split the data into training, validation, and testing sets. The training set is used to train the SVM model, the validation set is used to tune the hyperparameters of the model, and the testing set is used to evaluate the performance of the final model.
3. Choose a kernel function: Choose an appropriate kernel function, such as the radial basis function (RBF) kernel or polynomial kernel, based on the characteristics of the data and the problem being solved.
4. Choose hyperparameters: Choose appropriate values for the hyperparameters of the SVM model, including the penalty parameter C and the kernel parameter(s).
5. Train the SVM model: Train the SVM model on the training data using the chosen kernel function and hyperparameters.
6. Evaluate the model: Evaluate the performance of the model on the validation set, and adjust the hyperparameters if necessary to improve the performance.
7. Test the model: Test the final model on the testing set to evaluate its performance and generalization ability.
8. Deploy the model: Deploy the trained SVM model to make predictions on new, unseen data.

It is important to note that the performance of the SVM model heavily depends on the choice of kernel function and hyperparameters. Therefore, it is crucial to carefully choose the kernel function and tune the hyperparameters to achieve the best performance on the validation set

https://chat.openai.com/

Some issues

Choice of kernel

1. Gaussian or polynomial kernel is the default
2. if ineffective, more elaborate kernels are needed
3. domain experts can give assistance in formulating appropriate similarity measures

Choice of kernel parameters

1. e.g., σ in Gaussian kernel
2. σ is the distance between closest points with different classifications
3. In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

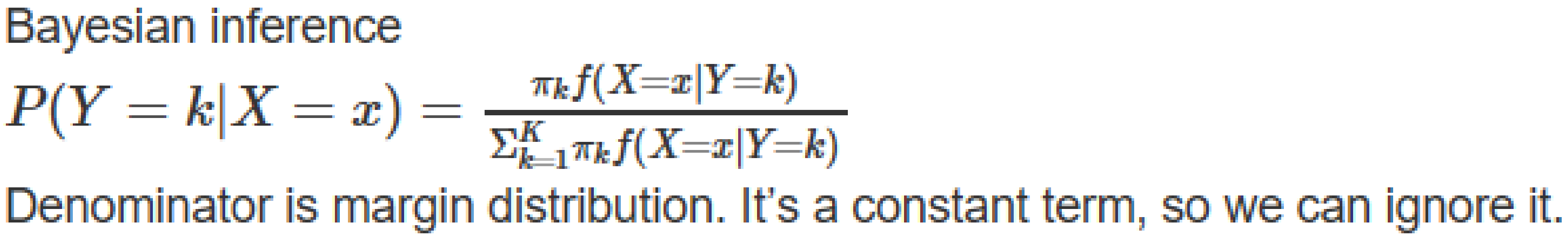
Optimization criterion – Hard margin v.s. Soft margin

a lengthy series of experiments in which various parameters are tested

Scaling before applying SVM is very important.

1. The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges.
2. Another advantage is to avoid numerical difficulties during the calculation.
3. Of course we have to use the same method to scale both training and testing data.

Discriminant Analysis

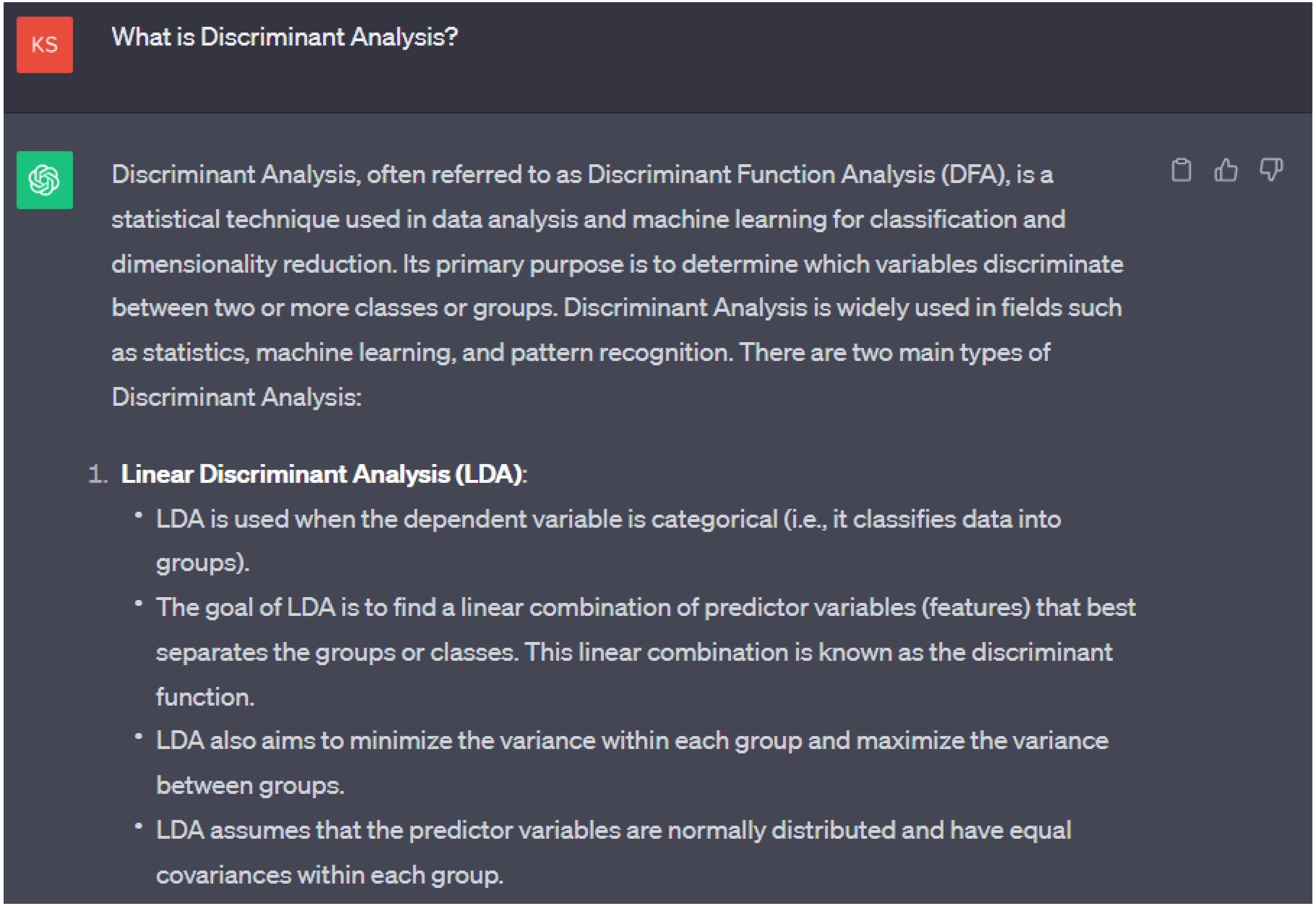


Bayesian inference

$$P(Y = k|X = x) = \frac{\pi_k f(X{=}x|Y{=}k)}{\Sigma_{k=1}^{K} \pi_k f(X{=}x|Y{=}k)}$$

Denominator is margin distribution. It's a constant term, so we can ignore it.

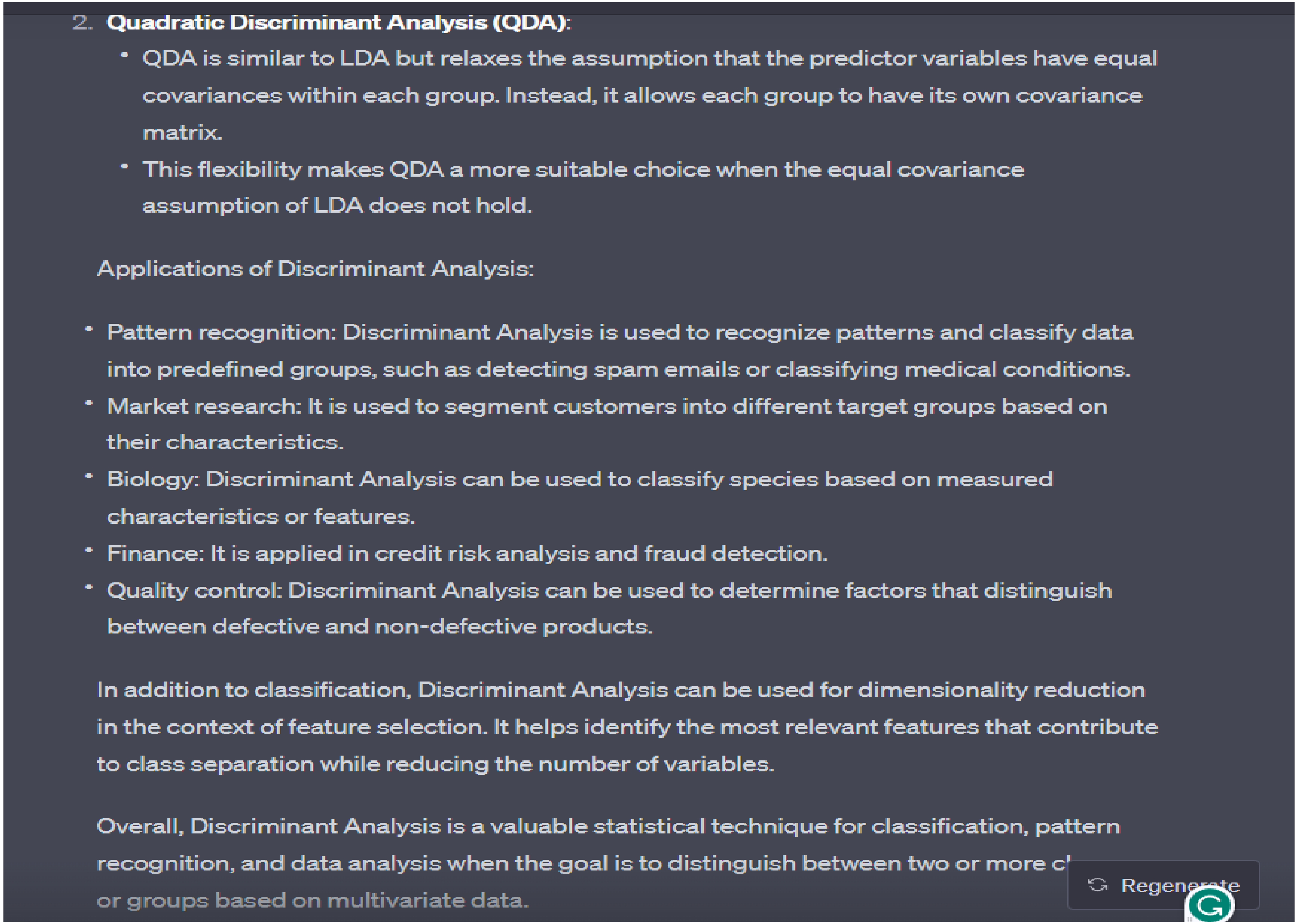
https://en.wikipedia.org/wiki/Linear_discriminant_analysis



What is Discriminant Analysis?

Discriminant Analysis, often referred to as Discriminant Function Analysis (DFA), is a statistical technique used in data analysis and machine learning for classification and dimensionality reduction. Its primary purpose is to determine which variables discriminate between two or more classes or groups. Discriminant Analysis is widely used in fields such as statistics, machine learning, and pattern recognition. There are two main types of Discriminant Analysis:

1. **Linear Discriminant Analysis (LDA):**
    - LDA is used when the dependent variable is categorical (i.e., it classifies data into groups).
    - The goal of LDA is to find a linear combination of predictor variables (features) that best separates the groups or classes. This linear combination is known as the discriminant function.
    - LDA also aims to minimize the variance within each group and maximize the variance between groups.
    - LDA assumes that the predictor variables are normally distributed and have equal covariances within each group.



2. **Quadratic Discriminant Analysis (QDA):**
    - QDA is similar to LDA but relaxes the assumption that the predictor variables have equal covariances within each group. Instead, it allows each group to have its own covariance matrix.
    - This flexibility makes QDA a more suitable choice when the equal covariance assumption of LDA does not hold.

Applications of Discriminant Analysis:

- Pattern recognition: Discriminant Analysis is used to recognize patterns and classify data into predefined groups, such as detecting spam emails or classifying medical conditions.
- Market research: It is used to segment customers into different target groups based on their characteristics.
- Biology: Discriminant Analysis can be used to classify species based on measured characteristics or features.
- Finance: It is applied in credit risk analysis and fraud detection.
- Quality control: Discriminant Analysis can be used to determine factors that distinguish between defective and non-defective products.

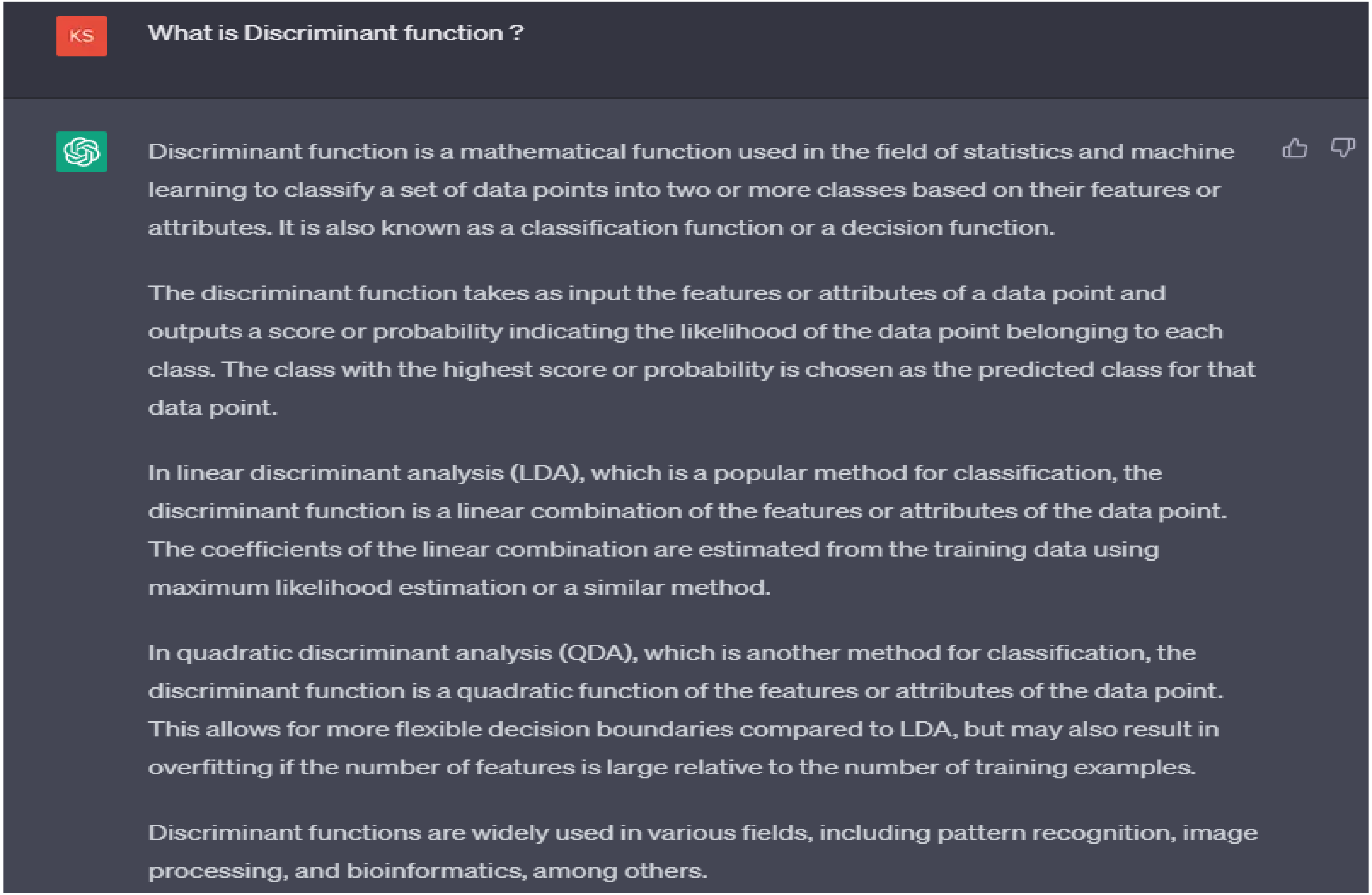
In addition to classification, Discriminant Analysis can be used for dimensionality reduction in the context of feature selection. It helps identify the most relevant features that contribute to class separation while reducing the number of variables.

Overall, Discriminant Analysis is a valuable statistical technique for classification, pattern recognition, and data analysis when the goal is to distinguish between two or more c
or groups based on multivariate data.

Discriminant function

- When K classes, the classifier is said to assign a feature vector $\boldsymbol{x}$ to class π_i if
  $$g_i(\boldsymbol{x}) > g_j(\boldsymbol{x}) \qquad \text{for all } j \neq i \in \{1, \cdots, K\}$$
- For two-class case, $g(\boldsymbol{x}) \equiv g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$
  Decide π_1 if $g(\boldsymbol{x}) > 0$; otherwise decide π_2
- An example we've learned before:

  LDA, QDA for equal priors and misclassification cost:
  $$g(\boldsymbol{x}) \equiv f_1(\boldsymbol{x}) - f_2(\boldsymbol{x})$$
  where $f_i(\cdot)$: probability density functions for π_i



What is Discriminant function ?

Discriminant function is a mathematical function used in the field of statistics and machine learning to classify a set of data points into two or more classes based on their features or attributes. It is also known as a classification function or a decision function.

The discriminant function takes as input the features or attributes of a data point and outputs a score or probability indicating the likelihood of the data point belonging to each class. The class with the highest score or probability is chosen as the predicted class for that data point.

In linear discriminant analysis (LDA), which is a popular method for classification, the discriminant function is a linear combination of the features or attributes of the data point. The coefficients of the linear combination are estimated from the training data using maximum likelihood estimation or a similar method.

In quadratic discriminant analysis (QDA), which is another method for classification, the discriminant function is a quadratic function of the features or attributes of the data point. This allows for more flexible decision boundaries compared to LDA, but may also result in overfitting if the number of features is large relative to the number of training examples.

Discriminant functions are widely used in various fields, including pattern recognition, image processing, and bioinformatics, among others.

https://chat.openai.com/

Review of some vector algebra

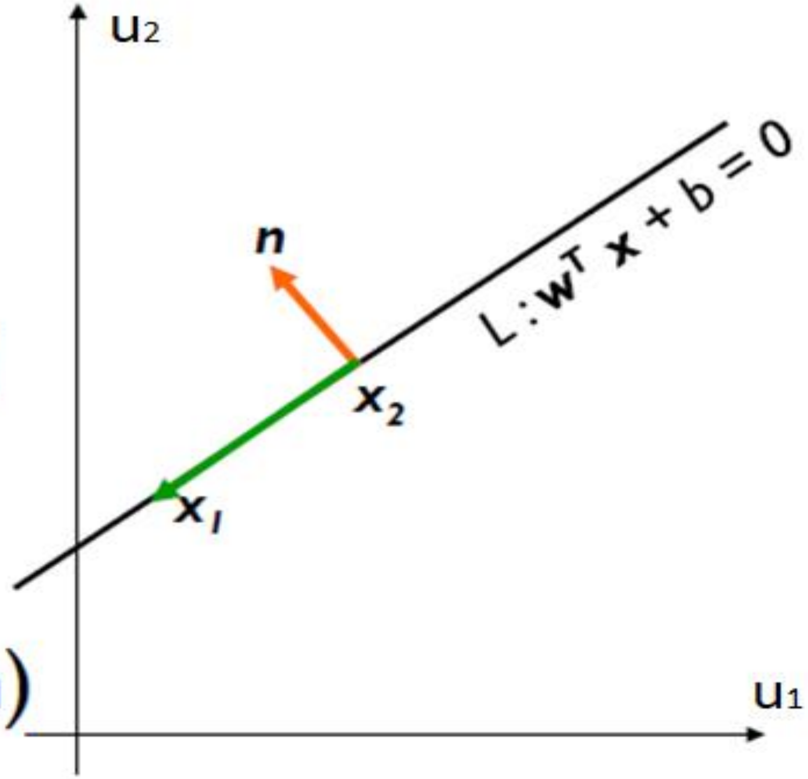
A hyper-plane L in the feature space defined by

$$g(\boldsymbol{x}) = \mathbf{w}^T \boldsymbol{x} + b = 0$$

For any two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ lying in L,

$\mathbf{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0$, hence

$\boldsymbol{n} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$ is the (unit-length) normal vector of L



Review of some vector algebra

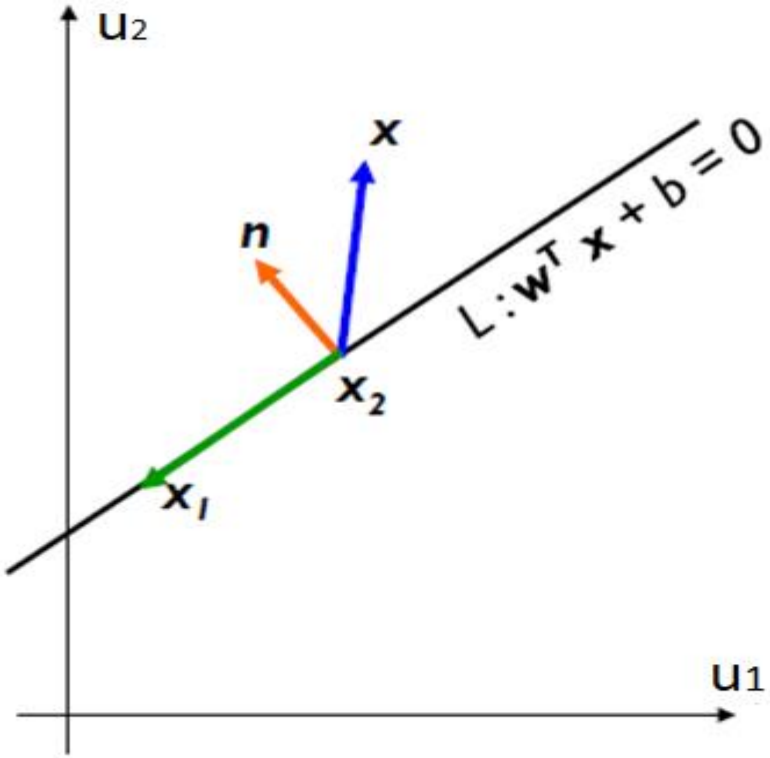
For any point $\boldsymbol{x}_2$ in L,
$\boldsymbol{w}^T\boldsymbol{x}_2 = -b$

The signed distance of any point $\boldsymbol{x}$ to L is given by

$$\boldsymbol{n}^T(\boldsymbol{x} - \boldsymbol{x}_2) = \frac{1}{\| \boldsymbol{w} \|}(\boldsymbol{w}^T\boldsymbol{x} + b)$$

$$= \frac{1}{\| \boldsymbol{w} \|} g(\boldsymbol{x})$$

Hence $g(\boldsymbol{x})$ is proportional to the signed distance from $\boldsymbol{x}$ to the hyperplane defined by $g(\boldsymbol{x}) = 0$



Linear discriminant function

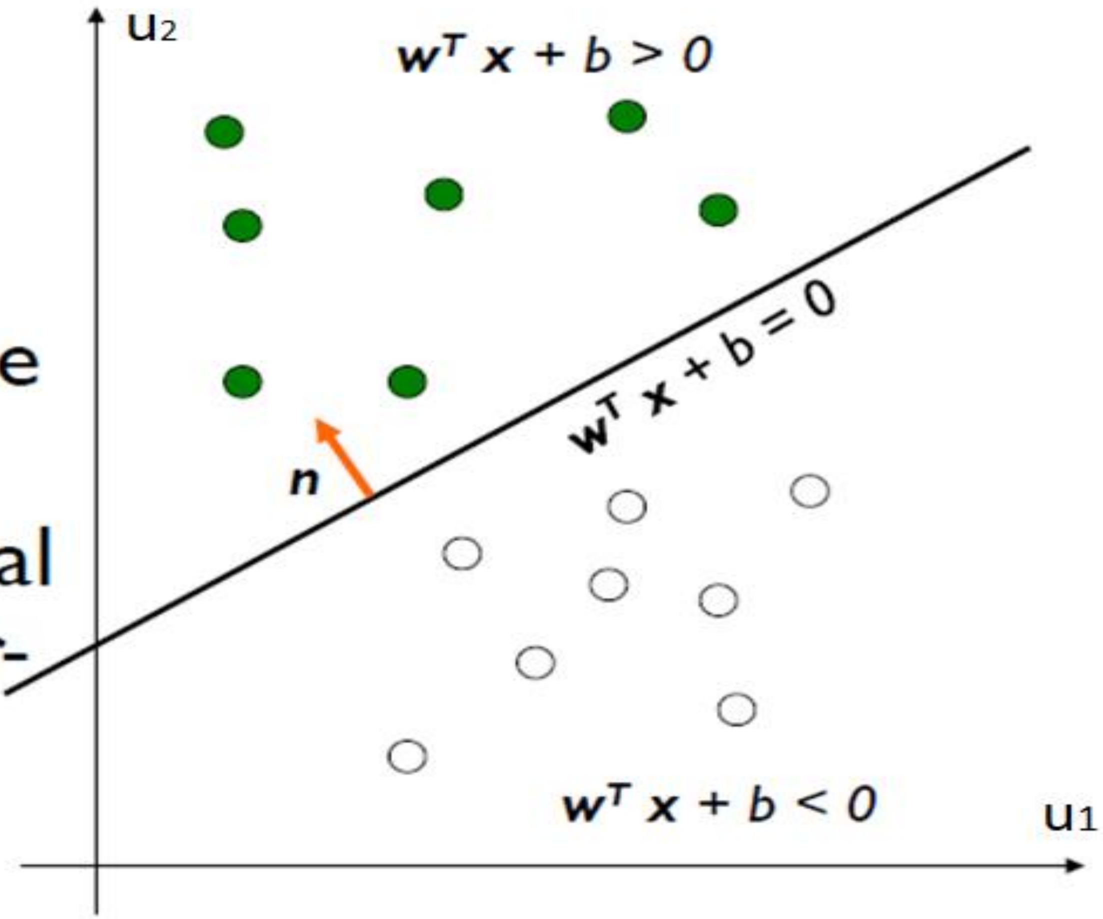
$g(\boldsymbol{x})$ is a linear function:

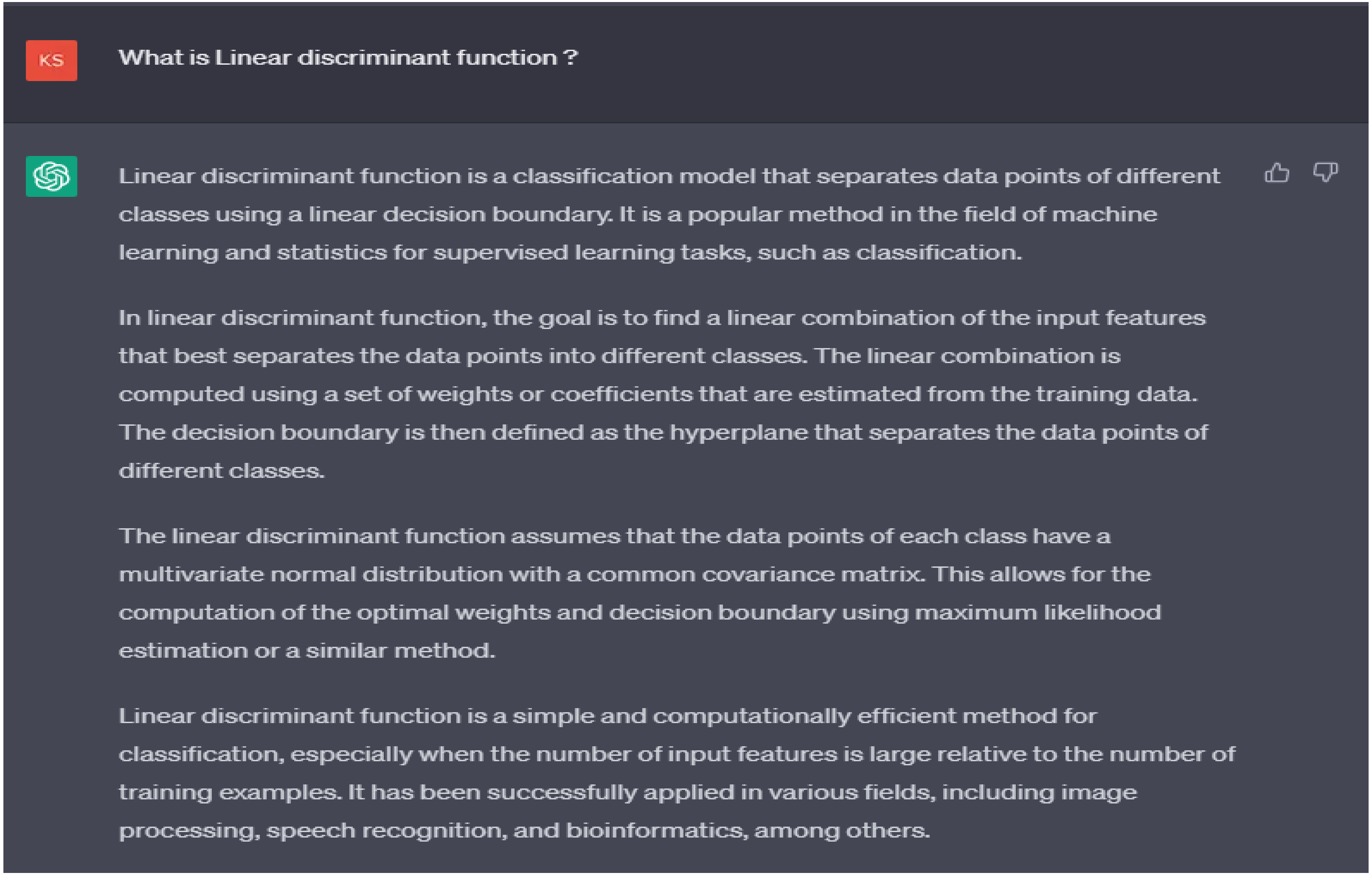
$$g(\boldsymbol{x}) = \mathbf{w}^T \boldsymbol{x} + b$$

A hyper-plane in the feature space

(Unit-length) normal vector of the hyper-plane:

$$\boldsymbol{n} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$


u2
$\mathbf{w}^T \mathbf{x} + b > 0$
$\mathbf{w}^T \mathbf{x} + b = 0$
$\boldsymbol{n}$
$\mathbf{w}^T \mathbf{x} + b < 0$
u1



What is Linear discriminant function ?

Linear discriminant function is a classification model that separates data points of different classes using a linear decision boundary. It is a popular method in the field of machine learning and statistics for supervised learning tasks, such as classification.

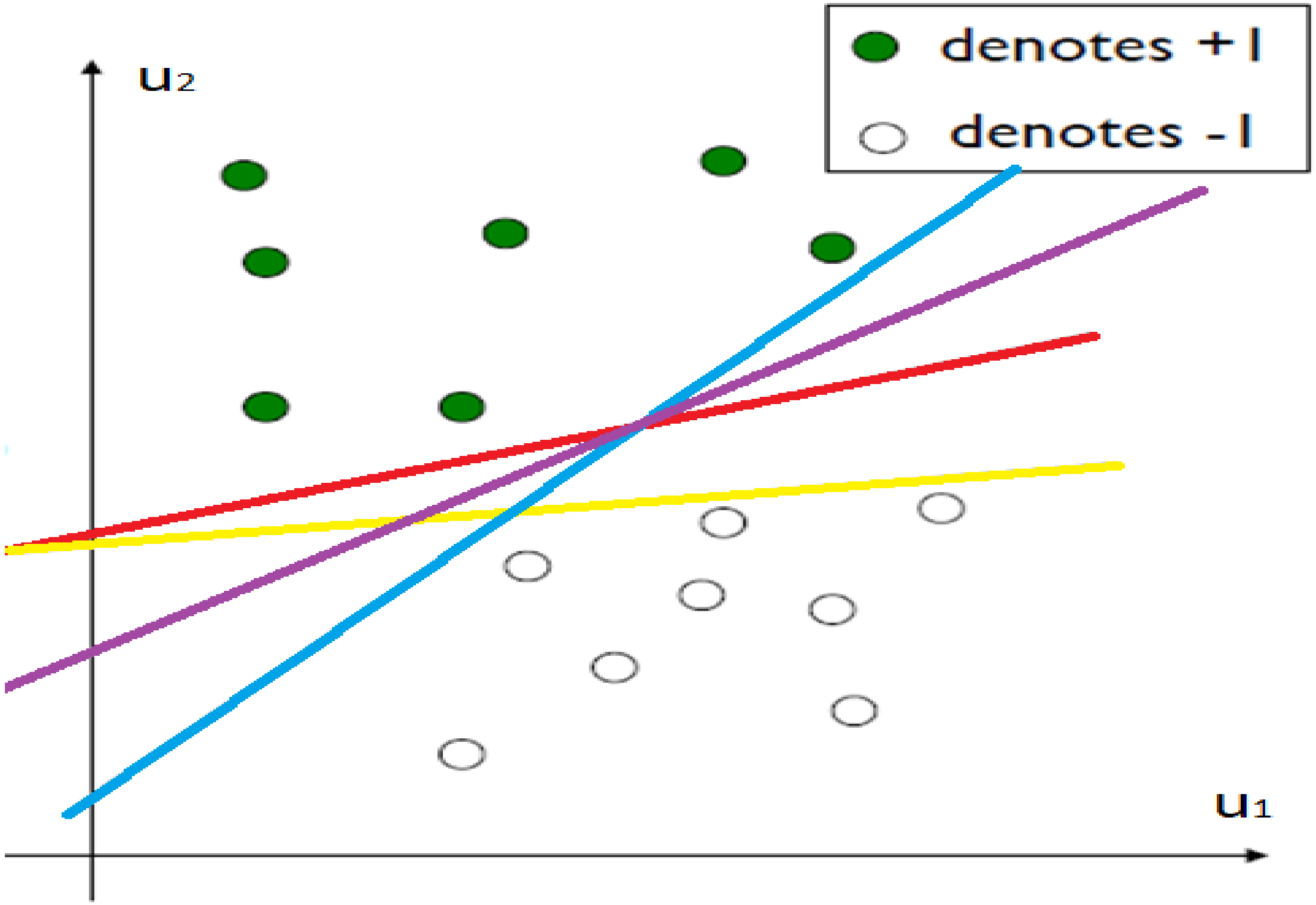
In linear discriminant function, the goal is to find a linear combination of the input features that best separates the data points into different classes. The linear combination is computed using a set of weights or coefficients that are estimated from the training data. The decision boundary is then defined as the hyperplane that separates the data points of different classes.

The linear discriminant function assumes that the data points of each class have a multivariate normal distribution with a common covariance matrix. This allows for the computation of the optimal weights and decision boundary using maximum likelihood estimation or a similar method.

Linear discriminant function is a simple and computationally efficient method for classification, especially when the number of input features is large relative to the number of training examples. It has been successfully applied in various fields, including image processing, speech recognition, and bioinformatics, among others.

https://chat.openai.com/

The question is, there are infinite linear discriminant function, which one is best?



LDA vs. QDA

QDA

If f(X=x|Y=k) are $N(\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})$

$$f(X=x|Y=k) = f(x|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) = (2\pi)^{-0.5d}|\boldsymbol{\Sigma_k}|^{-0.5}e^{-0.5(\boldsymbol{x}-\boldsymbol{\mu_k})^T\boldsymbol{\Sigma_k^{-1}}(\boldsymbol{x}-\boldsymbol{\mu_k})}$$

$$\pi_k f(x|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) = \pi_k(2\pi)^{-0.5d}|\boldsymbol{\Sigma_k}|^{-0.5}e^{-0.5(\boldsymbol{x}-\boldsymbol{\mu_k})^T\boldsymbol{\Sigma_k^{-1}}(\boldsymbol{x}-\boldsymbol{\mu_k})}$$

$$log\pi_k f(x|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) = log\pi_k - 0.5log|\boldsymbol{\Sigma_k}| - 0.5(\boldsymbol{x}-\boldsymbol{\mu_k})^T\boldsymbol{\Sigma_k^{-1}}(\boldsymbol{x}-\boldsymbol{\mu_k})$$

$$logP(Y=k|X=x) \sim log(\pi_k f(X=x|Y=k))$$

To see which probability of class is the largest, that is that class.

https://en.wikipedia.org/wiki/Quadratic_classifier

LDA

If $\boldsymbol{\Sigma_k} = \boldsymbol{\Sigma}$

$$log\pi_k f(X=x|Y=k) = log\pi_k - 0.5(\boldsymbol{x}-\boldsymbol{\mu_k})^T\boldsymbol{\Sigma^{-1}}(\boldsymbol{x}-\boldsymbol{\mu_k})$$

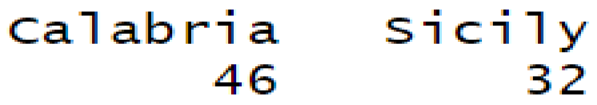
https://en.wikipedia.org/wiki/Linear_discriminant_analysis

Demo: LDA, QDA, Logistic Regression, Naïve Bayes, SVM

Binary logistic regression

Number of categories

```
table(xytrain$region)[c(3,8)]
```



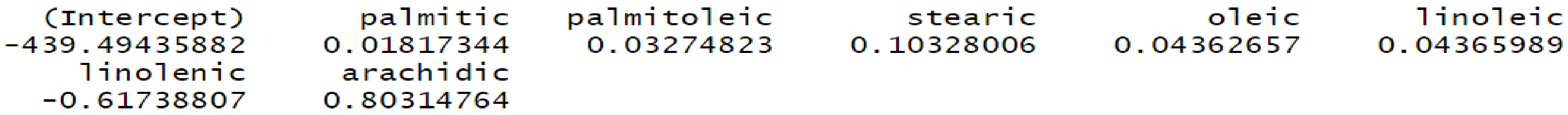
```
 Calabria    Sicily
       46        32
```

Model

```
mod2<-glm(formula=region~.,data=xytrain,family=binomial(link="logit"))
```

Coefficient

```
mod2$coefficients
```

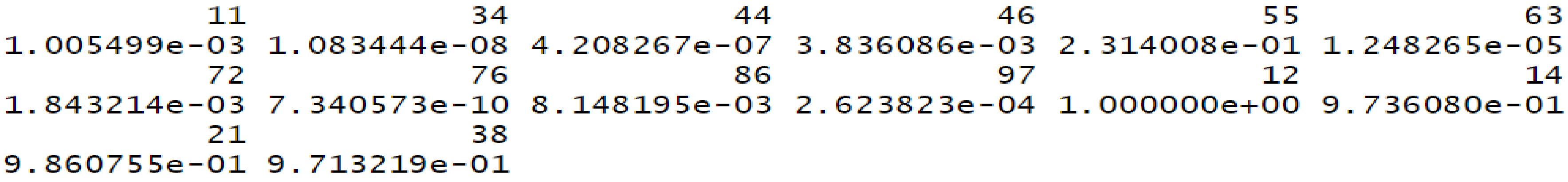


```
  (Intercept)      palmitic   palmitoleic       stearic         oleic      linoleic 
-439.49435882    0.01817344    0.03274823    0.10328006    0.04362657    0.04365989 
    linolenic     arachidic 
  -0.61738807    0.80314764 
```

Binary logistic regression

Predict probability

prob=predict(mod2,xytest,type="response");prob



```
           11            34            44            46            55            63 
1.005499e-03 1.083444e-08 4.208267e-07 3.836086e-03 2.314008e-01 1.248265e-05 
           72            76            86            97            12            14 
1.843214e-03 7.340573e-10 8.148195e-03 2.623823e-04 1.000000e+00 9.736080e-01 
           21            38 
9.860755e-01 9.713219e-01 
```

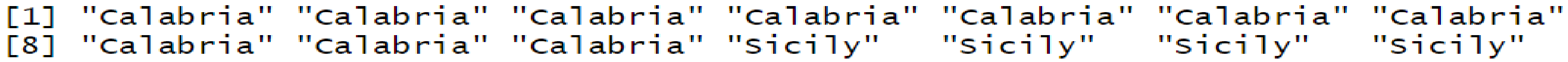
Probability converted to category

label=rep('',length(prob))

label[prob<0.5]='Calabria'

label[prob>=0.5]='Sicily'

label

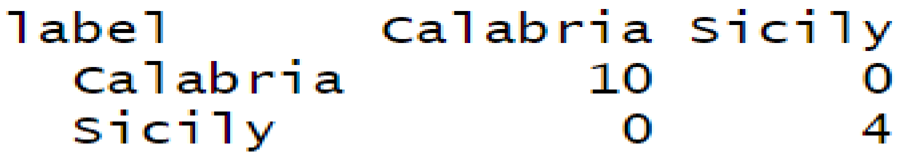


```
[1] "Calabria" "Calabria" "Calabria" "Calabria" "Calabria" "Calabria" "Calabria"
[8] "Calabria" "Calabria" "Calabria" "Sicily"   "Sicily"   "Sicily"   "Sicily"  
```

Binary logistic regression

Confusion matrix

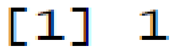
xytest$region=as.character(xytest$region)
tab=table(label,xytest$region);tab



```
label      Calabria Sicily
  Calabria       10      0
  Sicily          0      4
```

Accuracy

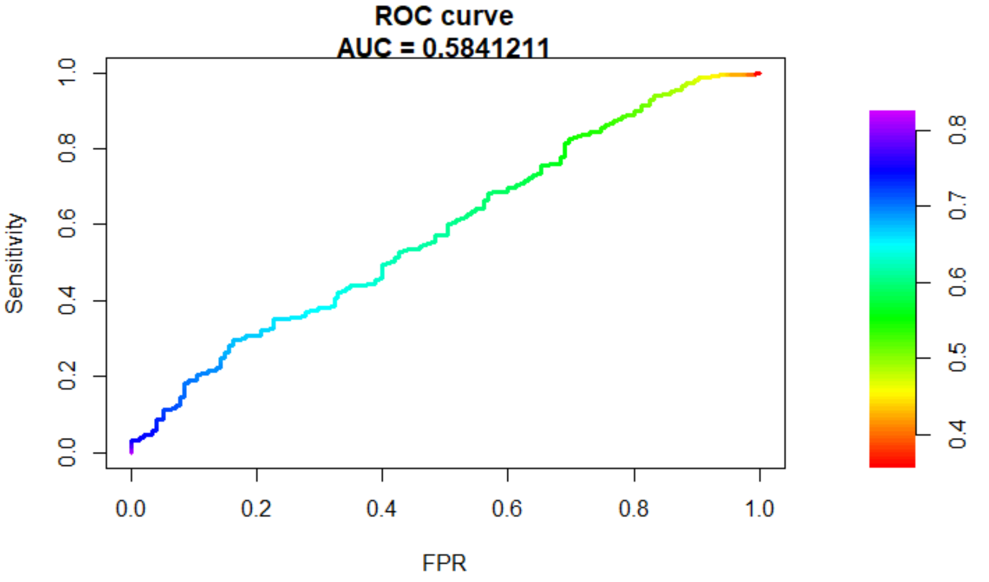
sum(diag(tab))/sum(tab)



```
[1] 1
```

Plot ROC Curve

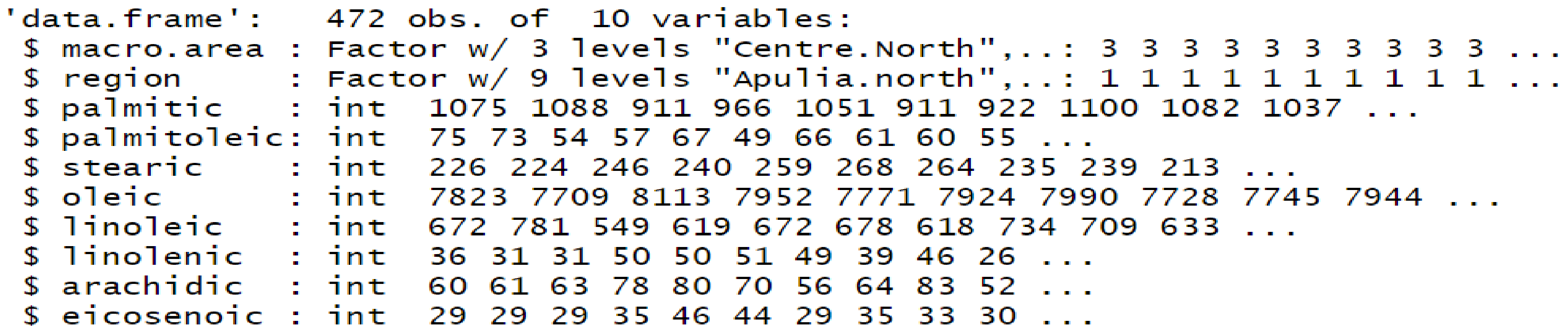
```
#install.packages('PRROC')
library(PRROC)
credit = read.csv('Credit.csv')
table(credit$Married)
credit$Married = as.factor(credit$Married)
mod3<-
glm(formula=Married~Income+Limit+Rating+Cards+Age+Educ
ation+Balance,data=credit,family=binomial(link="logit"))
prob=predict(mod3,credit,type="response");prob
PRROC_obj=roc.curve(scores.class0=prob,weights.class0=as.nu
meric(as.factor(credit$Married))-1,curve=TRUE)
plot(PRROC_obj)
```



Multiple logistic regression

Observation data

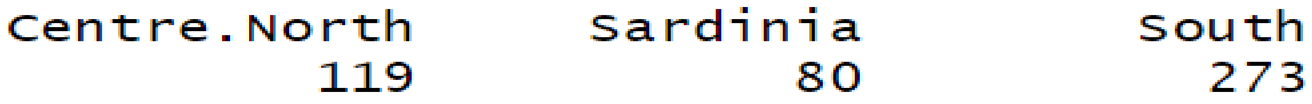
str(train)



```
'data.frame':   472 obs. of  10 variables:
 $ macro.area : Factor w/ 3 levels "Centre.North",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ region     : Factor w/ 9 levels "Apulia.north",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ palmitic   : int  1075 1088 911 966 1051 911 922 1100 1082 1037 ...
 $ palmitoleic: int  75 73 54 57 67 49 66 61 60 55 ...
 $ stearic    : int  226 224 246 240 259 268 264 235 239 213 ...
 $ oleic      : int  7823 7709 8113 7952 7771 7924 7990 7728 7745 7944 ...
 $ linoleic   : int  672 781 549 619 672 678 618 734 709 633 ...
 $ linolenic  : int  36 31 31 50 50 51 49 39 46 26 ...
 $ arachidic  : int  60 61 63 78 80 70 56 64 83 52 ...
 $ eicosenoic : int  29 29 29 35 46 44 29 35 33 30 ...
```

Number of categories

table(train$macro.area)



```
Centre.North     Sardinia        South
         119           80          273
```

Multiple logistic regression

Package

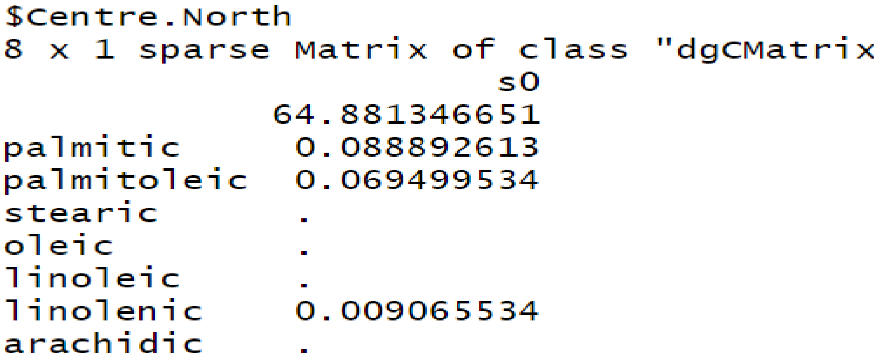
```
#install.packages('glmnet')
library(glmnet)
```

Build model

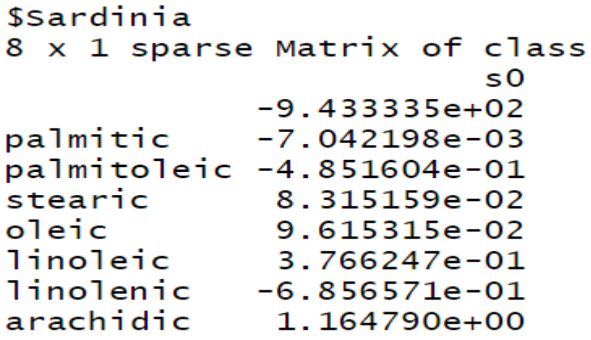
```
mod1 = glmnet(xtrain,ytrain,family="multinomial",lambda=0)
```

Coefficient

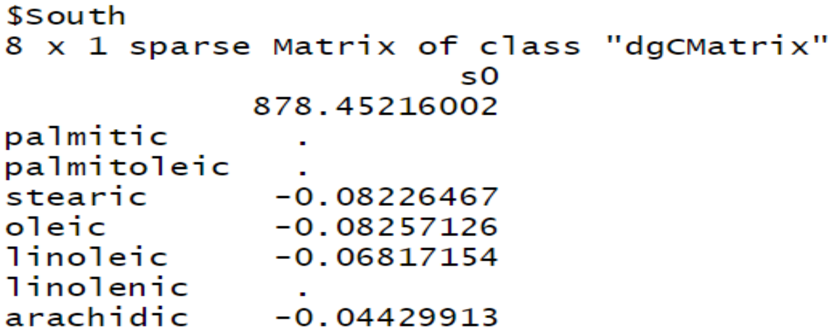
```
coef(mod1)
```



```
$Centre.North
8 x 1 sparse Matrix of class "dgCMatrix
                       s0
             64.881346651
palmitic      0.088892613
palmitoleic   0.069499534
stearic       .
oleic         .
linoleic      .
linolenic     0.009065534
arachidic     .
```



```
$Sardinia
8 x 1 sparse Matrix of class
                        s0
            -9.433335e+02
palmitic    -7.042198e-03
palmitoleic -4.851604e-01
stearic      8.315159e-02
oleic        9.615315e-02
linoleic     3.766247e-01
linolenic   -6.856571e-01
arachidic    1.164790e+00
```

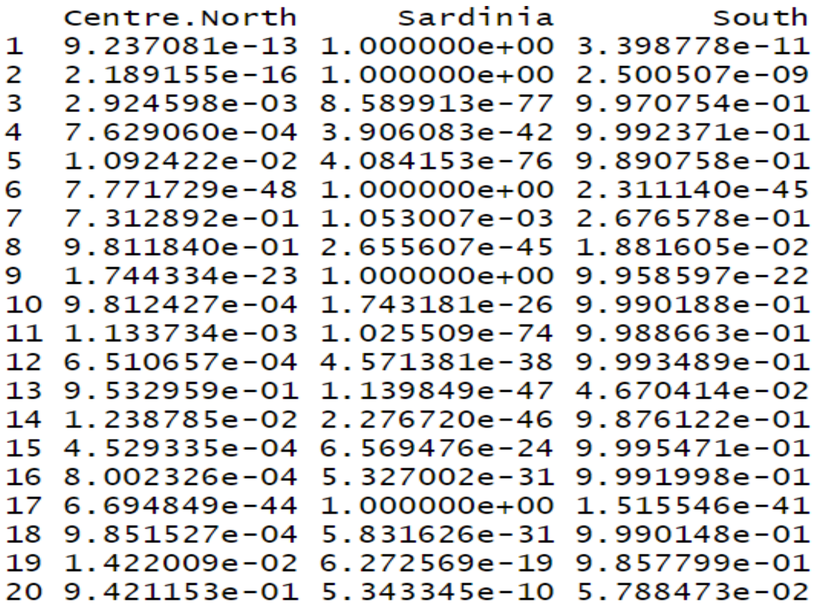


```
$South
8 x 1 sparse Matrix of class "dgCMatrix"
                       s0
             878.45216002
palmitic      .
palmitoleic   .
stearic      -0.08226467
oleic        -0.08257126
linoleic     -0.06817154
linolenic     .
arachidic    -0.04429913
```

Multiple logistic regression

Predict probability

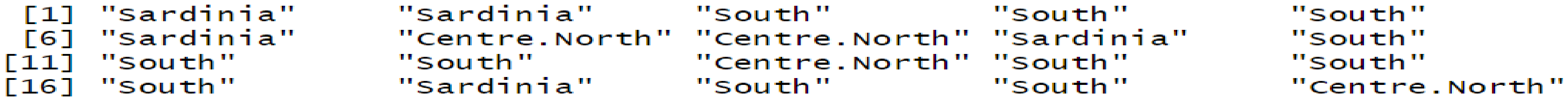
prob=predict(mod1,xtest,type="response");prob[1:20,1:3,1]



```
   Centre.North     Sardinia        South
1  9.237081e-13 1.000000e+00 3.398778e-11
2  2.189155e-16 1.000000e+00 2.500507e-09
3  2.924598e-03 8.589913e-77 9.970754e-01
4  7.629060e-04 3.906083e-42 9.992371e-01
5  1.092422e-02 4.084153e-76 9.890758e-01
6  7.771729e-48 1.000000e+00 2.311140e-45
7  7.312892e-01 1.053007e-03 2.676578e-01
8  9.811840e-01 2.655607e-45 1.881605e-02
9  1.744334e-23 1.000000e+00 9.958597e-22
10 9.812427e-04 1.743181e-26 9.990188e-01
11 1.133734e-03 1.025509e-74 9.988663e-01
12 6.510657e-04 4.571381e-38 9.993489e-01
13 9.532959e-01 1.139849e-47 4.670414e-02
14 1.238785e-02 2.276720e-46 9.876122e-01
15 4.529335e-04 6.569476e-24 9.995471e-01
16 8.002326e-04 5.327002e-31 9.991998e-01
17 6.694849e-44 1.000000e+00 1.515546e-41
18 9.851527e-04 5.831626e-31 9.990148e-01
19 1.422009e-02 6.272569e-19 9.857799e-01
20 9.421153e-01 5.343345e-10 5.788473e-02
```

Predict category

label=predict(mod1,xtest,type="class");label[1:20,]

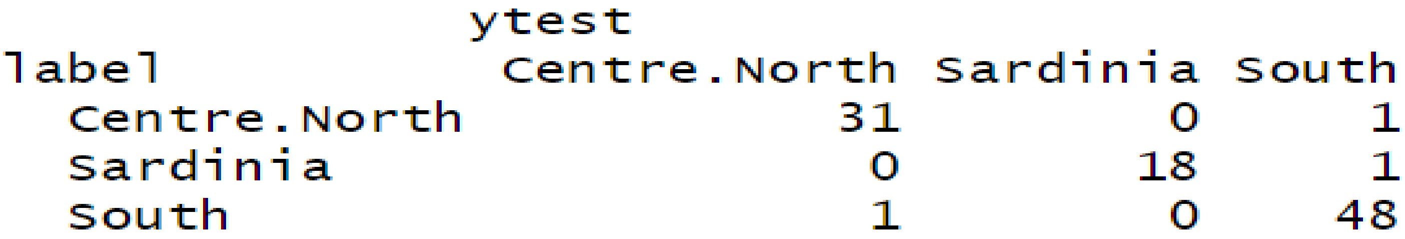


```
 [1] "Sardinia"     "Sardinia"     "South"        "South"        "South"       
 [6] "Sardinia"     "Centre.North" "Centre.North" "Sardinia"     "South"       
[11] "South"        "South"        "Centre.North" "South"        "South"       
[16] "South"        "Sardinia"     "South"        "South"        "Centre.North"
```

Multiple logistic regression

Confusion matrix

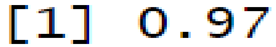
tab=table(label,ytest);tab



```
              ytest
label          Centre.North Sardinia South
  Centre.North           31        0     1
  Sardinia                0       18     1
  South                   1        0    48
```

Accuracy

sum(diag(tab))/sum(tab)



```
[1] 0.97
```

Linear Discriminant Analysis

Package

#install.packages('MASS')
library(MASS)

Model

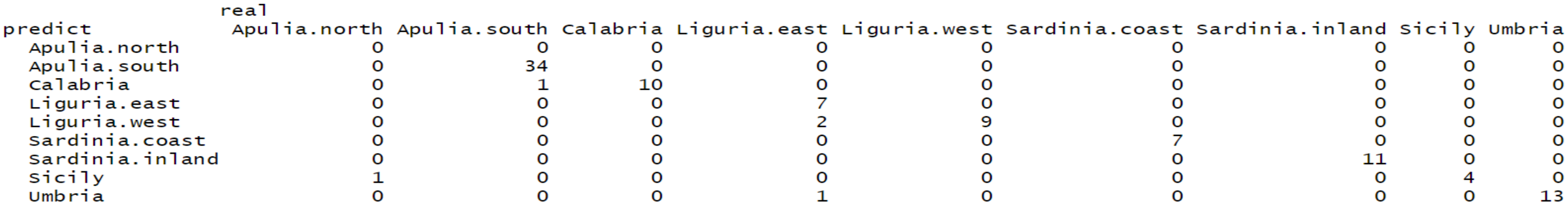
lda_model <-
lda(region~palmitic+palmitoleic+stearic+oleic+linoleic+linolenic+arachidic+eicosenoic,prior = c(1,1,1,1,1,1,1,1,1)/9,data = xytrain)

Predict category

label <- predict(lda_model,xytest[,2:9])$class

Confusion matrix

tab=table(predict=label,real=xytest$region);tab



```
                  real
predict           Apulia.north Apulia.south Calabria Liguria.east Liguria.west Sardinia.coast Sardinia.inland Sicily Umbria
  Apulia.north               0            0        0            0            0              0               0      0      0
  Apulia.south               0           34        0            0            0              0               0      0      0
  Calabria                   0            1       10            0            0              0               0      0      0
  Liguria.east               0            0        0            7            0              0               0      0      0
  Liguria.west               0            0        0            2            9              0               0      0      0
  Sardinia.coast             0            0        0            0            0              7               0      0      0
  Sardinia.inland            0            0        0            0            0              0              11      0      0
  Sicily                     1            0        0            0            0              0               0      4      0
  Umbria                     0            0        0            1            0              0               0      0     13
```

Accuracy

sum(diag(tab))/sum(tab)
[1] 0.95

LDA Visualize

Model

```
LDA=lda(macro.area~.,data=xy)
```

Get coordinate after dimensionality reduction

```
pre=data.frame(predict(LDA)$x)
```

Predict category

```
pre$class=predict(LDA)$class
```

Calculate mean for three category

```
setc=c(mean(pre$LD1[pre$class=='Centre.North']),mean(pre$LD2[pre$class=='Centre.North']))
verc=c(mean(pre$LD1[pre$class=='Sardinia']),mean(pre$LD2[pre$class=='Sardinia']))
virc=c(mean(pre$LD1[pre$class=='South']),mean(pre$LD2[pre$class=='South']))
mu=data.frame(LD1=c(setc[1],verc[1],virc[1]),LD2=c(setc[2],verc[2],virc[2]))
mu$class=c('Centre.North','Sardinia','South')
```

LDA Visualize

Model after dimensionality reduction

```
LDA2 <- lda(class ~ LD1 + LD2, data=pre)
```

Expand grid

```
ld1lim <- range(c(min(pre$LD1),max(pre$LD1)),mul=0.05)
ld2lim <- range(c(min(pre$LD2),max(pre$LD2)),mul=0.05)
ld1 <- seq(ld1lim[[1]], ld1lim[[2]], length.out=300)
ld2 <- seq(ld2lim[[1]], ld1lim[[2]], length.out=300)
```

Predict grids to category

```
newdat <- expand.grid(list(LD1=ld1,LD2=ld2))
preds <-predict(LDA2,newdata=newdat)
predclass <- preds$class
postprob <- preds$posterior
df <- data.frame(x=newdat$LD1, y=newdat$LD2, class=predclass)
df$classnum <- as.numeric(df$class)
df <- cbind(df,postprob)
```

Remove excess grids

```
df=df[-which(df$y>max(pre$LD2)),]
```

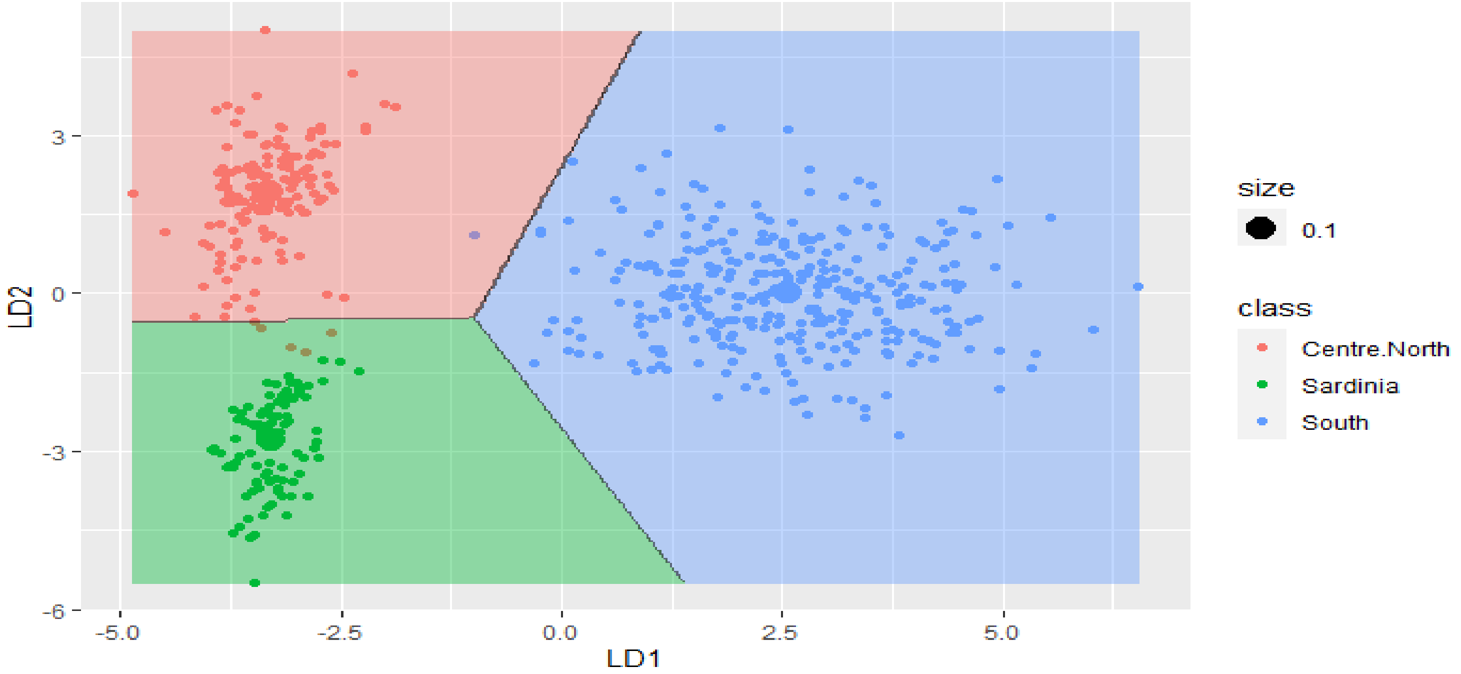
Replace true category to predict category

```
pre$class=xy$macro.area
```

LDA Visualize

Draw origin data, boundary and mean for three category

```
ggplot(pre, aes(x=LD1, y=LD2, colour=class) ) +
geom_point() +
geom_raster(data=df, aes(x=x, y=y, fill = factor(class)),alpha=0.4,show_guide=FALSE) +
geom_contour(data=df, aes(x=x, y=y, z=classnum), colour="black", alpha=0.5, breaks=c(1.5,2.5)) +
geom_point(data=mu, aes(x=LD1, y=LD2, colour=class,cex=0.1))
```



LDA Visualize

Package

```
#install.packages('ellipse')
library(ellipse)
dat_ell=pre
for(g in unique(pre$class))
{
exy=pre[pre$class==g,]
```

Subset by category

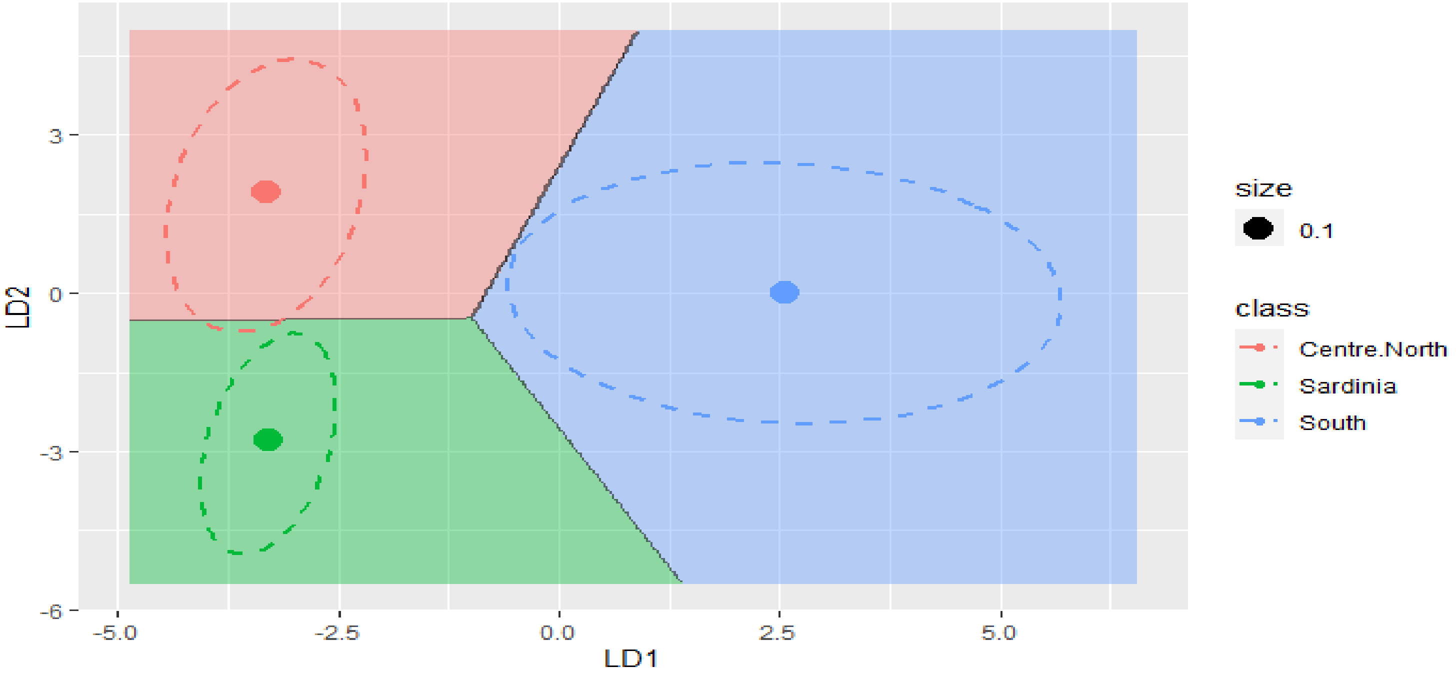
```
ex=exy$LD1
ey=exy$LD2
```

Draw ellipse with covariance, standard deviation and mean

```
ell=ellipse(cor(ex,ey),scale=c(sd(ex),sd(ey)),centre=c(mean(ex),mean(ey)))
ell=cbind(ell,rep(g,dim(exy)[1]))
ell=data.frame(ell)
colnames(ell)=colnames(pre)
dat_ell=rbind(dat_ell,ell)
}
dat_ell=dat_ell[-(1:dim(pre)[1]),]
dat_ell$LD1=as.numeric(dat_ell$LD1)
dat_ell$LD2=as.numeric(dat_ell$LD2)
```

LDA Visualize

```
ggplot(pre, aes(x=LD1, y=LD2, colour=class) ) +
geom_raster(data=df, aes(x=x, y=y, fill = factor(class)),alpha=0.4,show_guide=FALSE) +
geom_contour(data=df, aes(x=x, y=y, z=classnum), colour="black", alpha=0.5, breaks=c(1.5,2.5)) +
geom_point(data=mu, aes(x=LD1, y=LD2, colour=class,cex=0.1)) +
geom_path(data=dat_ell,aes(x=LD1,y=LD2,colour=class),size=1,linetype=2)
```



LDA Visualize In Iris Data

Model

```
LDA=lda(Species~.,data=iris)
```

Get coordinate after dimensionality reduction

```
pre=data.frame(predict(LDA)$x)
pre$LD1=-pre$LD1
```

Predict category

```
pre$class=predict(LDA)$class
```

Calculate mean for three category

```
setc=c(mean(pre$LD1[pre$class=='setosa']),mean(pre$LD2[pre$class=='setosa']))
verc=c(mean(pre$LD1[pre$class=='versicolor']),mean(pre$LD2[pre$class=='versicolor']))
virc=c(mean(pre$LD1[pre$class=='virginica']),mean(pre$LD2[pre$class=='virginica']))
mu=data.frame(LD1=c(setc[1],verc[1],virc[1]),LD2=c(setc[2],verc[2],virc[2]))
mu$class=c('setosa','versicolor','virginica')
```

LDA Visualize In Iris Data

Model after dimensionality reduction

```
LDA2 <- lda(class ~ LD1 + LD2, data=pre)
```

Expand grid

```
ld1lim <- range(c(min(pre$LD1),max(pre$LD1)),mul=0.05)
ld2lim <- range(c(min(pre$LD2),max(pre$LD2)),mul=0.05)
ld1 <- seq(ld1lim[[1]], ld1lim[[2]], length.out=300)
ld2 <- seq(ld2lim[[1]], ld1lim[[2]], length.out=300)
```

Predict grids to category

```
newdat <- expand.grid(list(LD1=ld1,LD2=ld2))
preds <-predict(LDA2,newdata=newdat)
predclass <- preds$class
postprob <- preds$posterior
df <- data.frame(x=newdat$LD1, y=newdat$LD2, class=predclass)
df$classnum <- as.numeric(df$class)
df <- cbind(df,postprob)
```

Remove excess grids

```
df=df[-which(df$y>max(pre$LD2)),]
```

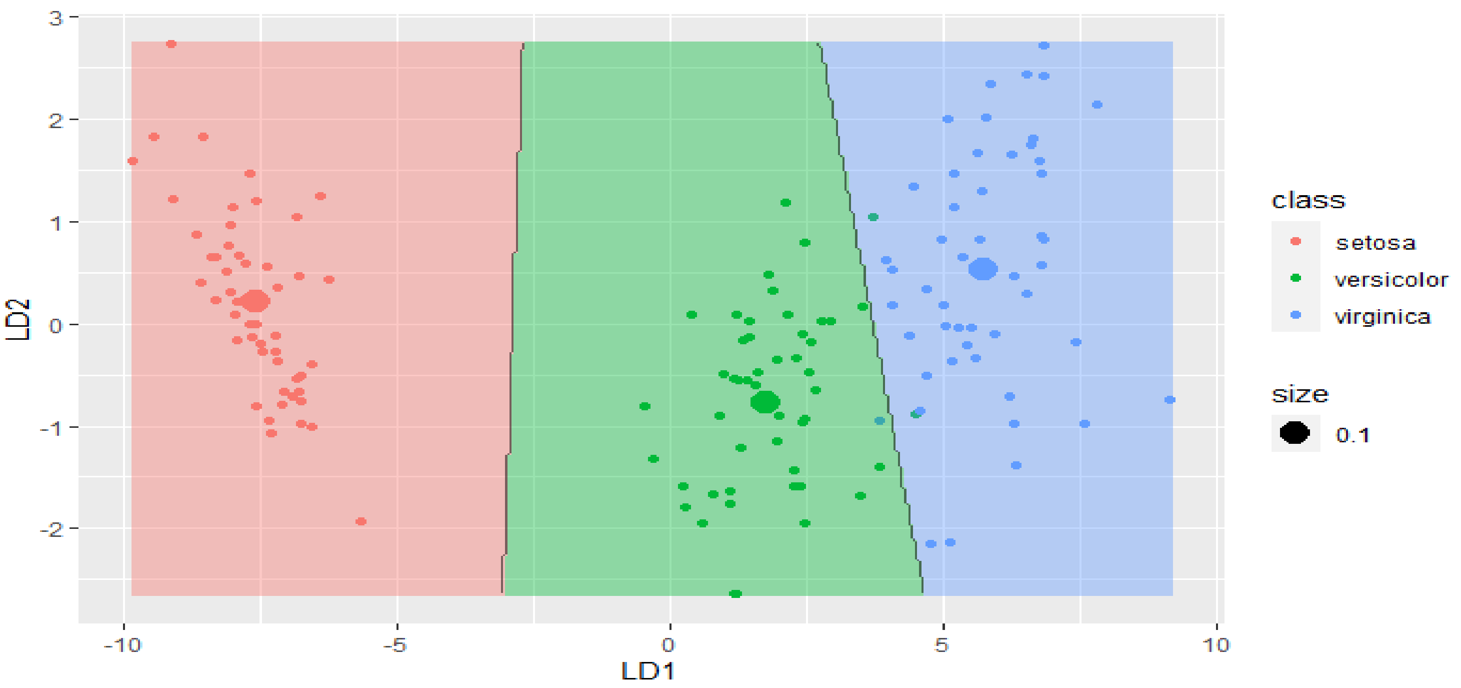
Replace true category to predict category

```
pre$class=iris$Species
```

LDA Visualize In Iris Data

Draw origin data, boundary and mean for three category

```
ggplot(pre, aes(x=LD1, y=LD2, colour=class) ) +
geom_point() +
geom_raster(data=df, aes(x=x, y=y, fill = factor(class)),alpha=0.4,show_guide=FALSE) +
geom_contour(data=df, aes(x=x, y=y, z=classnum), colour="black", alpha=0.5, breaks=c(1.5,2.5)) +
geom_point(data=mu, aes(x=LD1, y=LD2, colour=class,cex=0.1))
```



Quadratic Discriminant Analysis

Package

#install.packages('MASS')
library(MASS)

Model

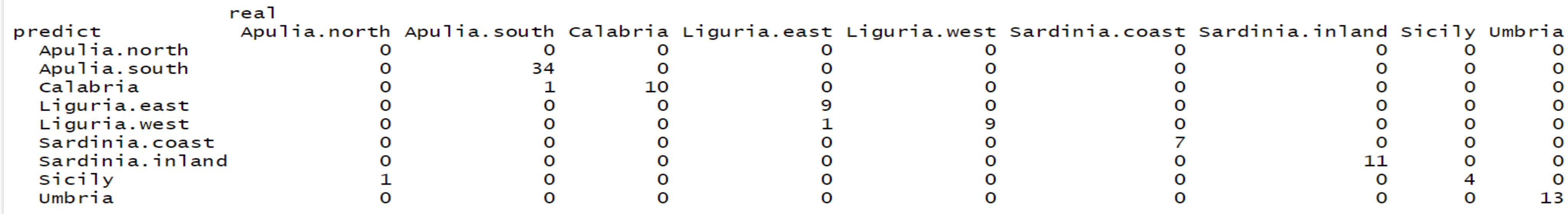
qda_model <- qda(region~palmitic+palmitoleic+stearic+oleic+linoleic+linolenic+arachidic+eicosenoic,prior = c(1,1,1,1,1,1,1,1,1)/9,data = xytrain)

Predict category

label <- predict(qda_model,xytest[,2:9])$class)

Confusion matrix

tab=table(predict=label,real=xytest$region);tab



```
                   real
predict             Apulia.north Apulia.south Calabria Liguria.east Liguria.west Sardinia.coast Sardinia.inland Sicily Umbria
  Apulia.north                 0            0        0            0            0              0               0      0      0
  Apulia.south                 0           34        0            0            0              0               0      0      0
  Calabria                     0            1       10            0            0              0               0      0      0
  Liguria.east                 0            0        0            9            0              0               0      0      0
  Liguria.west                 0            0        0            1            9              0               0      0      0
  Sardinia.coast               0            0        0            0            0              7               0      0      0
  Sardinia.inland              0            0        0            0            0              0              11      0      0
  Sicily                       1            0        0            0            0              0               0      4      0
  Umbria                       0            0        0            0            0              0               0      0     13
```

Accuracy

sum(diag(tab))/sum(tab)
[1] 0.97

QDA Visualize

Model after dimensionality reduction

```
QDA <- qda(class ~ LD1 + LD2, data=pre)
```

Expand grid

```
ld1lim <- range(c(min(pre$LD1),max(pre$LD1)),mul=0.05)
ld2lim <- range(c(min(pre$LD2),max(pre$LD2)),mul=0.05)
ld1 <- seq(ld1lim[[1]], ld1lim[[2]], length.out=300)
ld2 <- seq(ld2lim[[1]], ld1lim[[2]], length.out=300)
```

Predict grids to category

```
newdat <- expand.grid(list(LD1=ld1,LD2=ld2))
preds <-predict(LDA2,newdata=newdat)
predclass <- preds$class
postprob <- preds$posterior
df <- data.frame(x=newdat$LD1, y=newdat$LD2, class=predclass)
df$classnum <- as.numeric(df$class)
df <- cbind(df,postprob)
```

Remove excess grids

```
df=df[-which(df$y>5),]
```

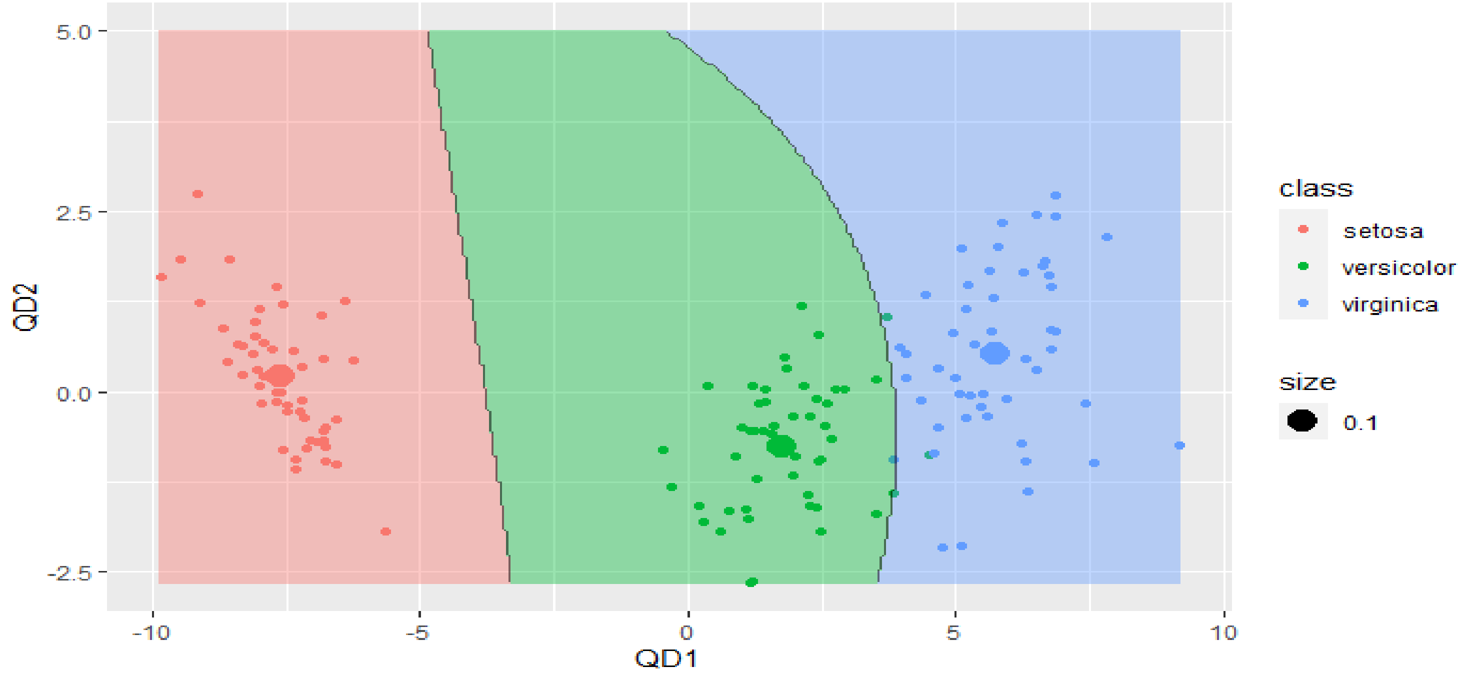
Replace true category to predict category

```
pre$class=iris$Species
```

QDA Visualize

Draw origin data, boundary and mean for three category

```
p=ggplot(pre, aes(x=LD1, y=LD2, colour=class) ) +
geom_point() +
geom_raster(data=df, aes(x=x, y=y, fill = factor(class)),alpha=0.4,show_guide=FALSE) +
geom_contour(data=df, aes(x=x, y=y, z=classnum), colour="black", alpha=0.5, breaks=c(1.5,2.5)) +
geom_point(data=mu, aes(x=LD1, y=LD2, colour=class,cex=0.1))
p+labs(x='QD1',y=('QD2'))
```



Naive Bayes

Package

```
#install.packages('e1071')
library(e1071)
```

Model

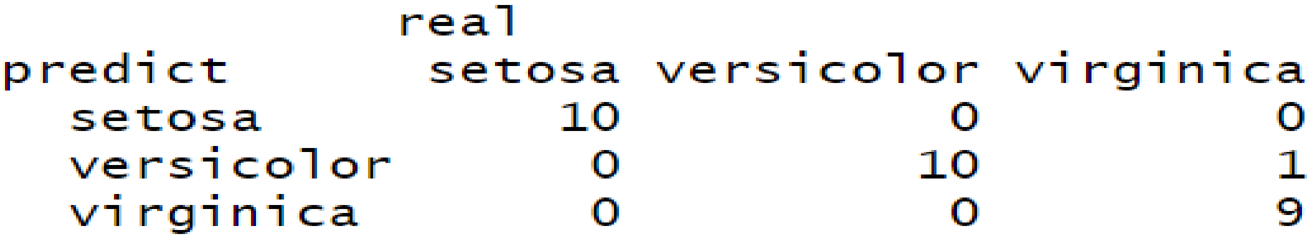
```
NBclass=naiveBayes(Species~.,data=xytrain)
```

Predict category

```
label=predict(NBclass,xytest)
```

Confusion matrix

```
tab=table(predict=label,real=xytest$Species);tab
```



```
            real
predict      setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         1
  virginica       0          0         9
```

Accuracy

```
sum(diag(tab))/sum(tab)
[1] 0.9666667
```

Homework 7 (submitted to e3.nycu.edu.tw before Nov 22, 2023)

- Use R, Python, and suitable computer packages to perform different types of Classification Models.
- Explain the results you obtain.
- Compare the results you obtain by different Classification approaches.
- Discuss possible problems you plan to investigate for future studies

Possible sources of open datasets:

- ➢ UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.php)
- ➢ Kaggle Datasets (https://www.kaggle.com/datasets)
- ➢ World Health Organization Datasets (https://www.who.int/)