Econ 425 Week 2

Classification I: logistic regression

Grigory Franguridi

UCLA Econ

USC CESR

franguri@usc.edu

Classification

- Data $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^n$.
    - $\boldsymbol{x}_i$: covariates of i-th instance
    - $y_i \in \{-1, 1\}$: label of i-th instance

- **Question**: Can we directly minimize the averaged 0-1 loss (misclassification rate)?

$$R(f) = \frac{1}{n} \sum_{i=1}^{n} I(f(\boldsymbol{x}_i) \neq y_i)$$

- **Answer**: No, the 0-1 loss function is non-convex and discontinuous, so (sub)gradient methods cannot be applied

Classification: surrogate loss

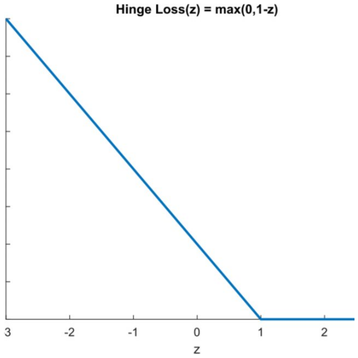
- Replace the 0-1 loss with other loss functions, viz.

$$\frac{1}{n}\sum_{i=1}^{n} L(f(\boldsymbol{x}_i), y_i) = \frac{1}{n}\sum_{i=1}^{n} \phi(f(\boldsymbol{x}_i)y_i)$$

  - Hinge loss: $\phi(x) = \max\{0, 1 - x\}$
  - Logistic loss: $\phi(x) = \log(1 + \exp(-x))$

Hinge Loss

$$L_{hinge}(f(\boldsymbol{x}_i), y_i) = (1 - f(\boldsymbol{x}_i)y_i)_+ = \begin{cases} 1 - f(\boldsymbol{x}_i)y_i & \text{if } f(\boldsymbol{x}_i)y_i \leq 1 \\ 0, & \text{if } f(\boldsymbol{x}_i)y_i > 1 \end{cases}$$



Why Hinge Loss?

- Let $\eta(\boldsymbol{x}) = \mathbb{P}(Y = 1|\boldsymbol{X} = \boldsymbol{x})$. Then **hinge risk**

$$\begin{aligned} R_{hinge}(f) =& \mathbb{E}_{\boldsymbol{X},Y}\left[L_{hinge}(f(\boldsymbol{X}), Y)\right] \\ =& \mathbb{E}_X\Big[\eta(\boldsymbol{X})(1 - f(\boldsymbol{X}))_+ + \big(1 - \eta(\boldsymbol{X})\big)(1 + f(\boldsymbol{X}))_+\Big] \end{aligned}$$

- Suppose $f(\boldsymbol{X}) \in [-1, 1]$ for any $\boldsymbol{X}$. Then

$$\begin{aligned} &\eta(\boldsymbol{X})(1 - f(\boldsymbol{X})) + \big(1 - \eta(\boldsymbol{X})\big)(1 + f(\boldsymbol{X})) \\ &= \eta(\boldsymbol{X}) - 2\eta(\boldsymbol{X})f(\boldsymbol{X}) + 1 + f(\boldsymbol{X}) - \eta(\boldsymbol{X}) \\ &= f(\boldsymbol{X})\big(1 - 2\eta(\boldsymbol{X})\big) + 1. \end{aligned}$$

- Minimizer f^*_{hinge} of $R_{hinge}(f)$ is (why?)
  - If $\eta(\boldsymbol{X}) < 1/2$, then $f^*_{hinge}(\boldsymbol{X}) = -1$
  - If $\eta(\boldsymbol{X}) > 1/2$, then $f^*_{hinge}(\boldsymbol{X}) = 1$

Why Hinge Loss?

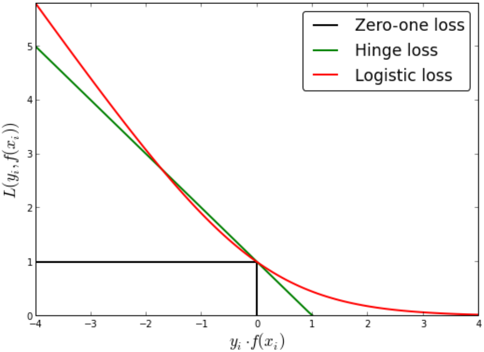
- Under binary loss, the optimal (Bayes) classifier is

$$f^*(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \mathbb{P}(Y = 1 | \boldsymbol{X} = \boldsymbol{x}) > 1/2 \\ -1, & \text{if } \mathbb{P}(Y = 1 | \boldsymbol{X} = \boldsymbol{x}) < 1/2 \end{cases}$$

- f^*_{hinge} is exactly Bayes classifier
- f^*_{hinge} is convex, allowing minimization of training error in practice

Another surrogate loss: logistic

$$L_{log}(f(\boldsymbol{x}_i), y_i) = \log\Big(1 + \exp(-f(\boldsymbol{x}_i)y_i)\Big)$$



Why Logistic Loss?

- Logistic risk:

$$
\begin{aligned}
R_{log}(f) &= \mathbb{E}_{\boldsymbol{X},Y}\Big[\log\Big(1+\exp(-f(\boldsymbol{X})Y)\Big)\Big] \\
=&\mathbb{E}_{\boldsymbol{X}}\Big[\eta(\boldsymbol{X})\log\Big(1+\exp(-f(\boldsymbol{X}))\Big) + \big(1-\eta(\boldsymbol{X})\big)\log\Big(1+\exp(f(\boldsymbol{X}))\Big)\Big]
\end{aligned}
$$

- Take derivative w.r.t. f,

$$
\begin{aligned}
&-\eta(\boldsymbol{X})\frac{\exp(-f(\boldsymbol{X}))}{1+\exp(-f(\boldsymbol{X}))} + \big(1-\eta(\boldsymbol{X})\big)\frac{\exp(f(\boldsymbol{X}))}{1+\exp(f(\boldsymbol{X}))} \\
=&-\eta(\boldsymbol{X})\frac{1}{1+\exp(f(\boldsymbol{X}))} + \big(1-\eta(\boldsymbol{X})\big)\frac{\exp(f(\boldsymbol{X}))}{1+\exp(f(\boldsymbol{X}))} \\
=&\frac{\exp(f(\boldsymbol{X}))}{1+\exp(f(\boldsymbol{X}))} - \eta(\boldsymbol{X}) = 0 \longleftrightarrow f^*_{log}(\boldsymbol{X}) = \log\frac{\eta(\boldsymbol{X})}{1-\eta(\boldsymbol{X})}
\end{aligned}
$$

Connection between binary loss and surrogate losses

- Binary-optimal (Bayes) classifier $f^*(\boldsymbol{x}) = \text{sign}(\eta(\boldsymbol{x}) - 1/2)$
- Hinge-optimal classifier $f^*_{hinge}(\boldsymbol{x}) = \text{sign}(\eta(\boldsymbol{x}) - 1/2)$
- Logistic-optimal classifier $f^*_{log}(\boldsymbol{x}) = \log \frac{\eta(\boldsymbol{X})}{1-\eta(\boldsymbol{X})}$
- **Question**: what is the connection between these optimal classifiers?
- **Answer**: The signs of $f^*, f^*_{hinge}, f^*_{log}$ are always the same, e.g., always positive as long as $\eta(\boldsymbol{x}) > 1/2$.

Logistic Regression

- To estimate $f^*_{log}(\boldsymbol{x})$, impose parametric structure on

$$\eta(\boldsymbol{x}) = \mathbb{P}\Big(Y = 1 \Big| \boldsymbol{X} = \boldsymbol{x}\Big)$$

- Logistic regression:

$$\eta(\boldsymbol{x}) = \frac{\exp(\beta_0 + \boldsymbol{\beta}'\boldsymbol{x})}{1 + \exp(\beta_0 + \boldsymbol{\beta}'\boldsymbol{x})}, \tag{1}$$

where

- $\boldsymbol{x} = (x_1, \ldots, x_p)'$ is a p-dimensional predictor
- β_0 and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)'$ are unknown parameters
- $\boldsymbol{\beta}'\boldsymbol{x} = \sum_{l=1}^{p} \beta_i x_i$

Log odds ratio

- Rewrite (1) to get

$$\exp(\beta_0 + \boldsymbol{\beta}'\boldsymbol{x}) = \frac{\mathbb{P}\Big(Y = 1\Big|\boldsymbol{X} = \boldsymbol{x}\Big)}{1 - \mathbb{P}\Big(Y = 1\Big|\boldsymbol{X} = \boldsymbol{x}\Big)} = \frac{\mathbb{P}\Big(Y = 1\Big|\boldsymbol{X} = \boldsymbol{x}\Big)}{\mathbb{P}\Big(Y = 0\Big|\boldsymbol{X} = \boldsymbol{x}\Big)},$$

  where the last expression is "odds ratio"

- In other words, the log odds ratio is linear in $\boldsymbol{\beta}$,

$$\log \frac{\mathbb{P}\Big(Y = 1\Big|\boldsymbol{X} = \boldsymbol{x}\Big)}{\mathbb{P}\Big(Y = 0\Big|\boldsymbol{X} = \boldsymbol{x}\Big)} = \beta_0 + \boldsymbol{\beta}'\boldsymbol{x}$$

  Interpretation: β_i is the average change in the log odds ratio under one-unit increase in x_i

Maximum likelihood estimation

- Likelihood function

$$L(\beta_0, \boldsymbol{\beta}) = \prod_{i=1}^{n} \mathbb{P}\big(Y = 1 \big| \boldsymbol{X} = \boldsymbol{x}\big)^{y_i} \mathbb{P}\big(Y = 0 \big| \boldsymbol{X} = \boldsymbol{x}\big)^{1-y_i}$$

- Log-likelihood

$$\begin{aligned}
&\log L(\beta_0, \boldsymbol{\beta}) \\
&= \sum_{i=1}^{n} \Big[y_i \log \mathbb{P}\big(Y = 1 \big| \boldsymbol{X} = \boldsymbol{x}\big) + (1 - y_i) \log \mathbb{P}\big(Y = 0 \big| \boldsymbol{X} = \boldsymbol{x}\big) \Big] \\
&= \sum_{i=1}^{n} \Big[y_i \big(\beta_0 + \boldsymbol{\beta}' \boldsymbol{x}\big) - \log \Big(1 + \exp(\beta_0 + \boldsymbol{\beta}' \boldsymbol{x})\Big) \Big]
\end{aligned}$$

Computation: gradient ascent

Let $\beta_0^{(t)}, \boldsymbol{\beta}^{(t)}$ be t-th step estimates.

Update (gradient ascent):

$$\beta_0^{(t+1)} \leftarrow \beta_0^{(t)} + \lambda \sum_{i=1}^{n} \left[y_i - \frac{\exp(\beta_0^{(t)} + \boldsymbol{x}'\boldsymbol{\beta}^{(t)})}{1 + \exp(\beta_0^{(t)} + \boldsymbol{x}'\boldsymbol{\beta}^{(t)})} \right],$$

$$\boldsymbol{\beta}^{(t+1)} \leftarrow \boldsymbol{\beta}^{(t)} + \lambda \sum_{i=1}^{n} \left[y_i - \frac{\exp(\beta_0^{(t)} + \boldsymbol{x}'\boldsymbol{\beta}^{(t)})}{1 + \exp(\beta_0^{(t)} + \boldsymbol{x}'\boldsymbol{\beta}^{(t)})} \right] \boldsymbol{x}_i,$$

where $\lambda > 0$ is step size (tuning parameter)

Logistic classifiction

Denote the final estimates by $\widehat{\beta}_0$ and $\widehat{\boldsymbol{\beta}}$

Estimate class probability $P(Y = 1|\boldsymbol{X})$ as

$$\widehat{\eta}(\boldsymbol{x}) = \frac{\exp(\widehat{\beta}_0 + \widehat{\boldsymbol{\beta}}'\boldsymbol{x})}{1 + \exp(\widehat{\beta}_0 + \widehat{\boldsymbol{\beta}}'\boldsymbol{x})}$$

Then the **logistic classifier** is (recall $\eta(\boldsymbol{x}) = P(Y = 1|\boldsymbol{X})$)

$$\widehat{f}(\boldsymbol{x}) = \begin{cases} 1, \text{ if } \widehat{\eta}(\boldsymbol{x}) > 1/2 \\ 0, \text{ if } \widehat{\eta}(\boldsymbol{x}) < 1/2 \end{cases}$$

If $\widehat{\eta}(\boldsymbol{x}) = 1/2$, then assign a label $\in \{0, 1\}$ randomly

Example

Data $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^{5000}$, where $\boldsymbol{x}_i = (x_{i1}, x_{i2}, x_{i3}, x_{i4})$.

- $x_{il} \sim Unif(0, 2)$, $l = 1, \ldots, 4$.
- $\beta_0 = 0.5$, $\boldsymbol{\beta} = (\beta_1, \beta_2, \beta_3, \beta_4)$ with $\beta_i \sim Unif(-1, 1)$
- Model:

$$Y_i \sim Bernoulli\left(\frac{\exp(\beta_0 + \boldsymbol{\beta}'\boldsymbol{x})}{1 + \exp(\beta_0 + \boldsymbol{\beta}'\boldsymbol{x})}\right),$$

i.e.

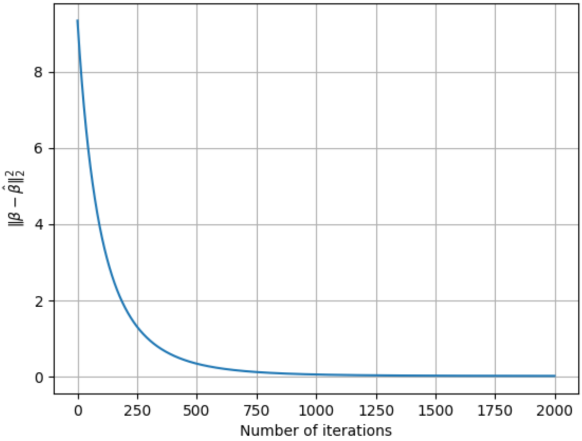
$$P(Y_i = 1|\boldsymbol{X}) = \frac{\exp(\beta_0 + \boldsymbol{\beta}'\boldsymbol{x})}{1 + \exp(\beta_0 + \boldsymbol{\beta}'\boldsymbol{x})}$$

Python code – data generation

```
import numpy as np
np.random.seed(2)
n,p = 5000,4  # Set training datasize and dimension of features
X = np.random.uniform(-1,1,[n,p]) # Generation of features
beta = np.random.uniform(0,2,4) # Generation of parameters
beta_0 = 0.5 # Set the intercept term to 0.5
logOdd = (X * beta).sum(axis=1)+beta_0 # Log-odds
Prob = np.exp(logOdd)/(1+np.exp(logOdd)) # Probability
Y = np.array(Prob - np.random.uniform(0,1,n)>0,dtype=int) # Generate labels
```

```
Beta_0_hat = 0. # Initialization of intercept term
Beta_hat = np.zeros(p) # Initialization of beta
lamb = 0.1 # Learning rate
Error = [] # Error set
for i in range(2000):# Iterations of gradient ascent
    logOdd_hat = (X * Beta_hat).sum(axis=1)+Beta_0_hat
    Beta_0_hat = Beta_0_hat + lamb * np.mean(Y - np.exp(logOdd_hat)/(1+np.exp(logOdd_hat)))
    Beta_hat = Beta_hat + lamb * ((Y - np.exp(logOdd_hat)/(1+np.exp(logOdd_hat))) * X.T).mean(axi
    Error.append(np.linalg.norm(Beta_hat-beta)**2)
import matplotlib.pyplot as plt
plt.plot(np.arange(0,2000),Error)
plt.xlabel('Number of iterations')
plt.ylabel('$\Vert \\beta - \hat{\\beta}\Vert_2^2$')
plt.grid()
```

Example: gradient ascent for logistic regression

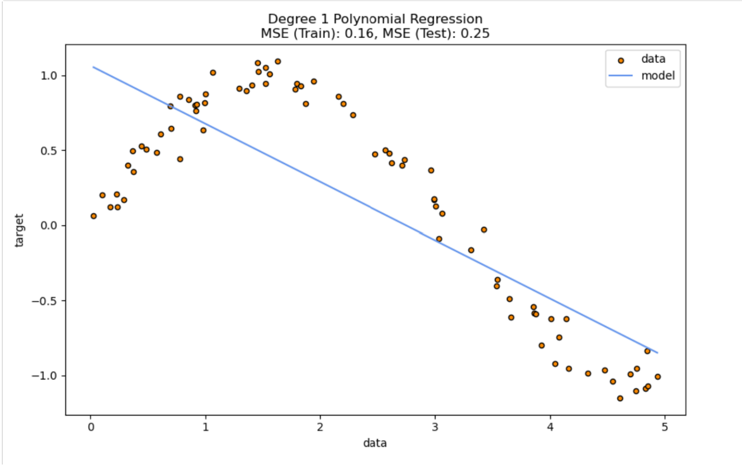


Over/underfitting problem

- **Overfitting**: model is **too flexible/complex**, learns the training data too well, capturing noise and making it perform poorly on unseen data
  - low bias, high variance
- **Underfitting**: model is **too rigid/simple** to capture patterns in the data, resulting in poor performance on both the training and test data
  - high bias, low variance
- Simple example: polynomial regression

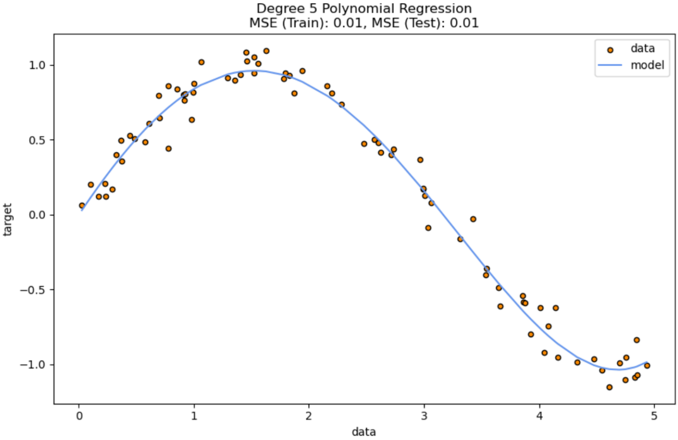
Example: underfitting

Degree 1: The model is unable to capture the patterns in the data



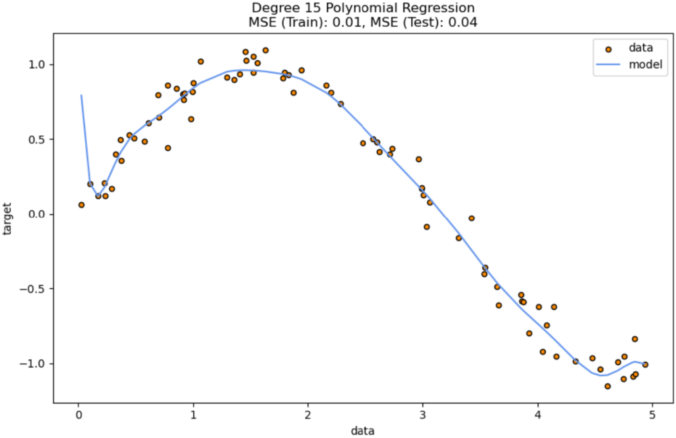
Example: good fit

Degree 5: The model captures the patterns well and generalizes to test data



Example: overfitting

Degree 15: The model fits the training data too well, capturing noise and performing poorly on unseen data



Classification outcomes

- true positive (TP): a correctly predicted positive data point, i.e., $\widehat{y} = y = 1$
- true negative (TN): a correctly predicted negative data point, i.e., $\widehat{y} = y = -1$
- false positive (FP): an incorrectly predicted positive data point, i.e., $\widehat{y} = 1$ but $y = -1$
- false negative (FN): an incorrectly predicted negative data point, i.e., $\widehat{y} = -1$ but $y = 1$

Example: image recognition

$Y = 1$ (dog), $Y = -1$ (not dog)

index	actual	predicted	Result
1	Dog	Dog	TP
2	Dog	Not Dog	FN
3	Dog	Dog	TP
4	Not Dog	Not Dog	TN
5	Dog	Dog	TP
6	Not Dog	Dog	FP
7	Dog	Dog	TP
8	Dog	Dog	TP
9	Not Dog	Not Dog	TN
10	Not Dog	Not Dog	TN

Example: image recognition

- Actual Dog Counts = ?
- Actual Not Dog Counts = ?
- True Positive Counts = ?
- False Positive Counts = ?
- True Negative Counts = ?
- False Negative Counts = ?

Example: image recognition

- Actual Dog Counts $= 6$
- Actual Not Dog Counts $= 4$
- True Positive Counts $= 5$
- False Positive Counts $= 1$
- True Negative Counts $= 3$
- False Negative Counts $= 1$

Example: confusion matrix

		Actual	
		Dog	**Not Dog**
Predicted	**Dog**	True Positive (TP =5)	False Positive (FP=1)
	Not Dog	False Negative (FN =1)	True Negative (TN=3)

- How to use the confusion matrix to assess classifier performance?

Classification metrics: accuracy

- Accuracy is the ratio of total correct instances to the total instances
- Accuracy $= \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$
- In our example: Accuracy = ?
- In our example: Accuracy $= (5+3)/(5+3+1+1) = 8/10 = 0.8$

Classification metrics: precision

- Precision is proportion of correct positive classifications
- Def: ratio of true positive predictions to the total number of positive predictions
- Precision $= \frac{\text{TP}}{\text{TP}+\text{FP}}$
- In our example: Precision = ?
- In our example: Precision $= 5/(5+1) = 5/6 = 0.8333$

Classification metrics: recall

- Recall measures the proportion of correctly classified positives
- Def: ratio of the number of true positive instances to the total number of positive instances
- Recall $= \frac{\text{TP}}{\text{TP}+\text{FN}}$
- In our example: Recall = ?
- In our example: Recall $= 5/(5+1) = 5/6 = 0.8333$

Classification metrics: F1 Score

- F1 score is the **harmonic mean of precision and recall**
- Harmonic mean is often used to calculate the average of ratios/rates
- Harmonic mean is the reciprocal of the arithmetic mean of the reciprocals of observations
- Example: harmonic mean of $1, 4, 4$ is

$$\left(\frac{1^{-1} + 4^{-1} + 4^{-1}}{3}\right)^{-1} = 2$$

Classification metrics based on confusion matrix: F1 Score

- F1 score $= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
- In our example: F1-Score=?
- In our example: F1-Score $=$ $(2 * 0.8333 * 0.8333)/(0.8333 + 0.8333) = 0.8333$

Example: classification metrics

- A binary classification model is used to predict whether an email is spam (positive class) or not spam (negative class). After testing the model on a dataset of 100 emails, you get the following results:
  - 40 emails are correctly identified as spam (True Positives).
  - 10 emails are incorrectly identified as spam (False Positives).
  - 30 emails are correctly identified as not spam (True Negatives).
  - 20 emails are incorrectly identified as not spam (False Negatives).

Example: classification metrics

Problem:

- construct the confusion matrix
- calculate the following metrics:
    - Accuracy
    - Precision
    - Recall
    - F1 Score

Example: classification metrics

Solution

- confusion matrix:
  - True Positives (TP): 40
  - False Positives (FP): 10
  - True Negatives (TN): 30
  - False Negatives (FN): 20
- Accuracy $= \frac{40+30}{40+30+10+20} = 0.7$
- Precision $= \frac{40}{40+10} = 0.8$
- Recall $= \frac{40}{40+20} = \frac{2}{3}$
- F-1 Score $= 2 \times \frac{0.8 \times 0.667}{0.8+0.667} \approx 0.727$

Application: banknote authentication

- The banknote authentication dataset is used for determining if a banknote is authentic based on features of wavelet-transformed images of banknotes
- Features:
  - x_1 is the variance of the image
  - x_2 is the skewness of the image
  - x_3 is the kurtosis of the image
  - x_4 is the entropy of the image

Application: banknote authentication

- Logistic regression:

$$P(\text{authentic}|X) = \frac{1}{1 + \exp\{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4\}}$$

- Classifier:
  - if $P(\text{authentic}|X) \geq 0.5$, predict that the banknote is authentic
  - if $P(\text{authentic}|X) < 0.5$, predict that the banknote is not authentic

Application: banknote authentication

```
   Variance  Skewness  Curtosis  Entropy  Class
0   3.62160    8.6661   -2.8073 -0.44699      0
1   4.54590    8.1674   -2.4586 -1.46210      0
2   3.86600   -2.6383    1.9242  0.10645      0
3   3.45660    9.5228   -4.0112 -3.59440      0
4   0.32924   -4.4552    4.5718 -0.98880      0
Accuracy: 0.98
Confusion Matrix:
[[144   4]
 [  2 125]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       148
           1       0.97      0.98      0.98       127

    accuracy                           0.98       275
   macro avg       0.98      0.98      0.98       275
weighted avg       0.98      0.98      0.98       275
```

Python package statsmodels

- designed for statistical modeling and hypothesis testing
- e.g. linear regression, logistic regression, time-series models
- includes modules for performing hypothesis tests, constructing confidence intervals, and fitting various statistical models with an emphasis on providing detailed statistical information
- widely used in business and academic research

Statsmodels: Linear regression

```python
import statsmodels.api as sm
import numpy as np

# Generate some random data for demonstration
np.random.seed(42)
X = np.random.rand(100, 2)
y = 3 * X[:, 0] + 2 * X[:, 1] + 1 + 0.1 * np.random.randn(100)

# Add a constant term to the independent variable
X = sm.add_constant(X)

# Create a linear model
model = sm.OLS(y, X)
results = model.fit()

# Print detailed statistical summary
print(results.summary())
```

Statsmodels: output

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.991
Model:                            OLS   Adj. R-squared:                  0.991
Method:                 Least Squares   F-statistic:                     5655.
Date:                Thu, 21 Dec 2023   Prob (F-statistic):          3.86e-101
Time:                        10:50:14   Log-Likelihood:                 89.304
No. Observations:                 100   AIC:                            -172.6
Df Residuals:                      97   BIC:                            -164.8
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.9772      0.026     37.651      0.000       0.926       1.029
x1             3.0339      0.033     91.615      0.000       2.968       3.100
x2             2.0355      0.035     57.426      0.000       1.965       2.106
==============================================================================
Omnibus:                        5.986   Durbin-Watson:                   2.104
Prob(Omnibus):                  0.050   Jarque-Bera (JB):                5.624
Skew:                           0.439   Prob(JB):                       0.0601
Kurtosis:                       3.761   Cond. No.                         5.22
==============================================================================
```

Python library scikit-learn

- provides tools for various ML tasks
- e.g., classification, regression, clustering, dimensionality reduction
- provides a consistent interface for various ML algorithms, making it easy to train models, perform feature engineering, and evaluate model performance
- widely used in industry for building and deploying machine learning models in areas such as image recognition, natural language processing, and predictive analytics