Topics

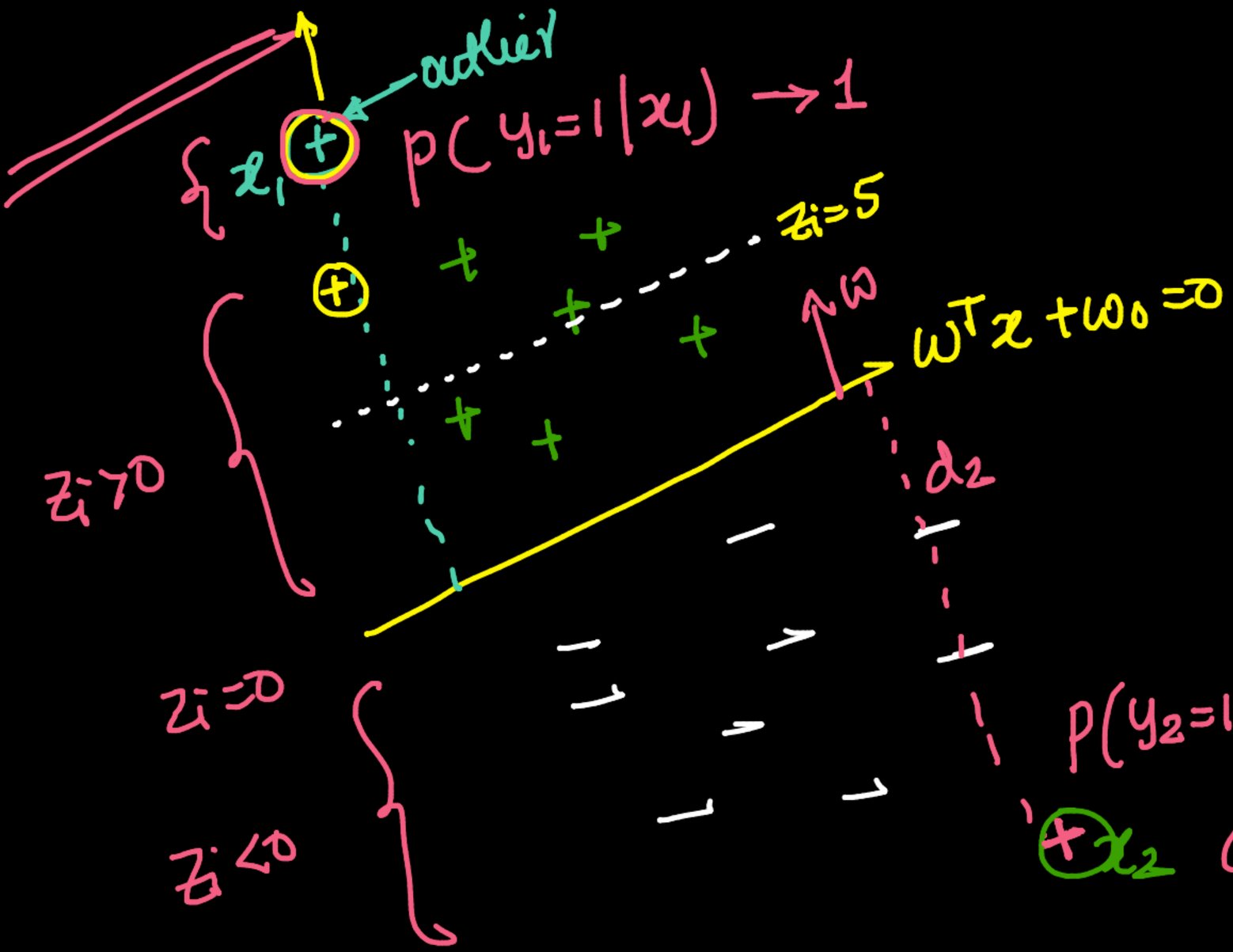
- Logistic regression (cont.) → outliers (visual)
- Churn case-study (fix) → imbalanced-data
- Confusion Matrix ; Precision ; Recall ; F1
- KNN (in detail)
  - simple (intuitive)
- odds ratio

Intuition + Math

weights

Next
- AUC-ROC
- PR-curve
- KNN (cont.)
- Imbalanced data

Logistic Regr (outliers)



Yes NO

$w^T x_i + w_0 = z_i$

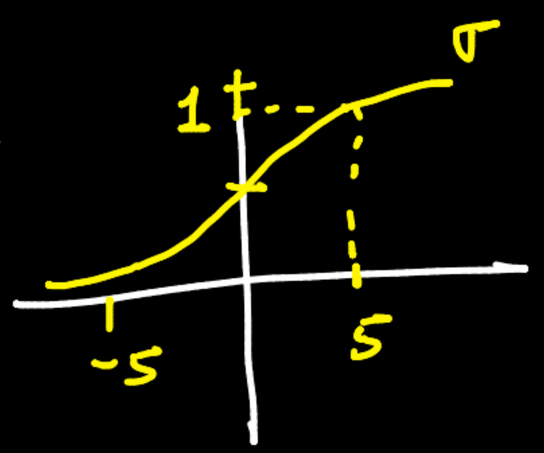
Case 1: +ve outlier on the correct side

$w^T x_i + w_0$: large +ve

squashing: $\sigma(w^T x_i + w_0)$: close to +1 (0.95 to 1)

↳ not changing a lot

$y_i = 1$



log-loss: $-\log(\hat{y}_i)$

↳ ≈0

{ Not much impact on w_{sep}

$$\text{opt}: \min_{w_j} \sum_{i=1}^{n} \underbrace{\text{log-loss}_i}_{} + \lambda \text{reg}$$

Case 2: +ve labelled point ($y_i = 1$) which is on wrong side

$z_2 = w^T x_2 + w_0$: large -ve value

$\sigma(z_2)$: close to zero $= \hat{y}_i$

$\log\text{-loss}_2 = -\log(\hat{y}_i)$: large +ve value

Chrome File Edit View History Bookmarks Profiles Tab Window Help 21:18

Logistic Regression.ipynb - Co | sklearn.linear_model.LogisticR | Classifier comparison — scikit- | plot(-log(x)) - Google Search

google.com/search?q=plot(-log(x))&rlz=1C5CHFA_enIN958IN958&oq=plot(-log(x&aqs=chrome.0.69i59j69i57j69i59j0i30l3j0i10i30j0i8i30l3.6304j0j7&sourceid=chrome&ie=UTF-8

Update

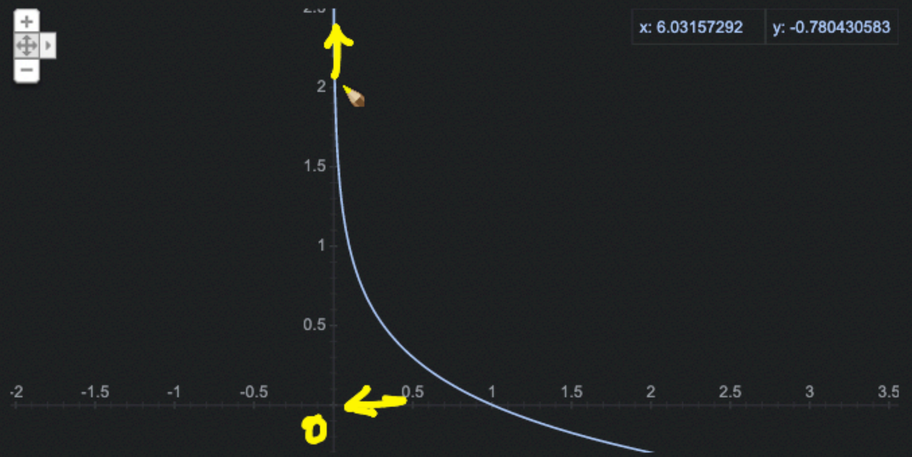
Google

plot(-log(x))

All Maps Images Videos News More Tools

About 1,42,00,00,000 results (0.53 seconds)

Graph for -log(x)



More info

$y_i = 1$

$\uparrow$

$y_i \rightarrow 0$

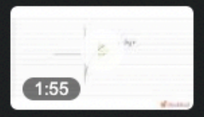
$$\text{log-loss}_i = -\log(\hat{y}_i)$$

https://www.rapidtables.com › math › algebra › logarith...

logarithm graph | graph of log(x) - RapidTables.com

Graph of **log(x)**. **log(x)** function **graph**. Logarithm **graph**. y = f (x) = log10(x). **log(x) graph** properties. **log(x)** is defined for positive values of x. **log(x)** ...

Videos



Draw the graph of `y=-log(x)` when the graph of `y=logx` is ...

YouTube · Doubtnut
31-May-2020

Graphing a Logarithm - Made Easy!

6 / 6

Opt: $\min_{w_j} \sum_{i=1}^{n} \overbrace{\text{logloss}_i}^{} + \lambda \text{ reg}$

log-loss_2 (outlier)

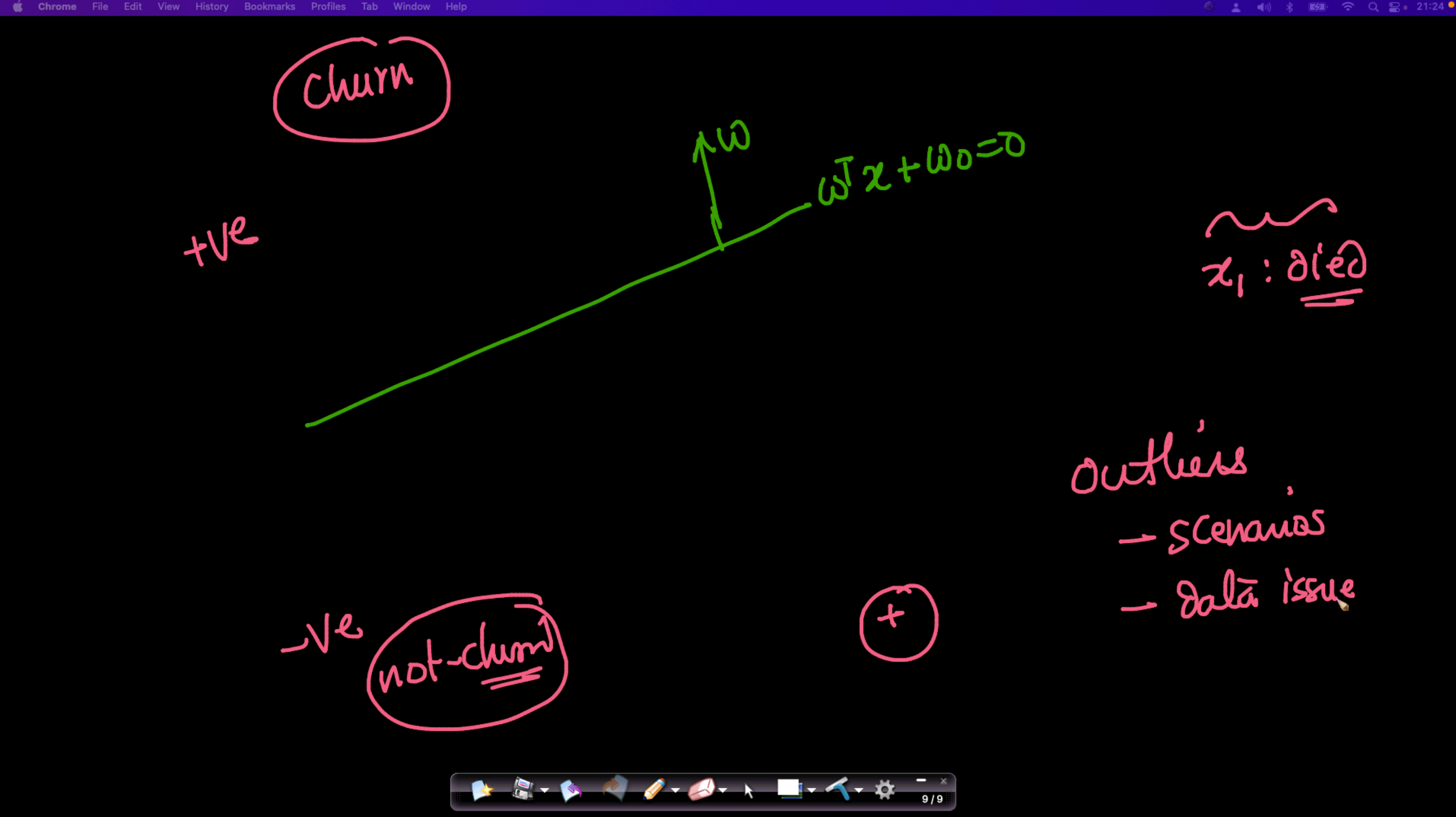
large true-value

tussle: x_2 & all other points

Savior → some impact by far-away points (outliers) on the wrong side of sep $\vec{\pi}$

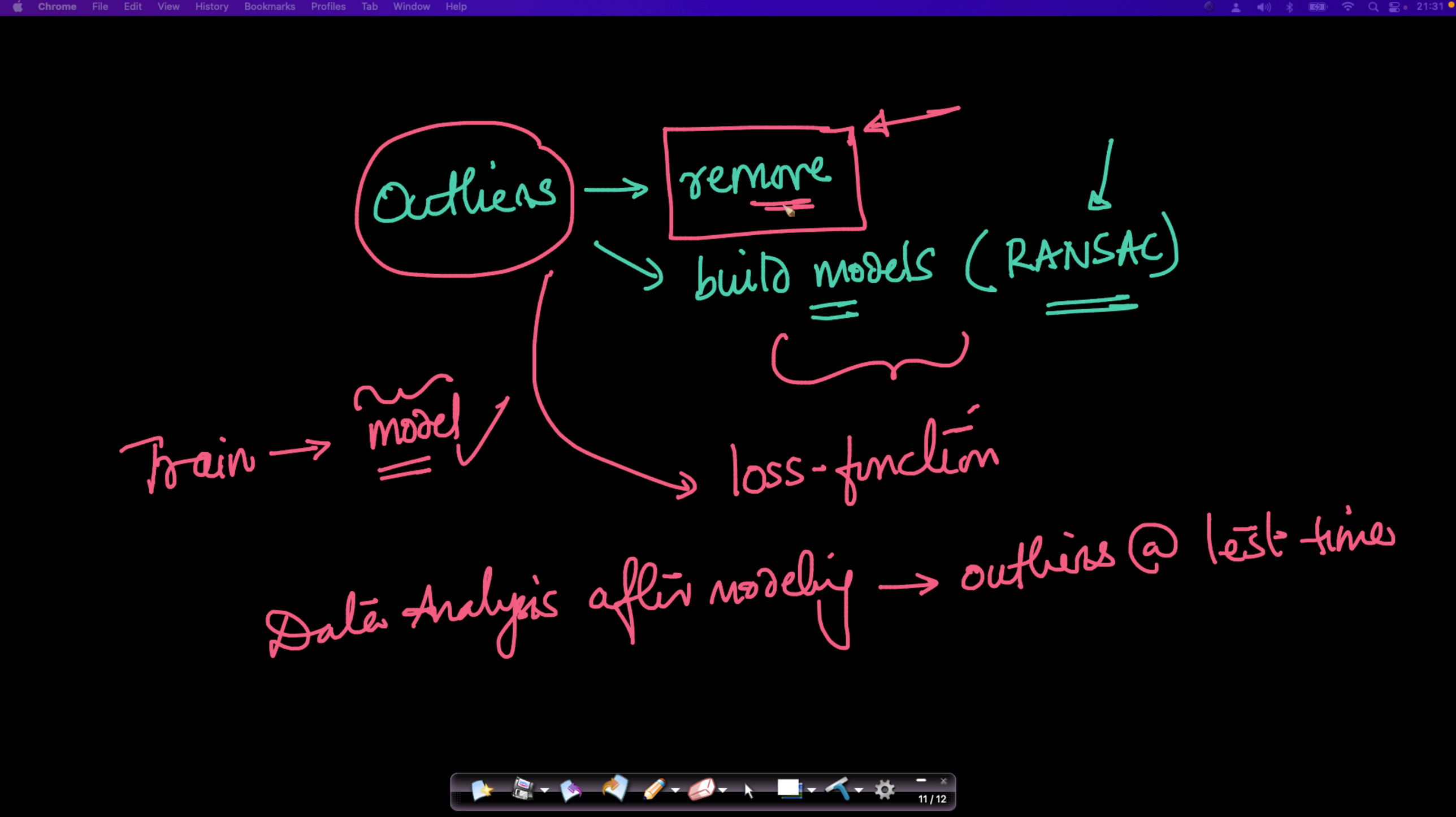
$-\log(\hat{y}_i)$

Some impact of such outliers



(+)

- no-sigmoid → impact of outliers is significant
- sigmoid → some outliers will not effect much
  - → not all


Outliers
remove
build models (RANSAC)
Train → model
loss-function
Data Analysis after modelig → outliers @ least-times

Churn - problem

Class 0 ⟶ 2850

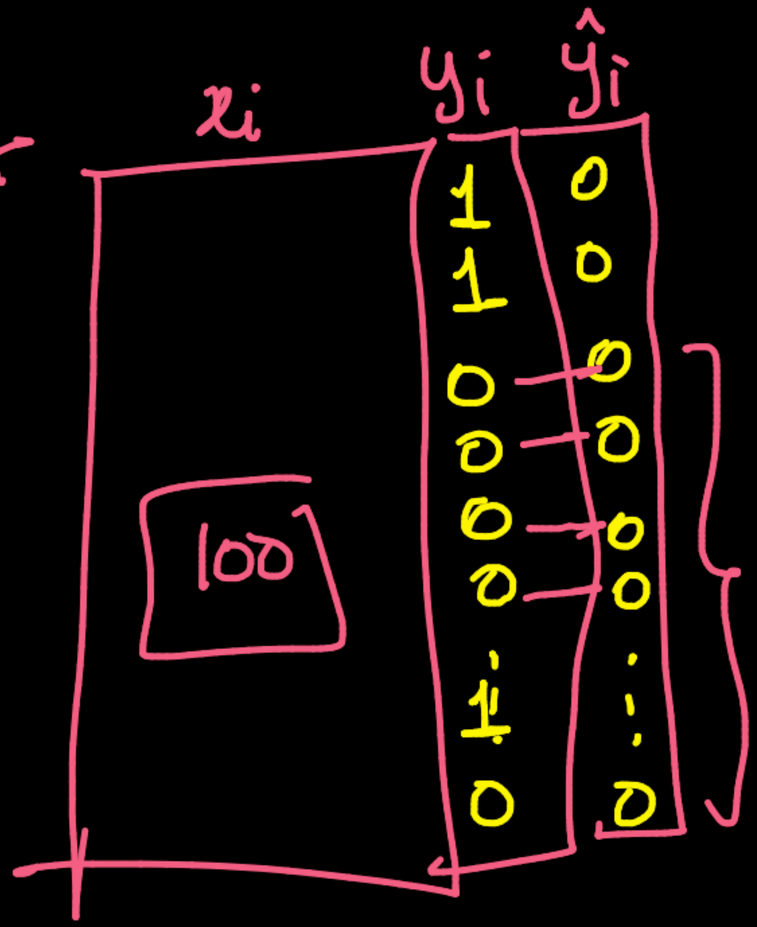
Class 1 ⟶ 483

[imbalanced - data]

~ 85.5% of data class - 0

Q

D Test



~85.5% of y_i's are '0'

$\hat{y}_i$ = logistic Reg(x_i)

$\forall i \quad \hat{y}_i = 0$ [Terrible bad-model]

Incorrect

accuracy: ~85.5%

85/100

imbalanced ⇒ accuracy is a bad metric

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression',
                 LogisticRegression(C=0.0142836737608913))])
```

```
test_score = scaled_lr.score(X_test, y_test)
print(test_score)

y_pred = scaled_lr.predict(X_test)
```

```
0.8560719640179910
```

```
from sklearn.metrics import accuracy_score

print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy : 85.60719640179910%
```

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_pred)
```

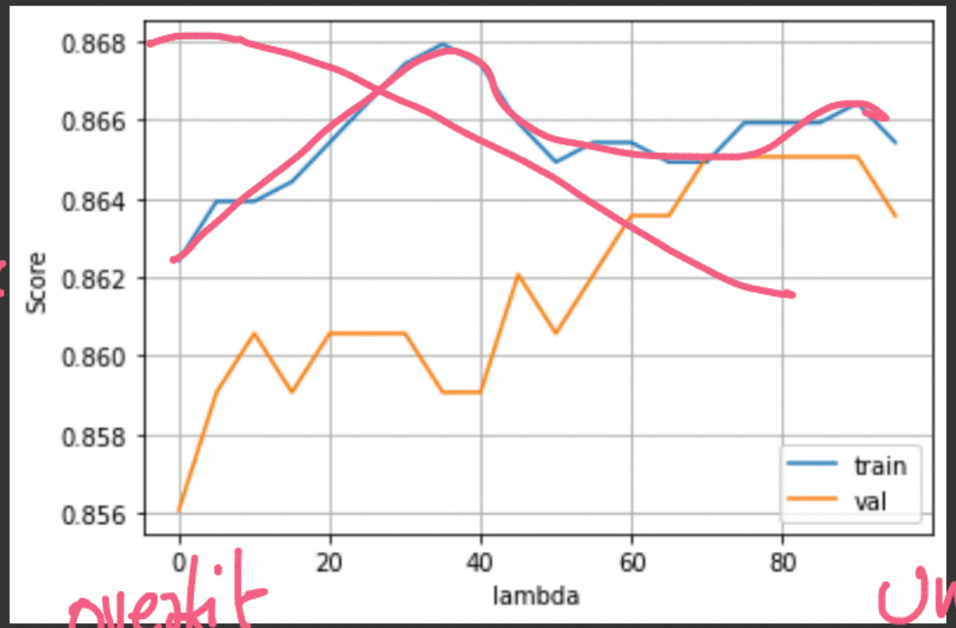
```
array([[560,   6],
       [ 90,  11]])
```

imbalanced data

85.60% as accuracy

same as a terrible model $\hat{y}_i = 0 \ \forall i$

```
[ ] plt.show()
```



acc:

overfit

underfit ✓

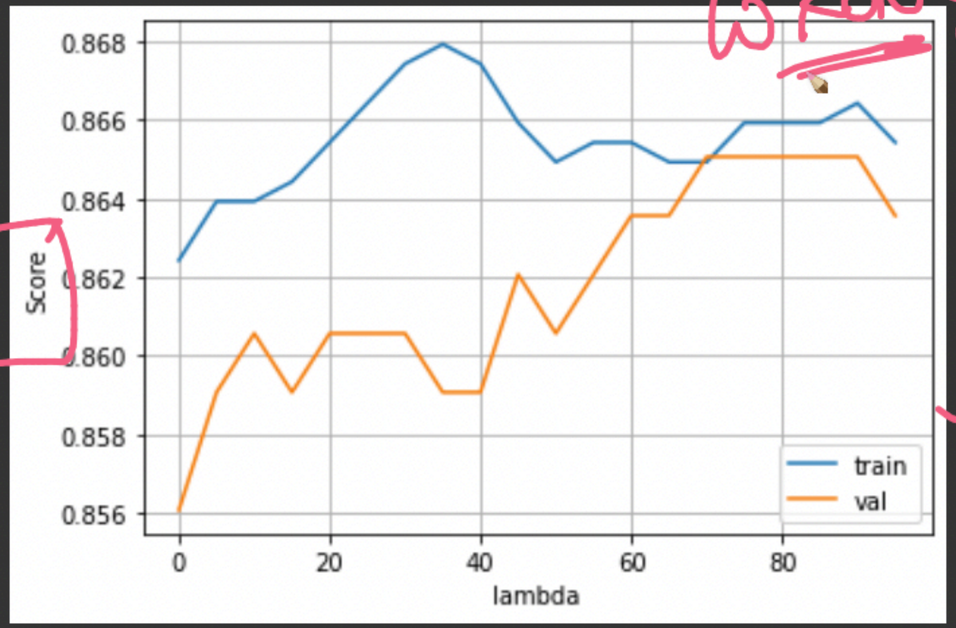
```
[ ] np.argmax(val_scores)
```

```
14
```

```
[ ] val_scores[0]
```

```
0.856071964017991
```

```
[ ] plt.show()
```

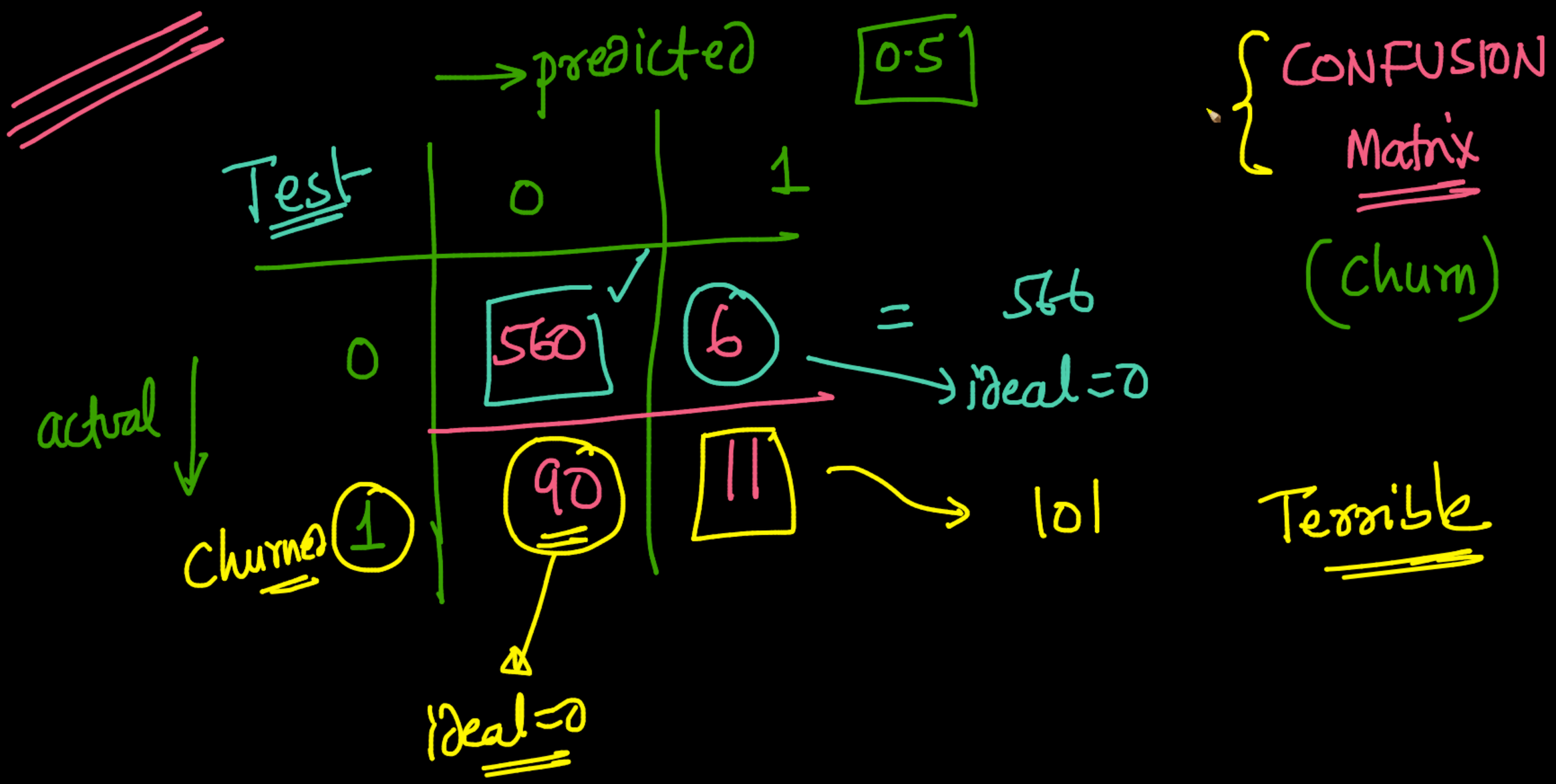


wRONG

acc on imbalanced data

```
[ ] np.argmax(val_scores)

    14
```

```
[ ] val_scores[0]

    0.856071964017991
```


predicted
0.5
Test
0
1
0
560
6
= 566
actual
ideal=0
Churned
1
90
11
lol
Terrible
ideal=0
CONFUSION
Matrix
(Churn)

```
[ ] from sklearn.metrics import accuracy_score

    print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy : 85.60719640179911%
```

```
[ ] from sklearn.metrics import confusion_matrix

    confusion_matrix(y_test, y_pred)
```

```
array([[560,   6],
       [ 90,  11]])
```

```
[ ] # many class-1 points classified as class-0 => useless model
```

```
[ ] # how to fix the model?

    # Hyper-pram tuning
    from sklearn.pipeline import make_pipeline
    train_scores = []
    val_scores = []
    scaler = StandardScaler()
```

```
                    LogisticRegression(C=0.0142836737608913))])
```

```
[ ] test_score = scaled_lr.score(X_test, y_test)
    print(test_score)

    y_pred = scaled_lr.predict(X_test)
```

```
0.856071964017991
```

```
[ ] from sklearn.metrics import accuracy_score

    print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy : 85.6071964017991%
```

```
[ ] from sklearn.metrics import confusion_matrix

    confusion_matrix(y_test, y_pred)
```

```
array([[560,   6],
       [ 90,  11]])
```

```
[ ] # many class-1 points classified as class-0 => useless model
```

```
[ ]  from sklearn.metrics import accuracy_score

     print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy : 85.60719640179911%
```

```
[ ]  from sklearn.metrics import confusion_matrix

     confusion_matrix(y_test, y_pred)
```

```
array([[560,   6],
       [ 90,  11]])
```

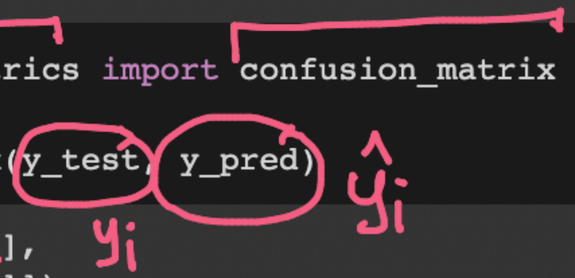
```
[ ]  # many class-1 points classified as class-0 => useless model
```

```
[ ]  # how to fix the model?

     # Hyper-pram tuning
     from sklearn.pipeline import make_pipeline
     train_scores = []
     val_scores = []
     scaler = StandardScaler()
```

```
[ ] from sklearn.metrics import accuracy_score

    print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy : 85.60719640179911%
```

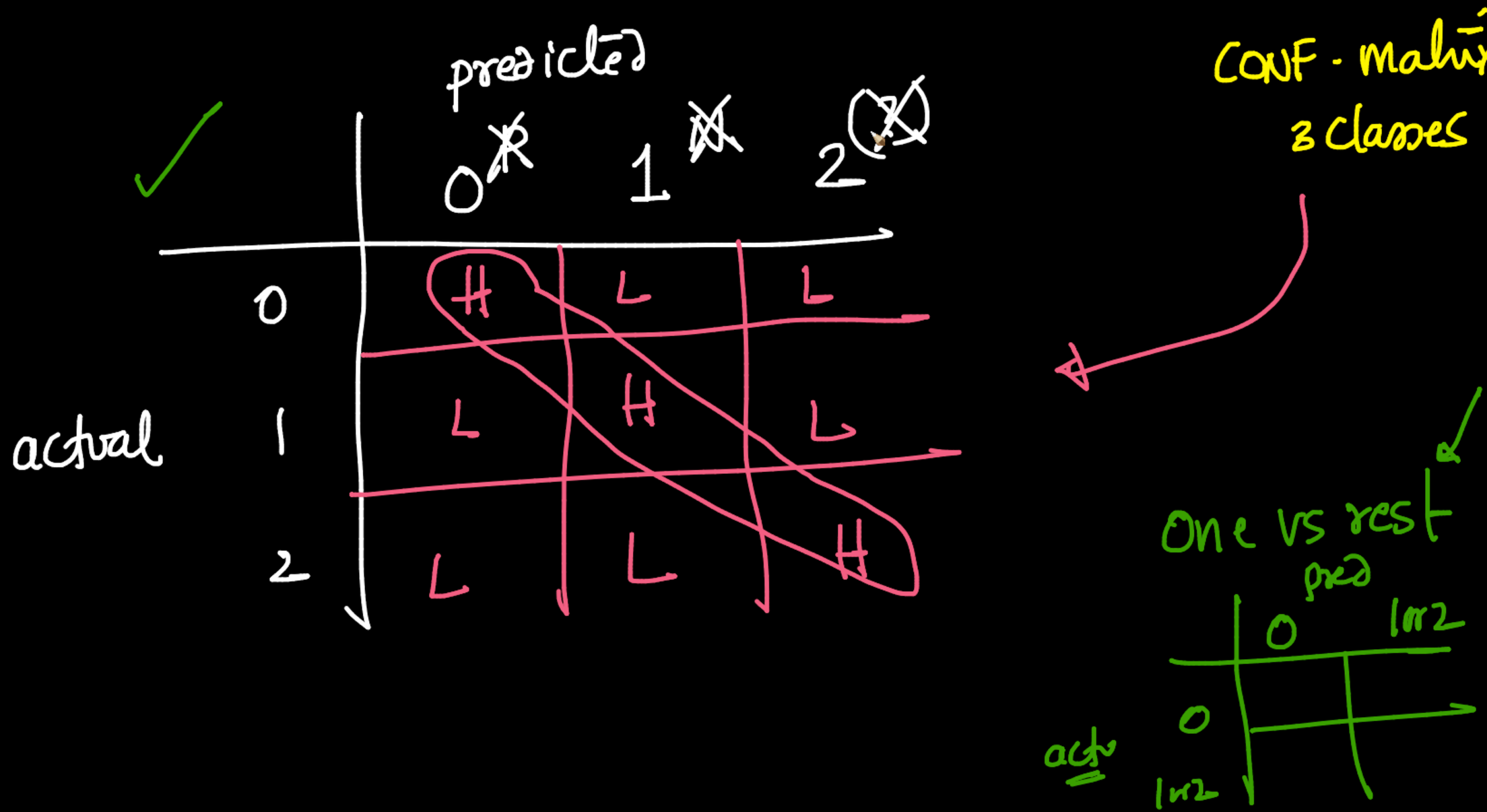
```
[ ] from sklearn.metrics import confusion_matrix

    confusion_matrix(y_test, y_pred)
```

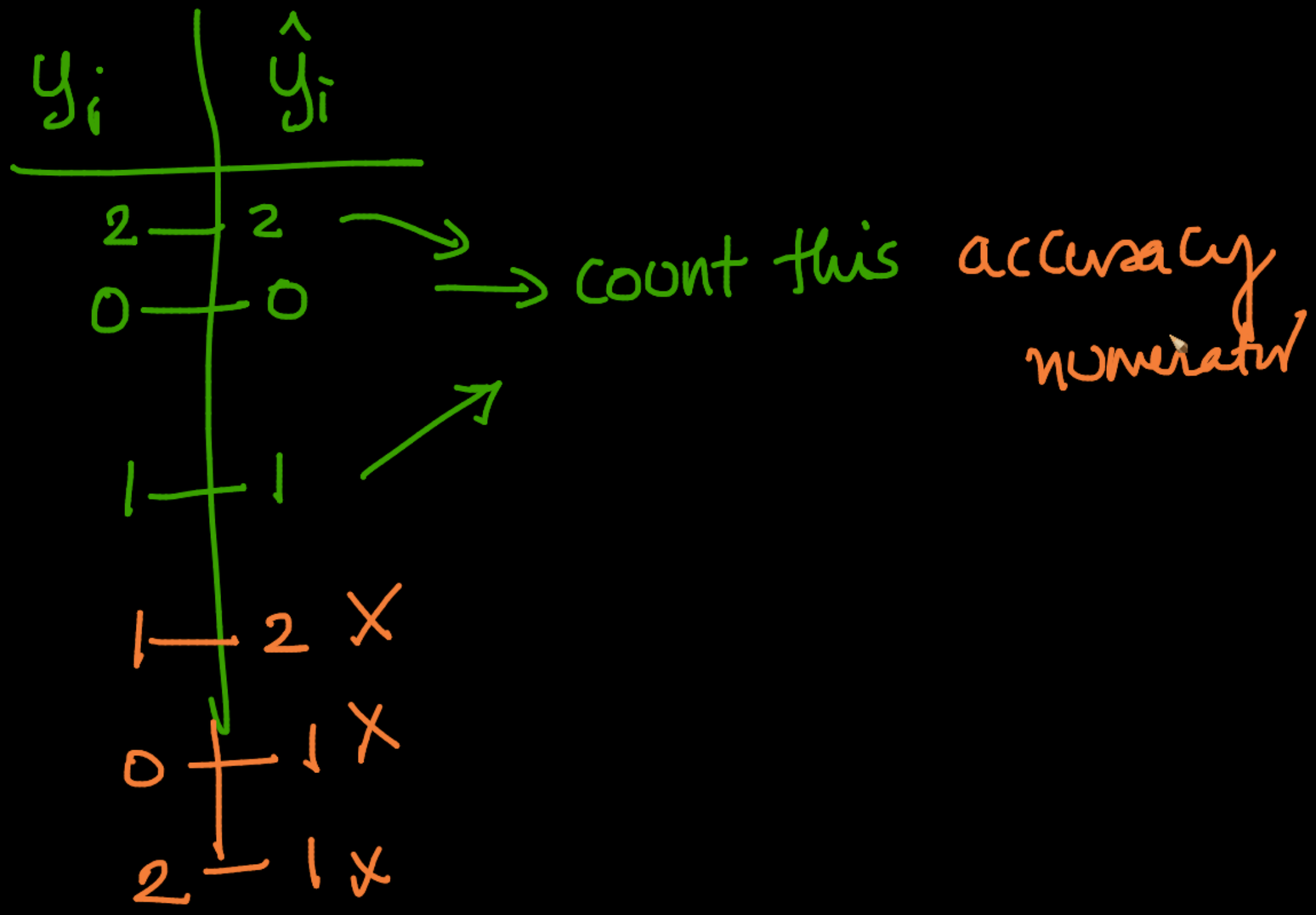



```
array([[560,   6],
       [ 90,  11]])
```

```
[ ] # many class-1 points classified as class-0 => useless model
```

```
[ ] # how to fix the model?

    # Hyper-pram tuning
    from sklearn.pipeline import make_pipeline
    train_scores = []
    val_scores = []
    scaler = StandardScaler()
```


predicted
0
1
2
actual
0
1
2
H
L
L
L
H
L
L
L
H
CONF - Matrix
3 classes
One vs rest
pred
0
1or2
0
1or2
act

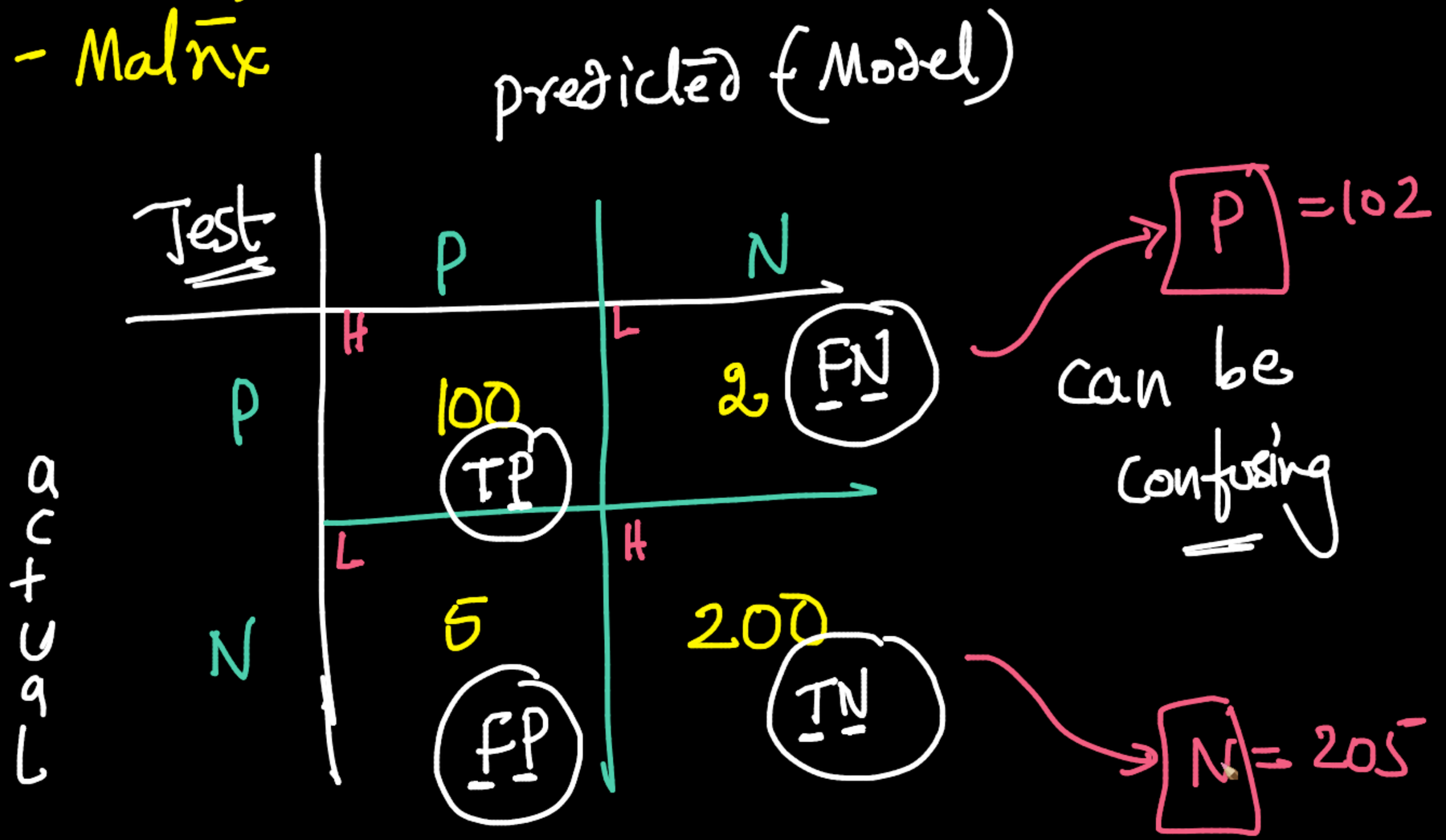

y_i
$\hat{y}_i$
2 2
0 0
count this accuracy numerator
1 1
1 2 X
0 1 X
2 1 X

Confusion-Matrix in sklearn

↳ one vs rest X

CONFUSION - Matrix

binary Classfd:



= case-study

↓

fix

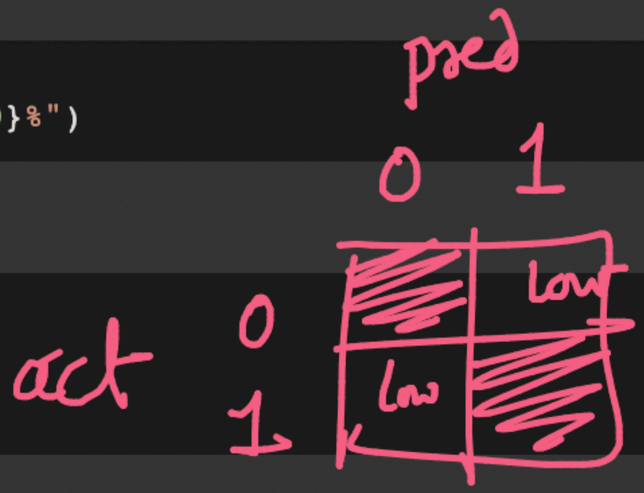
```
[ ]  print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy : 85.60719640179911%
```

```
[ ]  from sklearn.metrics import confusion_matrix

     confusion_matrix(y_test, y_pred)
```

```
array([[560,   6],
       [ 90,  11]])
```

J_v.bad




```
[ ]  # many class-1 points classified as class-0 => useless model
```

```
[ ]  # how to fix the model?

     # Hyper-pram tuning
     from sklearn.pipeline import make_pipeline
     train_scores = []
     val_scores = []
     scaler = StandardScaler()

     for la in np.arange(0.01, 100.0, 5):
       scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la , class_weight={ 0:0.1, 1:0.9 }))
```

```
[ ]  from sklearn.metrics import accuracy_score

     print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy : 85.60719640179911%
```

```
[ ]  from sklearn.metrics import confusion_matrix

     confusion_matrix(y_test, y_pred)
```

```
array([[560,   6],
       [ 90,  11]])
```

0 1

```
[ ]  # many class-1 points classified as class-0 => useless model
```

```
[ ]  # how to fix the model?

     # Hyper-pram tuning
     from sklearn.pipeline import make_pipeline
     train_scores = []
     val_scores = []
     scaler = StandardScaler()

     for la in np.arange(0.01, 100.0, 5):
```

```
[ ]  from sklearn.metrics import accuracy_score

     print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy : 85.60719640179911%
```

```
[ ]  from sklearn.metrics import confusion_matrix

     confusion_matrix(y_test, y_pred)
```

```
array([[560,   6],
       [ 90,  11]])
```

```
[ ]  # many class-1 points classified as class-0 => useless model
```

```
[ ]  # how to fix the model?

     # Hyper-pram tuning
     from sklearn.pipeline import make_pipeline
     train_scores = []
     val_scores = []
     scaler = StandardScaler()

     for la in np.arange(0.01, 100.0, 5):
```

fixes → imbalance correction → weights

sklearn.linear_model.LogisticRegression

class sklearn.linear_model.LogisticRegression(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

Logistic Regression (aka logit, MaxEnt) classifier.

0 : ↓
1 : ↑

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default**. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

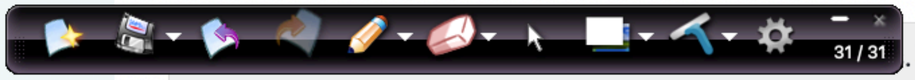
Read more in the User Guide.

Parameters:

penalty : *{'l1', 'l2', 'elasticnet', 'none'}, default='l2'*

Specify the norm of the penalty:

- `'none'`: no penalty is added;
- `'l2'`: add a L2 penalty term and it is the default choice;
- `'l1'`: add a L1 penalty term;
- `'elasticnet'`: both L1 and L2 penalty terms are added.



. See the parameter `solver` below, to know

Chrome File Edit View History Bookmarks Profiles Tab Window Help 21:58

Logistic Regression.ipynb - Co | sklearn.linear_model.LogisticR | Classifier comparison — scikit- | Confusion matrix - Wikipedia

colab.research.google.com/drive/1G_D39wa40kVahULehDS4lWhB_PXhP-xJ#scrollTo=okqWocHMnGXG

Update

+ Code + Text Reconnect

```
[ ]  # how to fix the model?

     # Hyper-pram tuning
     from sklearn.pipeline import make_pipeline
     train_scores = []
     val_scores = []
     scaler = StandardScaler()

     for la in np.arange(0.01, 100.0, 5):
       scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la , class_weight={ 0:0.1, 1:0.9 }))
       scaled_lr.fit(X_train, y_train)
       train_score = scaled_lr.score(X_train, y_train)
       val_score = scaled_lr.score(X_val, y_val)
       train_scores.append(train_score)
       val_scores.append(val_score)

     plt.figure()
     plt.plot(list(np.arange(0.01, 100.0, 5)), train_scores, label="train")
     plt.plot(list(np.arange(0.01, 100.0, 5)), val_scores, label="val")
     plt.legend(loc='lower right')
     plt.xlabel("lambda")
     plt.ylabel("Score")
     plt.grid()
     plt.show()
```

cl-0 : 0.1

cl-1 : 0.9

$\sum_{i=1}^{n} \text{log-loss}_i \times (w_i)$

32 / 32

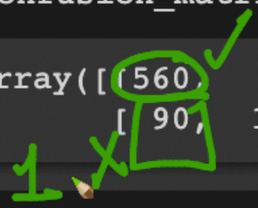
```
0.856071964017991
```

```
[ ] from sklearn.metrics import accuracy_score

    print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy : 85.60719640017991%
```

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_pred)
```

```
array([[560,   6],
       [ 90,  11]])
```



```
[ ] # many class-1 points classified as class-0 => useless model
```

```
[ ] # how to fix the model?

    # Hyper-pram tuning
    from sklearn.pipeline import make_pipeline
    train_scores = []
    val_scores = []
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 21:59

Logistic Regression.ipynb - Co | sklearn.linear_model.LogisticR | Classifier comparison — scikit- | Confusion matrix - Wikipedia

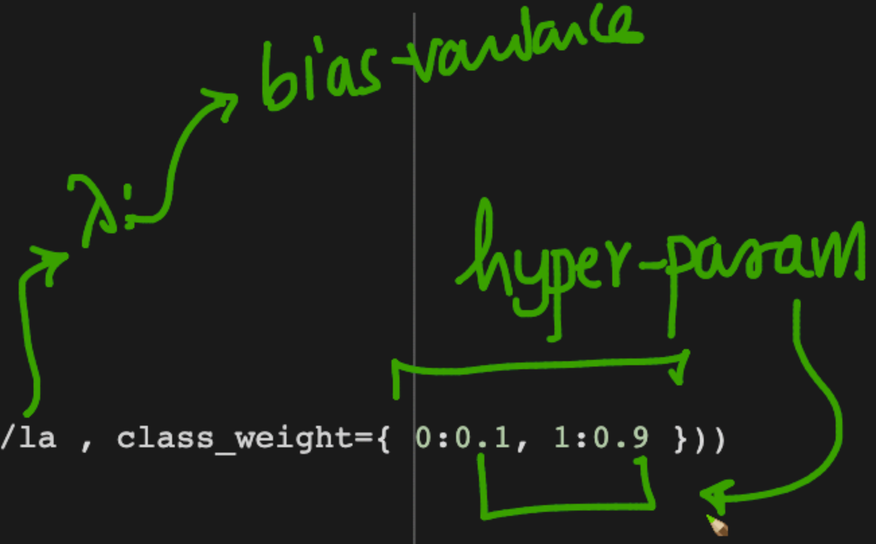
colab.research.google.com/drive/1G_D39wa40kVahULehDS4lWhB_PXhP-xJ#scrollTo=okqWocHMnGXG

Update

+ Code + Text Reconnect

```
[ ] # many class-1 points classified as class-0 => useless model
```

```
[ ] # how to fix the model?

    # Hyper-pram tuning
    from sklearn.pipeline import make_pipeline
    train_scores = []
    val_scores = []
    scaler = StandardScaler()

    for la in np.arange(0.01, 100.0, 5):
      scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la , class_weight={ 0:0.1, 1:0.9 }))
      scaled_lr.fit(X_train, y_train)
      train_score = scaled_lr.score(X_train, y_train)
      val_score = scaled_lr.score(X_val, y_val)
      train_scores.append(train_score)
      val_scores.append(val_score)

    plt.figure()
    plt.plot(list(np.arange(0.01, 100.0, 5)), train_scores, label="train")
    plt.plot(list(np.arange(0.01, 100.0, 5)), val_scores, label="val")
    plt.legend(loc='lower right')
    plt.xlabel("lambda")
    plt.ylabel("Score")
    plt.grid()
```



34 / 34

```
[ ] # many class-1 points classified as class-0 => useless model
```

```
[ ] # how to fix the model?

    # Hyper-pram tuning
    from sklearn.pipeline import make_pipeline
    train_scores = []
    val_scores = []
    scaler = StandardScaler()

    for la in np.arange(0.01, 100.0, 5):
      scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la , class_weight={ 0:0.1, 1:0.9 }))
      scaled_lr.fit(X_train, y_train)
      train_score = scaled_lr.score(X_train, y_train)
      val_score = scaled_lr.score(X_val, y_val)
      train_scores.append(train_score)
      val_scores.append(val_score)

    plt.figure()
    plt.plot(list(np.arange(0.01, 100.0, 5)), train_scores, label="train")
    plt.plot(list(np.arange(0.01, 100.0, 5)), val_scores, label="val")
    plt.legend(loc='lower right')
    plt.xlabel("lambda")
    plt.ylabel("Score")
    plt.grid()
```

15:85

```
[ ] # many class-1 points classified as class-0 => useless model
```

```
[ ] # how to fix the model?

    # Hyper-pram tuning
    from sklearn.pipeline import make_pipeline
    train_scores = []
    val_scores = []
    scaler = StandardScaler()

    for la in np.arange(0.01, 100.0, 5):
      scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la , class_weight={ 0:0.1, 1:0.9 }))
      scaled_lr.fit(X_train, y_train)
      train_score = scaled_lr.score(X_train, y_train)
      val_score = scaled_lr.score(X_val, y_val)
      train_scores.append(train_score)
      val_scores.append(val_score)

    plt.figure()
    plt.plot(list(np.arange(0.01, 100.0, 5)), train_scores, label="train")
    plt.plot(list(np.arange(0.01, 100.0, 5)), val_scores, label="val")
    plt.legend(loc='lower right')
    plt.xlabel("lambda")
    plt.ylabel("Score")
    plt.grid()
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 21:59

Logistic Regression.ipynb - Co | sklearn.linear_model.LogisticR | Classifier comparison — scikit- | Confusion matrix - Wikipedia

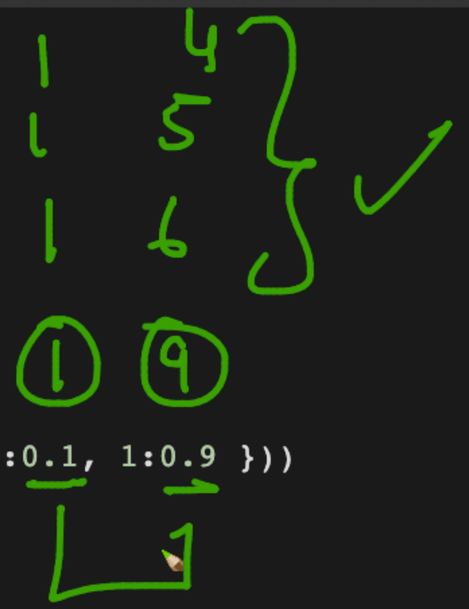
colab.research.google.com/drive/1G_D39wa40kVahULehDS4lWhB_PXhP-xJ#scrollTo=okqWocHMnGXG

Update

+ Code + Text Reconnect

```
[ ] # many class-1 points classified as class-0 => useless model
```

```
[ ] # how to fix the model?

    # Hyper-pram tuning
    from sklearn.pipeline import make_pipeline
    train_scores = []
    val_scores = []
    scaler = StandardScaler()

    for la in np.arange(0.01, 100.0, 5):
      scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la , class_weight={ 0:0.1, 1:0.9 }))
      scaled_lr.fit(X_train, y_train)
      train_score = scaled_lr.score(X_train, y_train)
      val_score = scaled_lr.score(X_val, y_val)
      train_scores.append(train_score)
      val_scores.append(val_score)

    plt.figure()
    plt.plot(list(np.arange(0.01, 100.0, 5)), train_scores, label="train")
    plt.plot(list(np.arange(0.01, 100.0, 5)), val_scores, label="val")
    plt.legend(loc='lower right')
    plt.xlabel("lambda")
    plt.ylabel("Score")
    plt.grid()
```

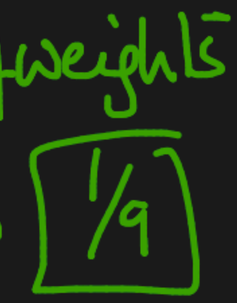


37 / 37

```
[ ] # many class-1 points classified as class-0 => useless model
```

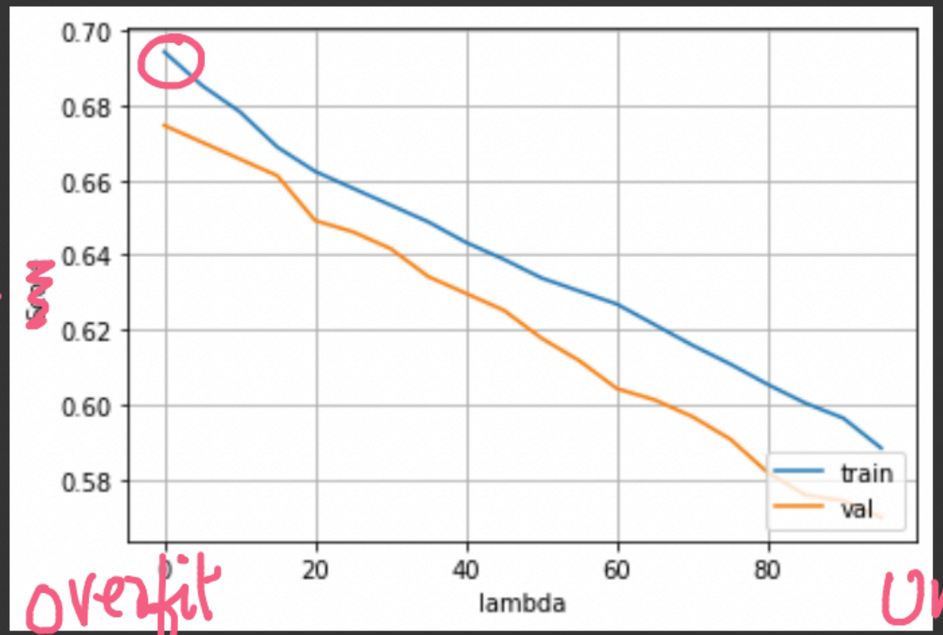
```
[ ] # how to fix the model?

    # Hyper-pram tuning
    from sklearn.pipeline import make_pipeline
    train_scores = []
    val_scores = []
    scaler = StandardScaler()

    for la in np.arange(0.01, 100.0, 5):
      scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la , class_weight={ 0:0.1, 1:0.9 }))
      scaled_lr.fit(X_train, y_train)
      train_score = scaled_lr.score(X_train, y_train)
      val_score = scaled_lr.score(X_val, y_val)
      train_scores.append(train_score)
      val_scores.append(val_score)

    plt.figure()
    plt.plot(list(np.arange(0.01, 100.0, 5)), train_scores, label="train")
    plt.plot(list(np.arange(0.01, 100.0, 5)), val_scores, label="val")
    plt.legend(loc='lower right')
    plt.xlabel("lambda")
    plt.ylabel("Score")
    plt.grid()
```

ex: tune ratio of weights



```
plt.figure()
plt.plot(list(np.arange(0.01, 100.0, 5)), train_scores, label="train")
plt.plot(list(np.arange(0.01, 100.0, 5)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("Score")
plt.grid()
plt.show()
```

→ log-loss



acc

overfit

Underfit

Chrome File Edit View History Bookmarks Profiles Tab Window Help 22:02

Logistic Regression.ipynb - Co | sklearn.linear_model.LogisticR | Classifier comparison — scikit- | Confusion matrix - Wikipedia

colab.research.google.com/drive/1G_D39wa40kVahULehDS4lWhB_PXhP-xJ#scrollTo=TLN4Igtpu2vS

Update

+ Code + Text

RAM Disk

[119]
0s

```
print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

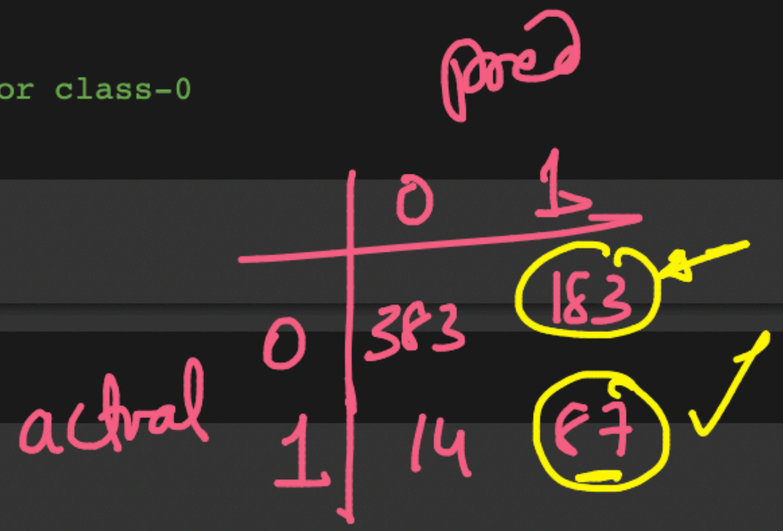
```
Accuracy : 70.4647676161919%
```

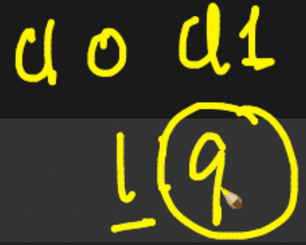
0s

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_pred)
# Better for class-1, but worse for class-0
# Any ideas on how to solve?
```

```
array([[383, 183],
       [ 14,  87]])
```

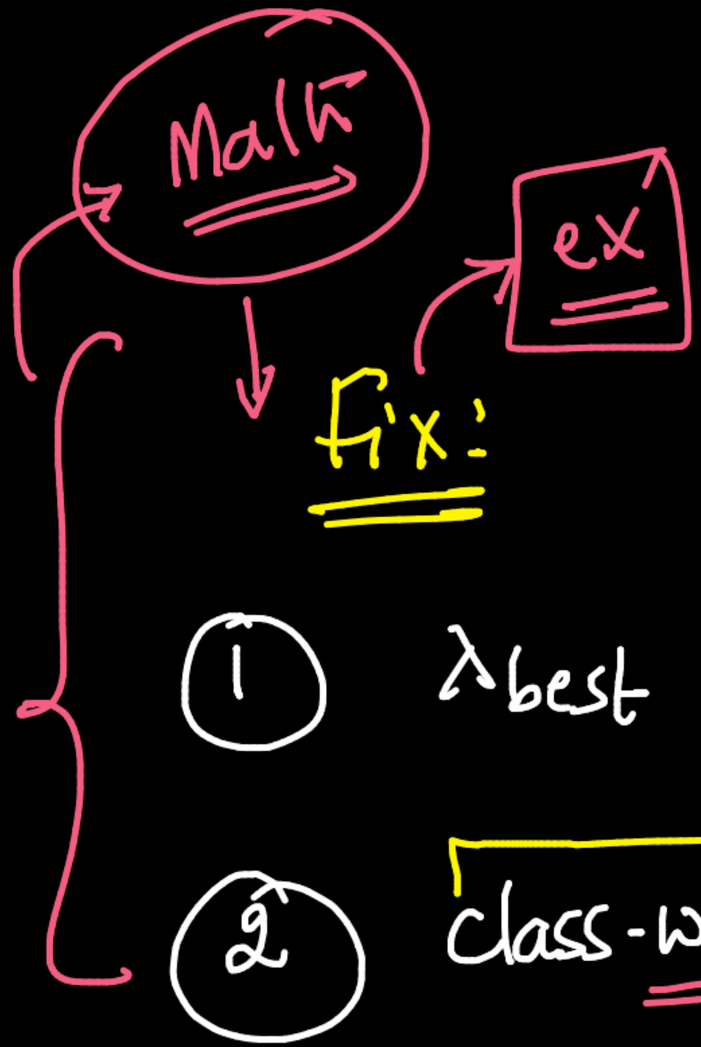
```
[ ]
```

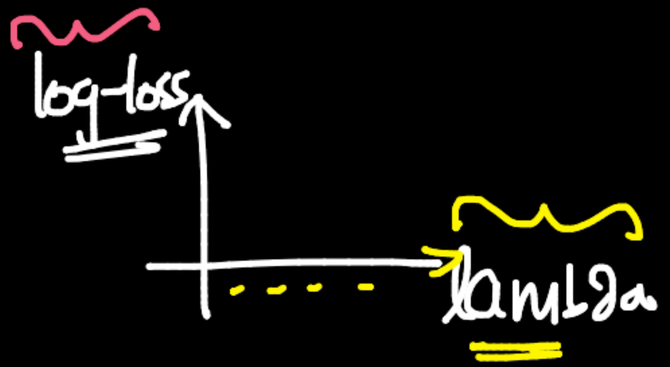


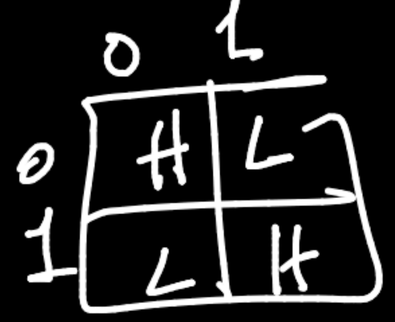


40 / 40

$\frac{1}{2}; \frac{1}{5}; \frac{1}{6}; \frac{1}{9}; \frac{1}{10}; \ldots$



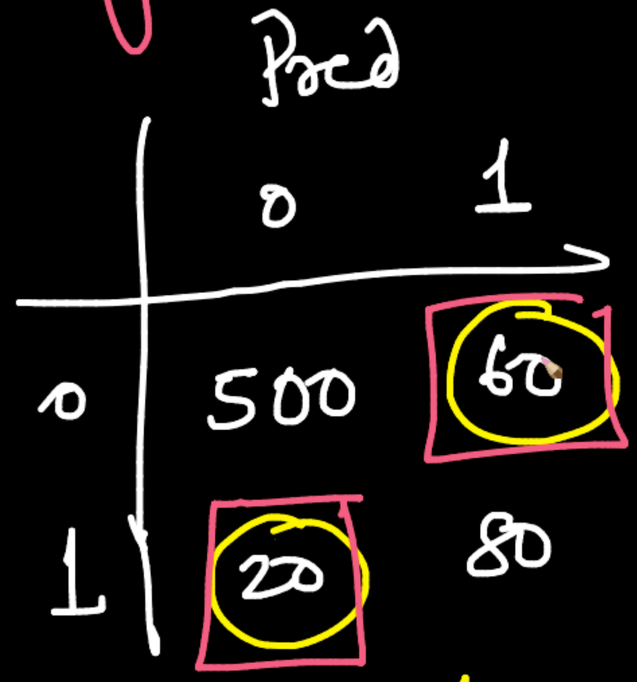




Fix - everything → not-seperable linearly

- non-linear

{f1 f2 ... fd

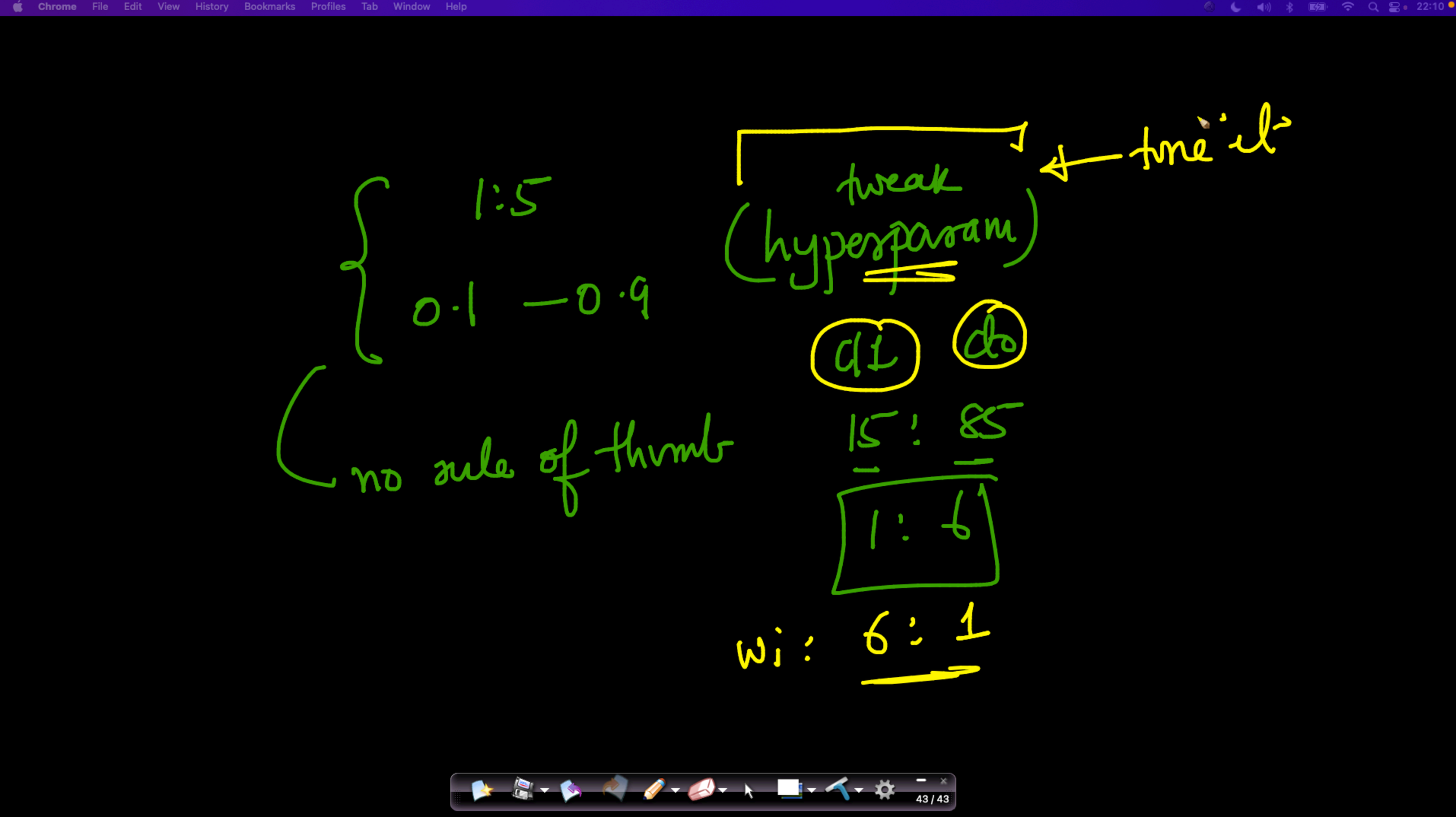


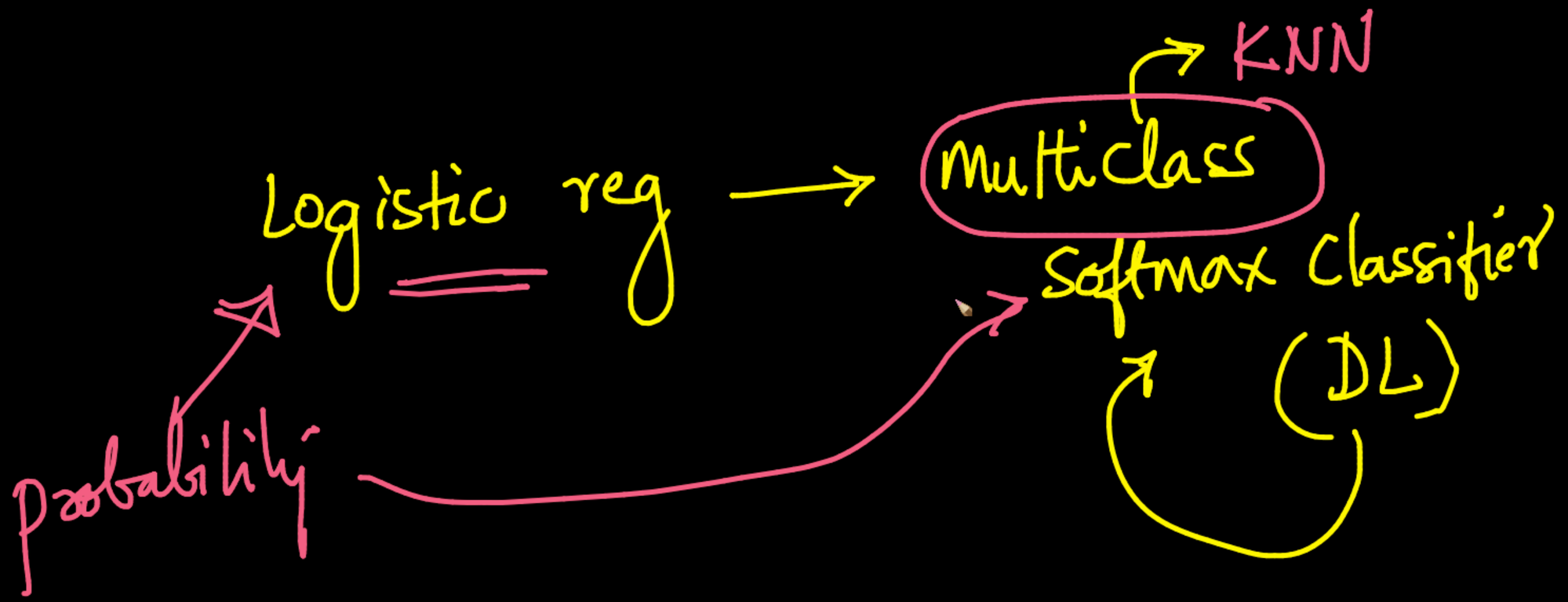
Linear model

How can you further improve

- outliers
- look @ FP & FN

→ 0.5 to 0.4 ...
hyper-param


tweak
(hyperparam)
tune it
1:5
0.1 — 0.9
no rule of thumb
d1
d0
15 : 85
1 : 6
wi : 6 : 1


KNN
Logistic reg
Multiclass
Softmax Classifier
(DL)
probability

predicted

actual	P	N
P	TP (large)	FN (small)
N	FP (small)	TN

→ { Information retrieval:

P → P'

TP TP FP

(IR) Algo → Model

P N N N
N P N N
N N P N
P

✓ [Precision: $\frac{2}{3} = \frac{TP}{TP+FP}$

[Recall: $\frac{2}{4} = \frac{TP}{TP+FN}$

Example [edit]

Given a sample of 12 individuals, 8 that have been diagnosed with cancer and 4 that are cancer-free, where individuals with cancer belong to class 1 (positive) and non-cancer individuals belong to class 0 (negative), we can display that data as follows:

Individual Number	1	2	3	4	5	6	7	8	9	10	11	12
Actual Classification	1	1	1	1	1	1	1	1	0	0	0	0

Assume that we have a classifier that distinguishes between individuals with and without cancer in some way, we can take the 12 individuals and run them through the classifier. The classifier then makes 9 accurate predictions and misses 3: 2 individuals with cancer wrongly predicted as being cancer-free (sample 1 and 2), and 1 person without cancer that is wrongly predicted to have cancer (sample 9).

Individual Number	1	2	3	4	5	6	7	8	9	10	11	12
Actual Classification	1	1	1	1	1	1	1	1	0	0	0	0
Predicted Classification	0	0	1	1	1	1	1	1	1	0	0	0

Notice, that if we compare the actual classification set to the predicted classification set, there are 4 different outcomes that could result in any particular column. One, if the actual classification is positive and the predicted classification is positive (1,1), this is called a true positive result because the positive sample was correctly identified by the classifier. Two, if the actual classification is positive and the predicted classification is negative (1,0), this is called a false negative result because the positive sample is incorrectly identified by the classifier as being negative. Thi

A test result which wrongly indicates that a particular condition or attribute is absent

sensitivity, recall, hit rate, or true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

specificity, selectivity or true negative rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

precision or positive predictive value (PPV)

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}$$

negative predictive value (NPV)

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}} = 1 - \mathrm{FOR}$$

miss rate or false negative rate (FNR)

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$

fall-out or false positive rate (FPR)

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

false discovery rate (FDR)

$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} = 1 - \mathrm{PPV}$$

false omission rate (FOR)

$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TN}} = 1 - \mathrm{NPV}$$

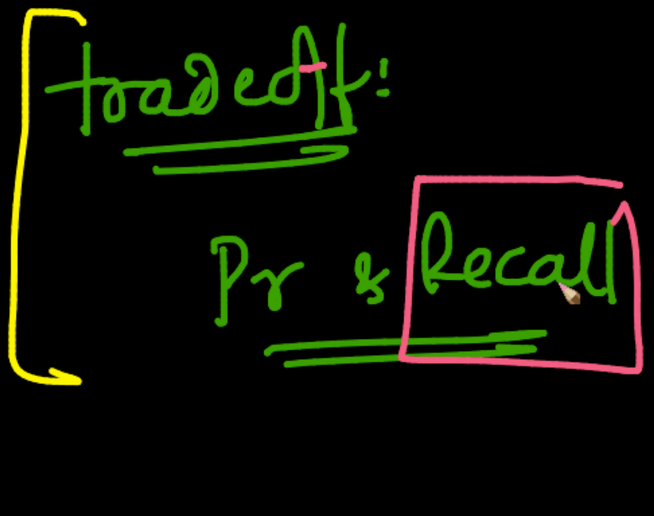
Positive likelihood ratio (LR+)

$$\mathrm{LR+} = \frac{\mathrm{TPR}}{\mathrm{FPR}}$$

e likelihood ratio (LR-)

Precision: ↑

Recall: ↑



e.g: covid diagnosis

RT-PCR: Recall → 99.999%
V-high

99% ← high Precision

Precision : $TP/(TP+FP)$

Recall : $TP/(TP+FN)$

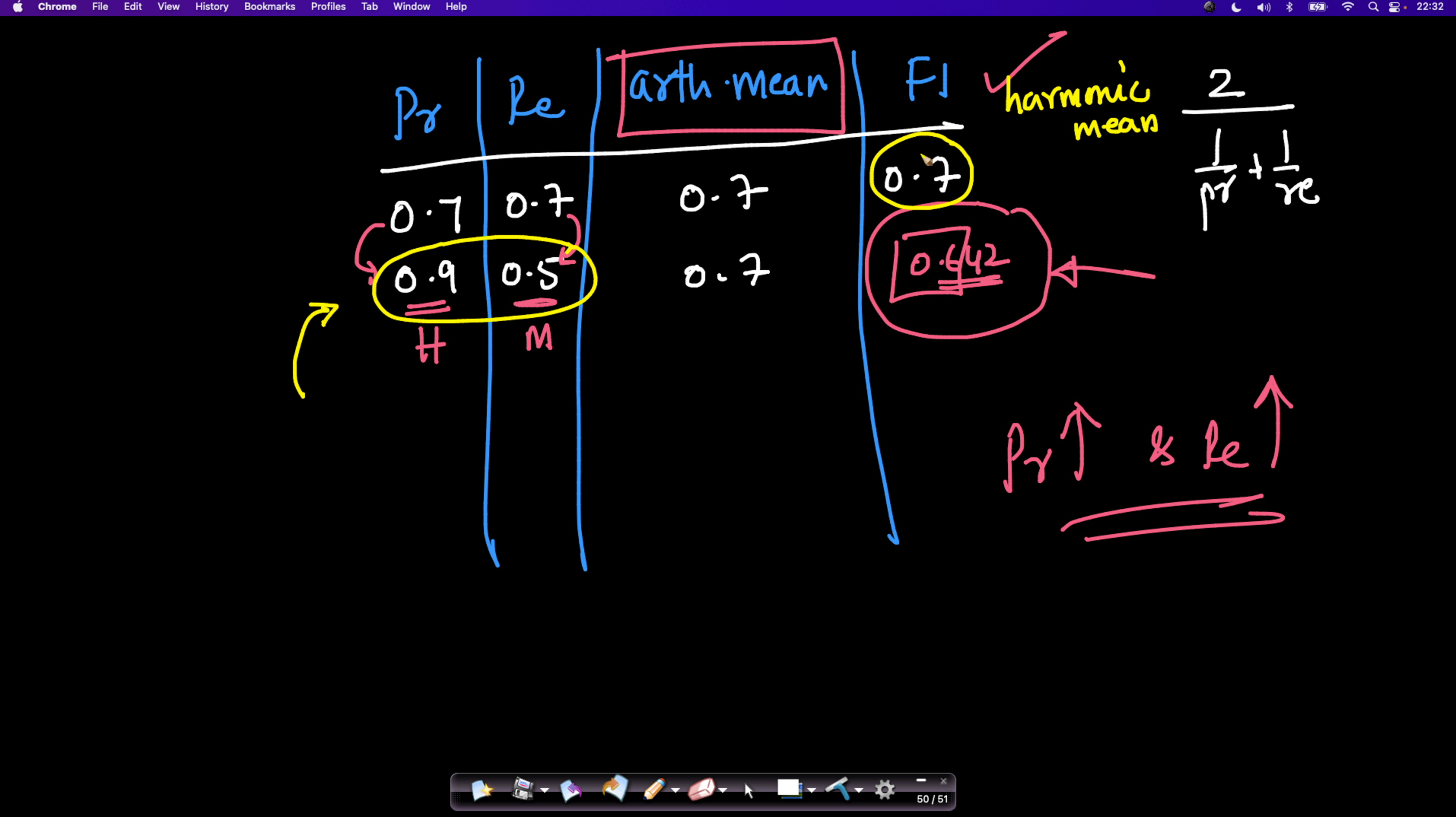
→ one number metric ↑ when Pr ↑ & re ↑

$\begin{cases} 0-1 \\ 0-1 \end{cases}$

$$avg = \frac{\uparrow Pr + Re\uparrow}{2} = \text{arithmetic mean}\uparrow \quad \alpha\, Pr \quad \alpha\, Re$$

$\uparrow\uparrow$ $\begin{cases} F1: \\ Score \end{cases}$ $\dfrac{2}{\left(\frac{1}{\uparrow Pr} + \frac{1}{Re\uparrow}\right)}$: harmonic mean

$\begin{cases} Re \propto F1 \\ Pr \propto F1 \end{cases}$


Pr
Re
arth mean
F1
harmonic mean
2 / (1/Pr + 1/Re)
0.7
0.7
0.7
0.7
0.9
0.5
0.7
0.642
H
M
Pr ↑ & Re ↑
50 / 51

$$\left\{ F1: \quad \frac{2}{\frac{1}{pr} + \frac{1}{re}} = 1 \right.$$

max F1

$$\frac{1}{p_1} \quad \frac{1}{1}$$

$pr \propto F1$

$Re \propto F1$

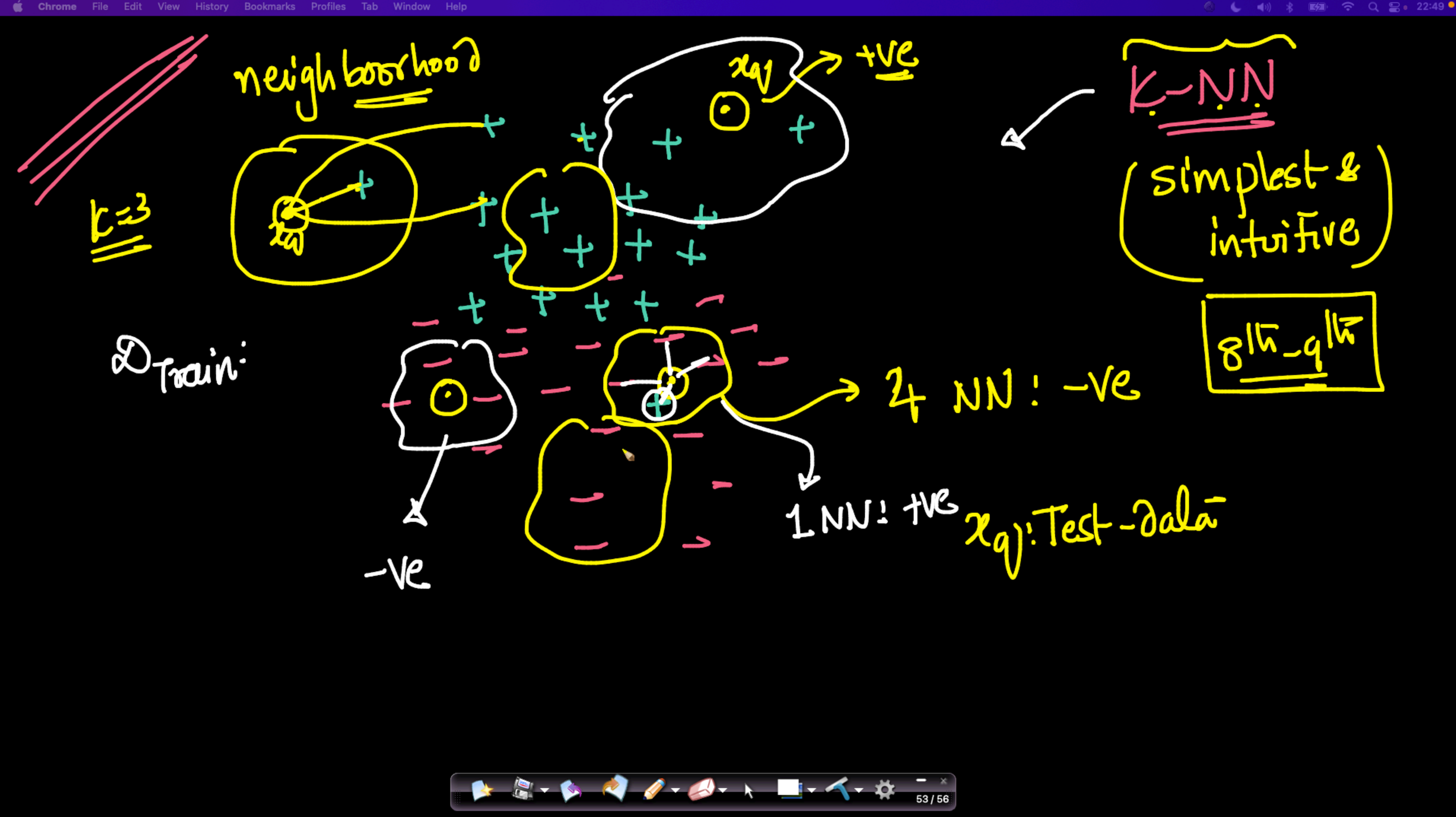
Next-class: — ROC; AUC

PR-curves

→ odds-ratio

→ imbalanced data

KNN


neighboorhood
x_q
+ve
K-NN
simplest & intuitive
K=3
D_{train}:
$\|a-q\|$
4 NN : -ve
1 NN : +ve
x_q : Test-data
-ve

Inputs : D_{Train} , x_q , K = odd : 5

d = [

[for each x_i in D_{Train}

- compute $dist(x_q, x_i)$
- append to d

- Sort the d[...]
- pick K nearest neighbors
- Majority vote & return Class label

+ve → 3

-ve → 2

$$\|x_q - x_i\| = \sqrt{\sum_{j=1}^{d} (x_{qj} - x_{ij})^2}$$

$K = 4$

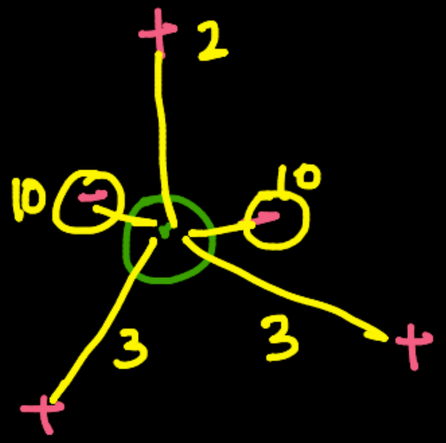
+ve : 2
-ve : 2
] → bst

Weighted K-NN

1. $K = 11$
2. $K = 5$;

-ve: 20
+ve: 8
} ✓

$K = 5$: odd



{ +ve : 3
-ve : 2

Weight based on

$\frac{1}{dist}$

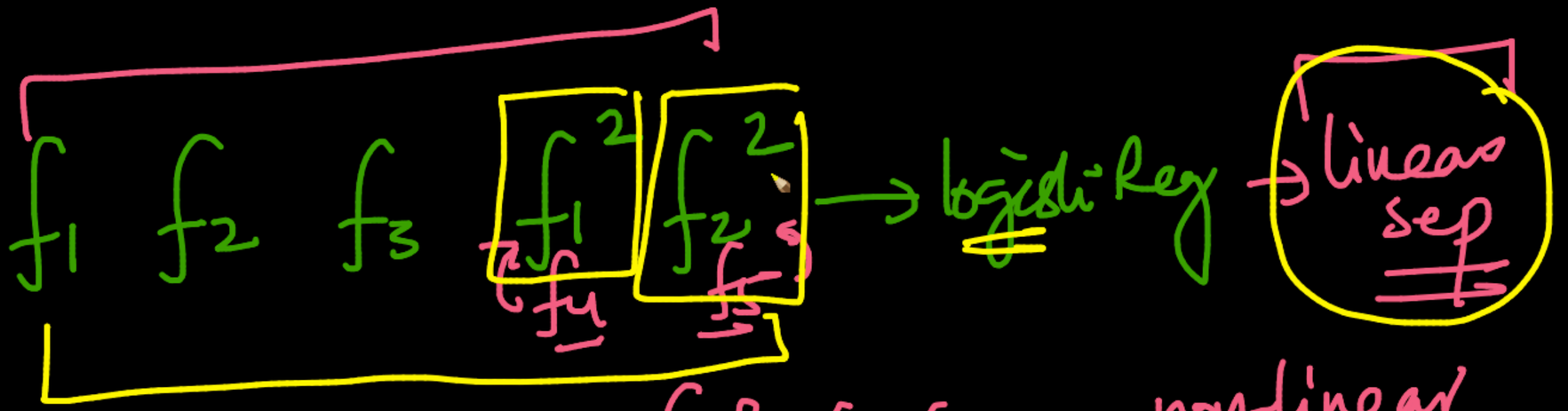
$w \propto \frac{1}{dist}$

Assumption KNN: → Neighborhoods are homogenous most of the time

Assumption logistic Regr: → Linearly Seperability

Good

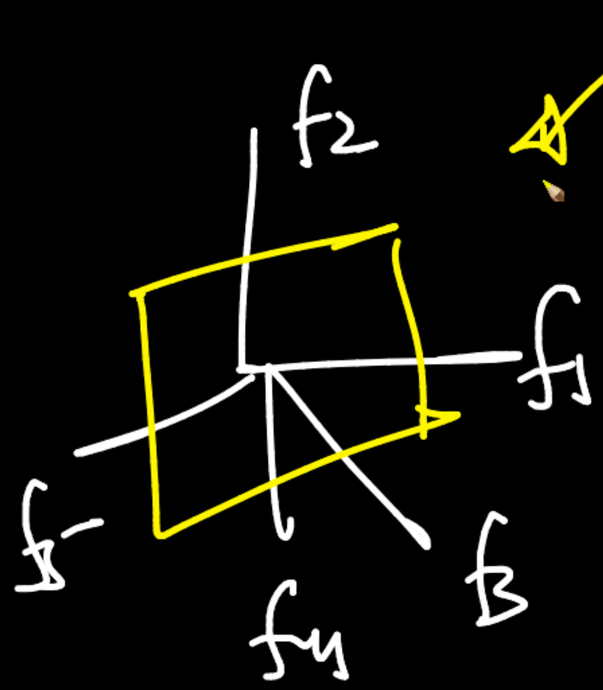
f_1 f_2 f_3 → logistic reg → /



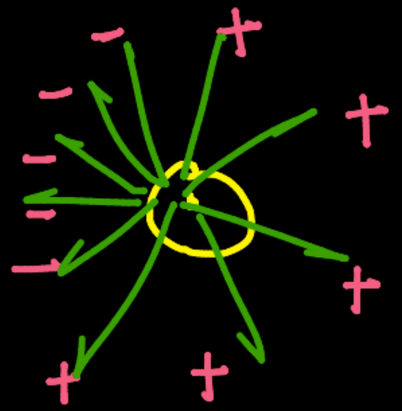
{ f_1, f_2, f_3 → non-linear Sep

f_1 f_2 f_3 $f_4 = f_1^2$ $f_5 = f_2^2$

w_1 w_2 w_3 w_4 w_5



$$w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + w_4 x_{i4} + w_5 x_{i5} + w_0 = 0$$



k = 5

ties

+ve

-ve

Q

K-NN

disadv

$D_{train} \rightarrow n$ large pts ; d → 100's dim

100 M

Space-complx → $O(n \cdot d + \cancel{K})$

(Test) Time complx to run K-NN on x_q

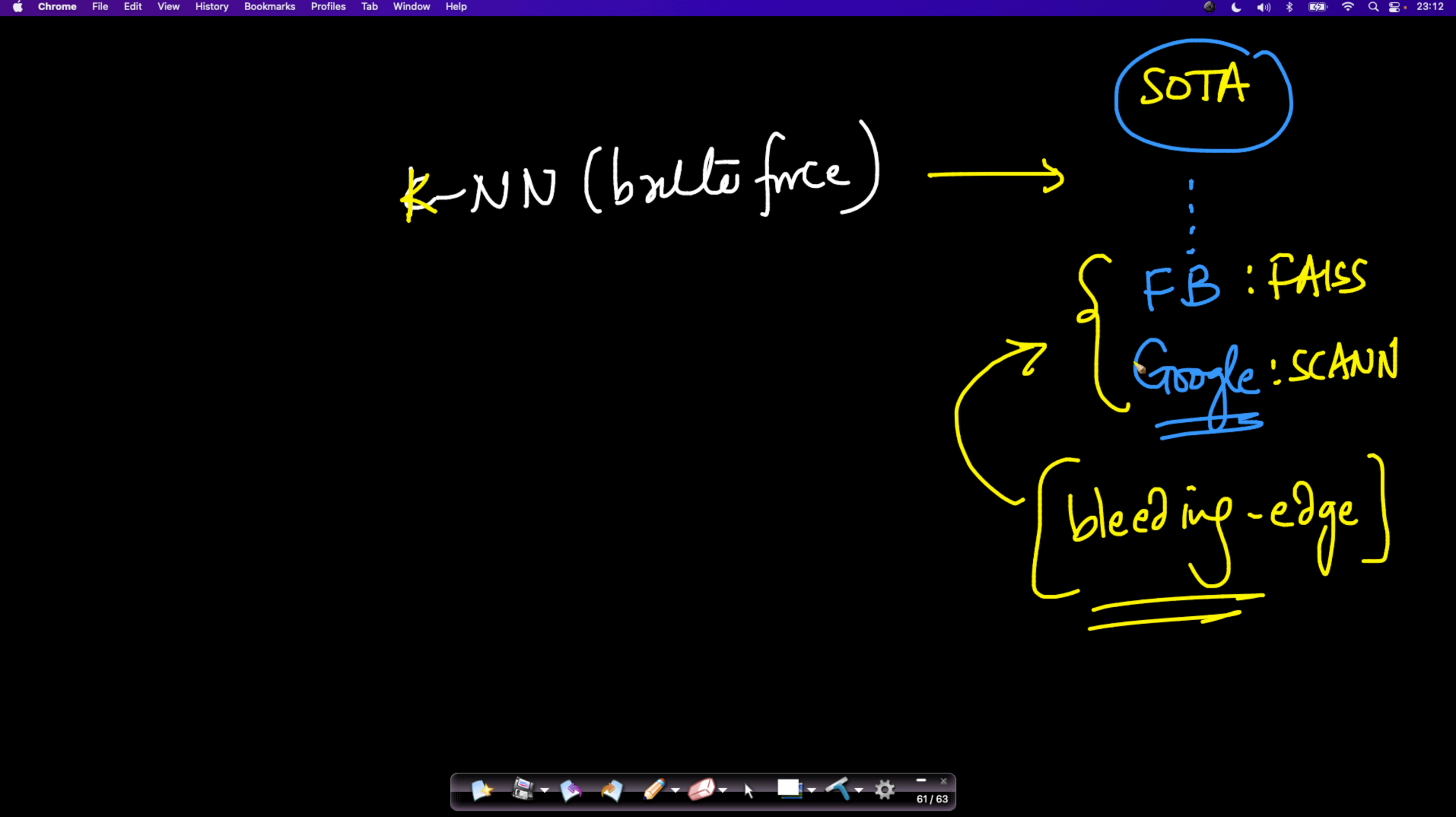
$x_q \rightarrow$

- $dist(x_i, x_q)\ \forall i$
- n (g_i)'s : sort
- K-top-pts

BRUTE-FORCE

$O(n \cdot d + n \log n + \cancel{K})$ (ms)

$O(n \cdot (d + \log n))$ small



Q

✓ logistic reg: d-dim data n-train pts

indep of n → 100

x_q

$\sigma(w^T x_q + w_0)$

d-mul

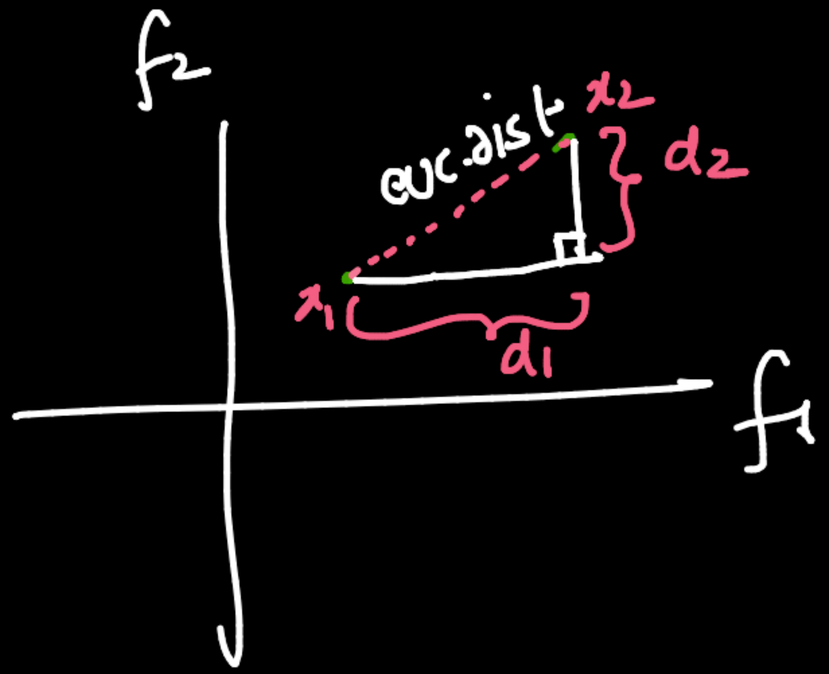
time-complx : O(d)

Space-complx: O(d)

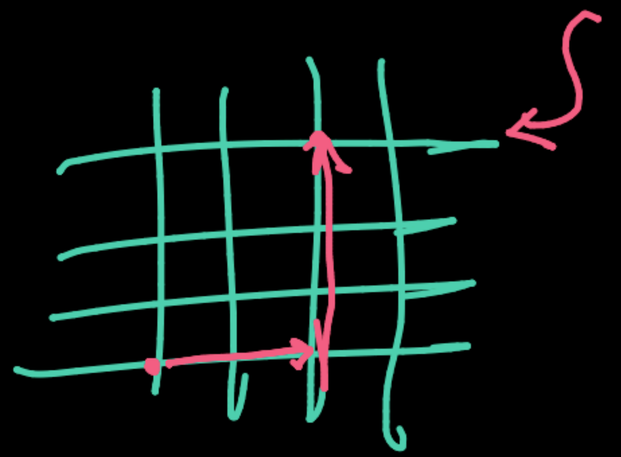
euc. dist $(x_1, x_2) =$ $\left[\sum_{j=1}^{d} |(x_{1j} - x_{2j})|^2 \right]^{1/2}$

- does not work when d is high (curse of dim) → Math (later) ...

$$\text{Manhattan dist}(x_1, x_2) = \left(\sum_{j=1}^{d} |x_{1j} - x_{2j}|^{\frac{1}{r}} \right)^{\frac{1}{r}}$$



$d_1 + d_2$: manhattan dist



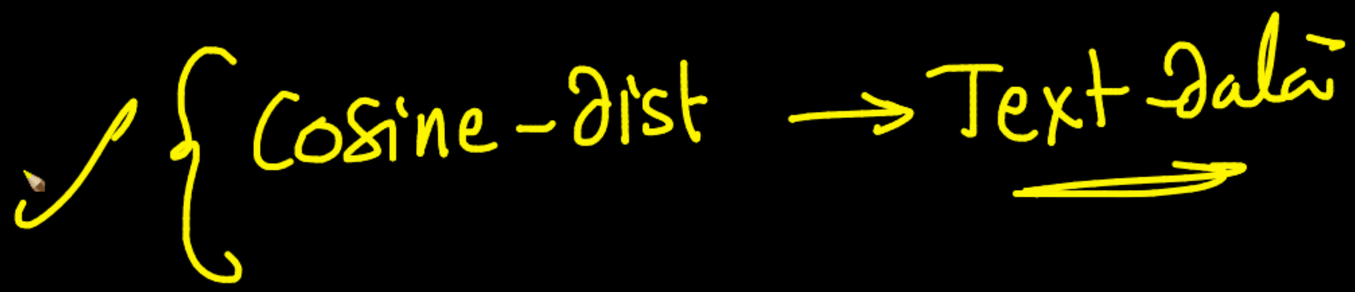
Minkowski dist (x_1, x_2, p)

$$\left[\sum_{j=1}^{d} |x_{1j} - x_{2j}|^{p} \right]^{1/p}$$

generalization of euc & manhattan dist

$p = 2 \rightarrow$ euc. dist

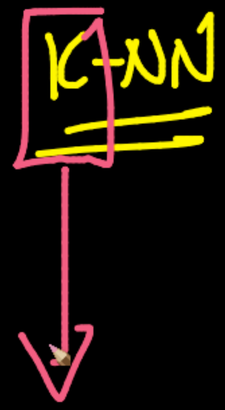
$p = 1 \rightarrow$ manhattan dist

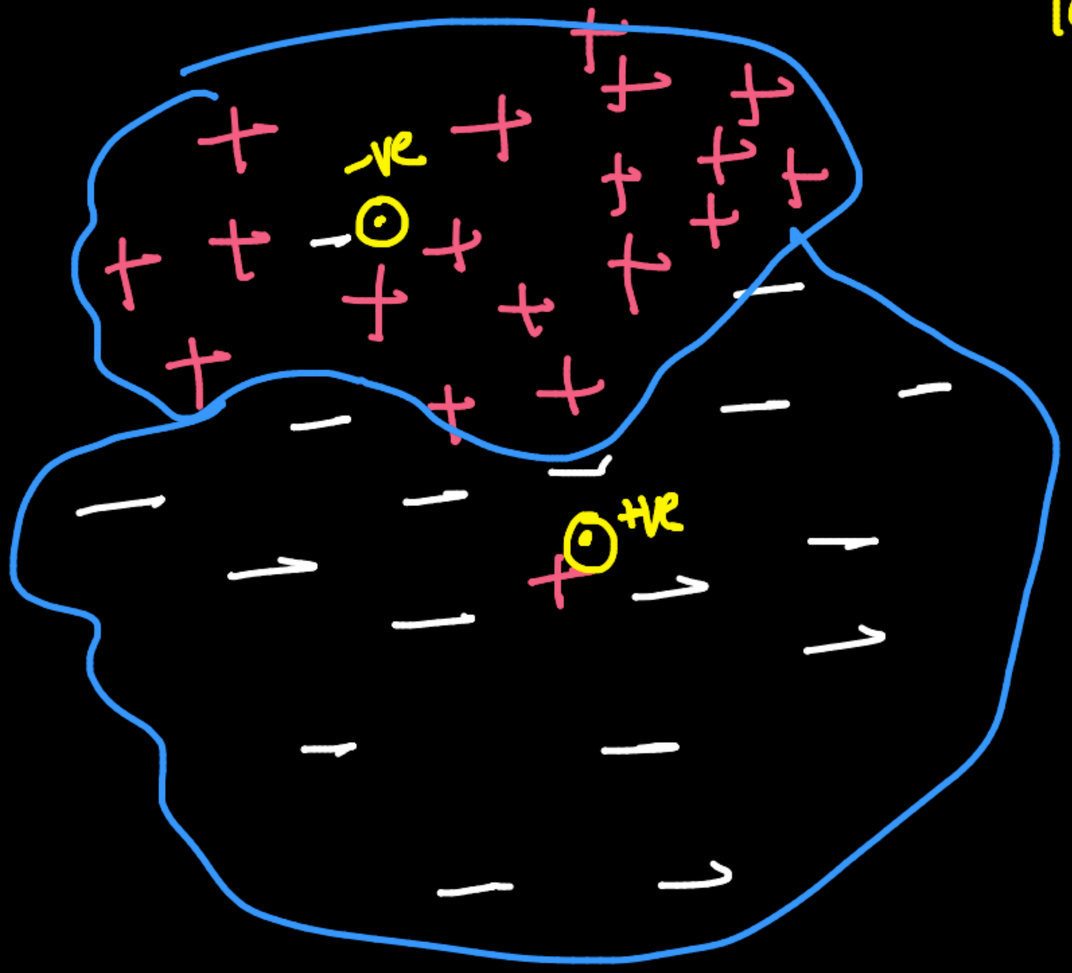


No training : D_{train}

Test time ✓ x_q

bias - variance tradeoff



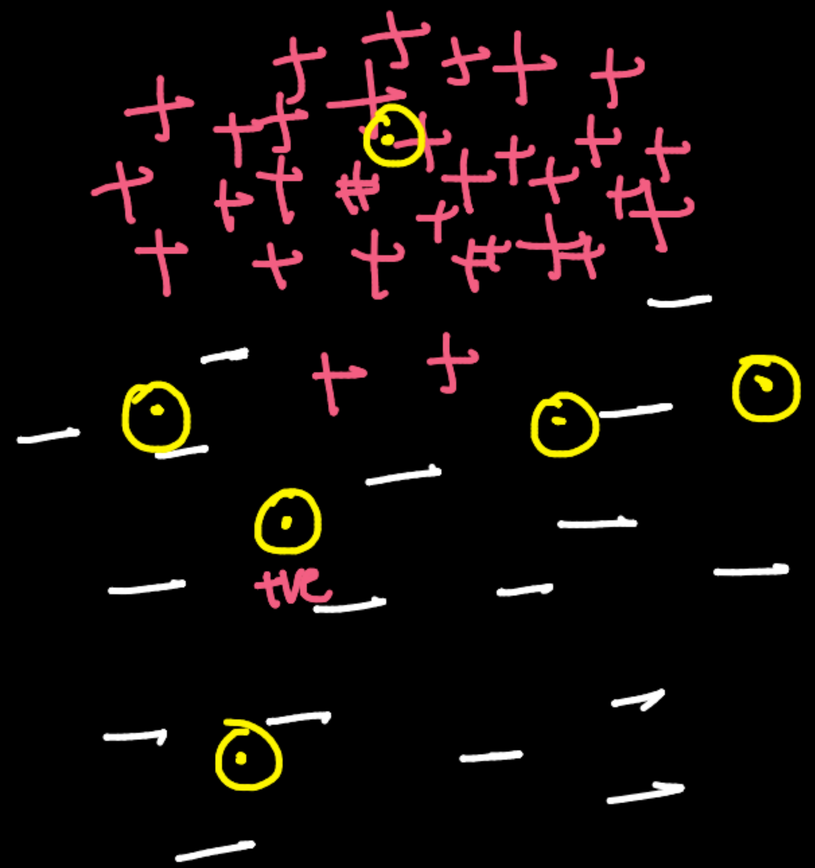


low

K: 1

filling to the noise/outliers

overfitting

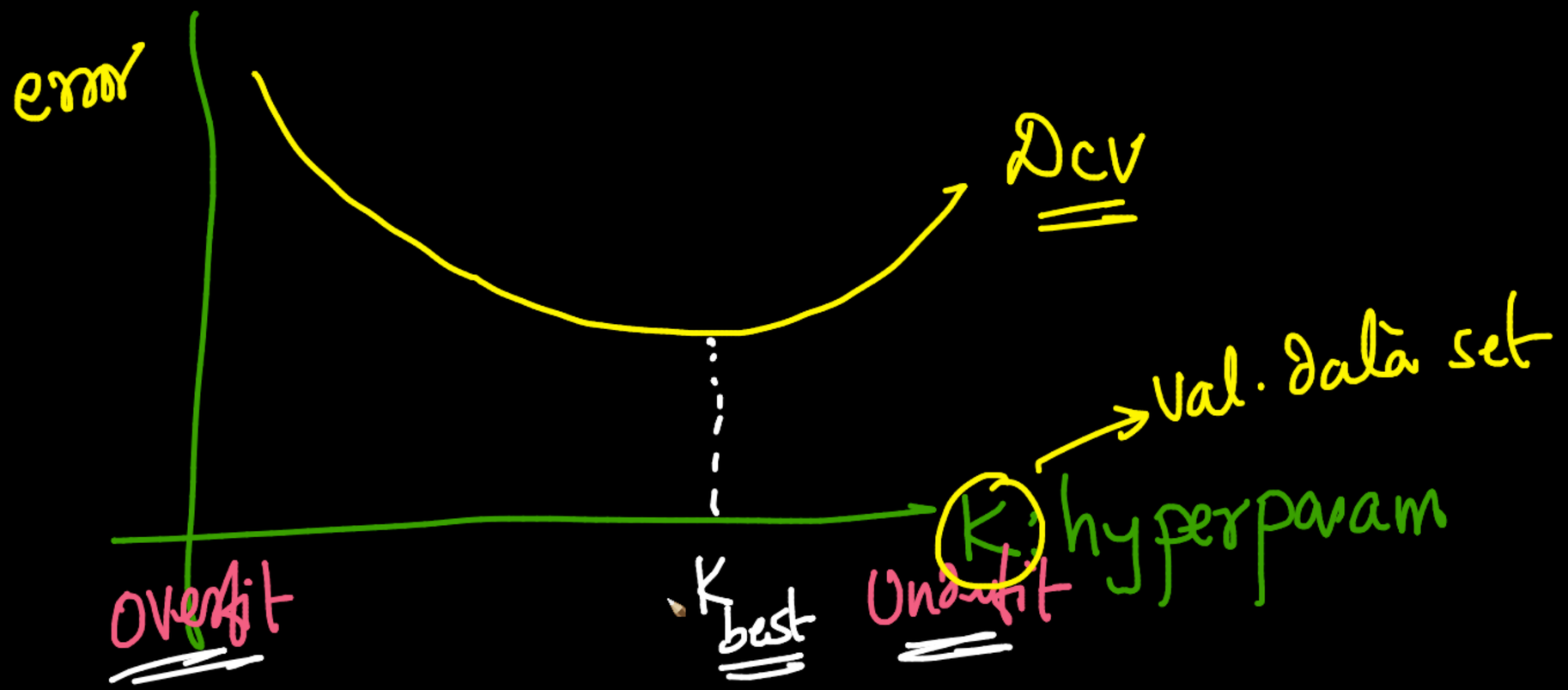


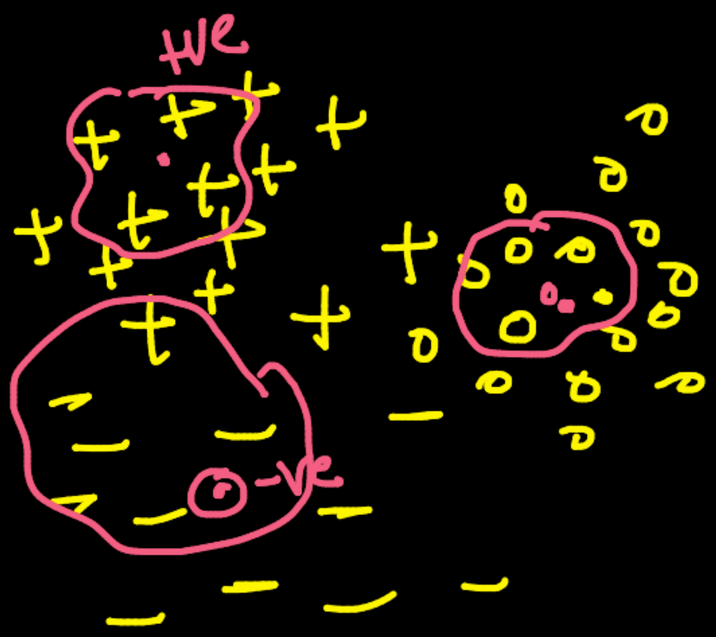
largest $K = n = 101$

{ +ve: 61 ✓
-ve: 40

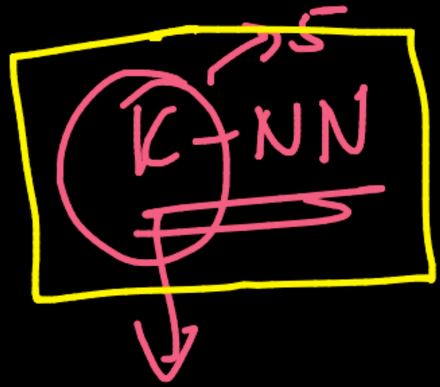
y_q = +ve

Underfitting


error
Dcv
Val. data set
K
hyperparam
overfit
K best
Underfit



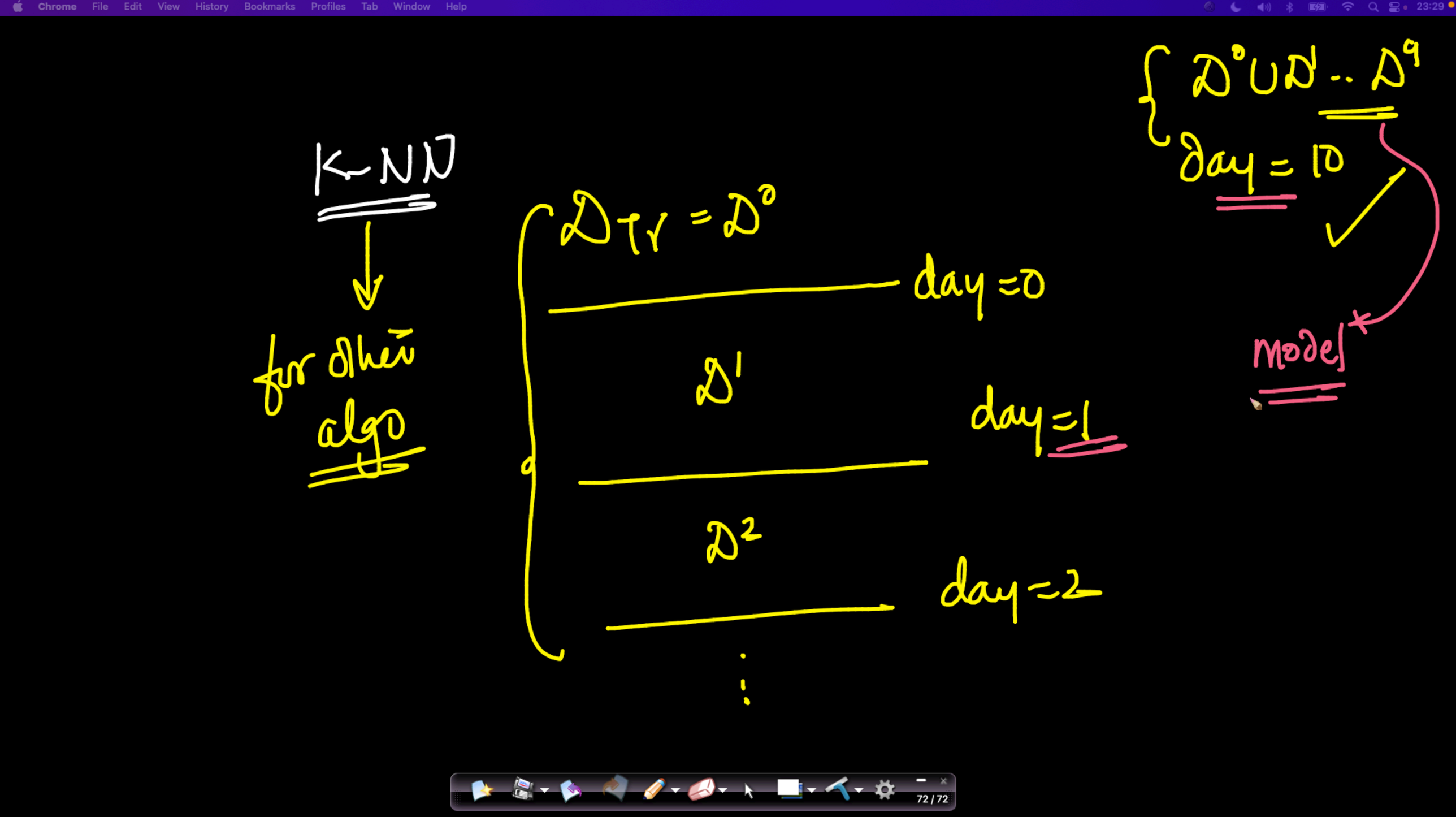
multi-class classification

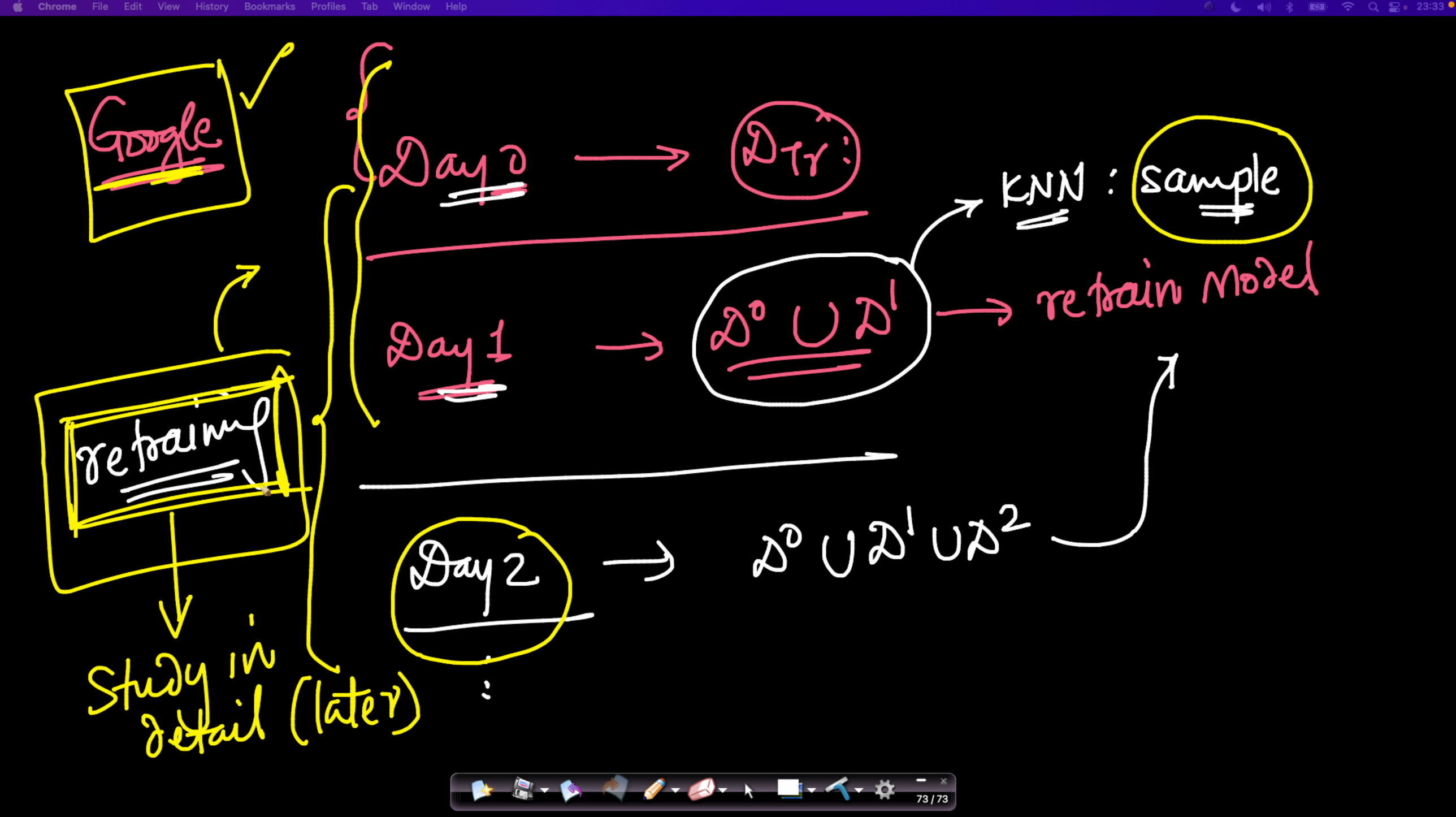


hyper-param

{ Majority vote
↳ class amongst K-NNs

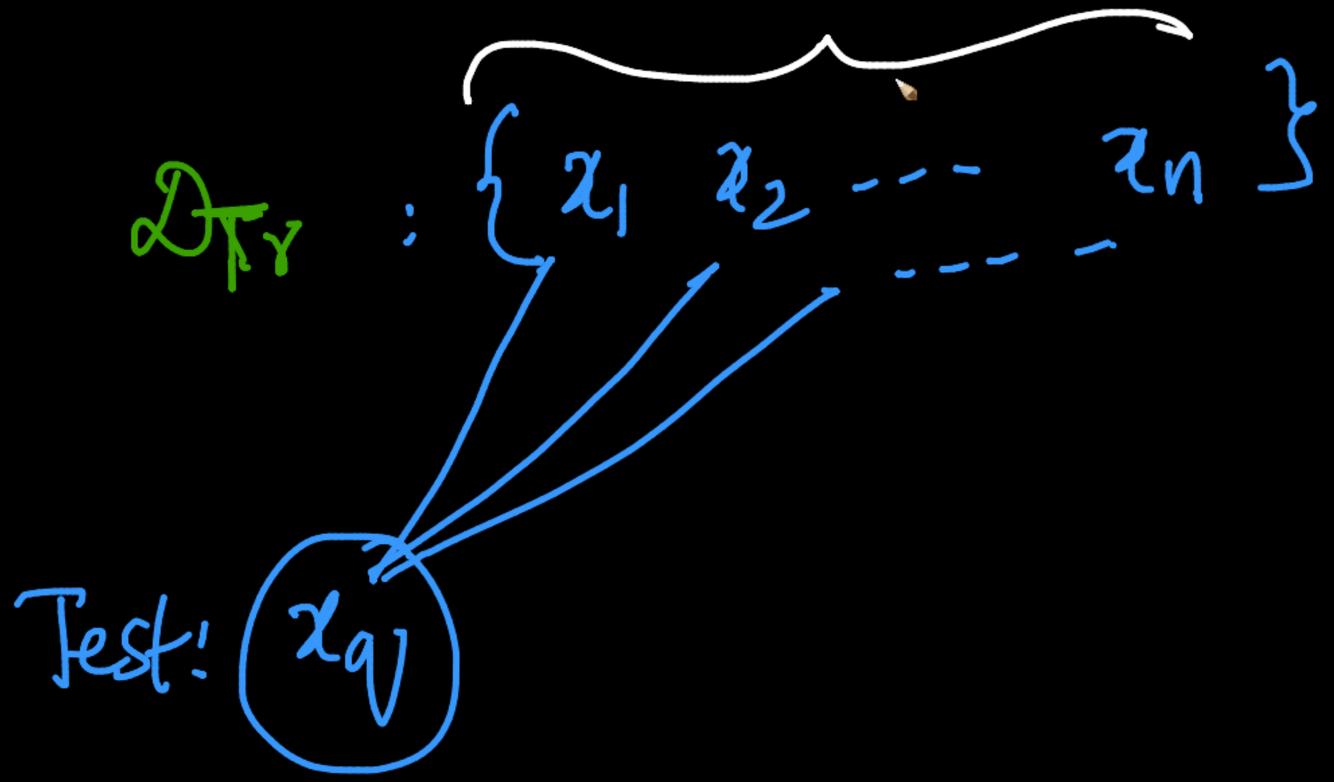
71 / 71


KNN
for other algo
$D_{TR} = D^0$
day = 0
D^1
day = 1
D^2
day = 2
$\{ D^0 \cup D^1 .. D^9$
day = 10
model


Google
Day 0
D_{tr}
KNN : sample
Day 1
$D^0 \cup D^1$
retrain model
retraining
Day 2
$D^0 \cup D^1 \cup D^2$
Study in detail (later)

Plan

- K-NN (contd) ... Misc (next-class)
- Odds ratio
- AUC; RoC; PRC
- ⋮
- !


D_{TY} : $\{ x_1 \quad x_2 \quad \cdots \quad x_n \}$
Test: x_q

logistic reg: imbalanced data

$$L = \sum_{i=1}^{d} \text{log loss}_i \cdot c_i + \lambda \cdot L_2 \text{reg}$$

model: w_j's

$$\min_{w_j} L$$

$$c_i = \begin{cases} 0.9 & \text{if } y_i = 1 \\ 0.1 & \text{if } y_i = 0 \end{cases}$$

$$\frac{\partial L}{\partial w_j}$$

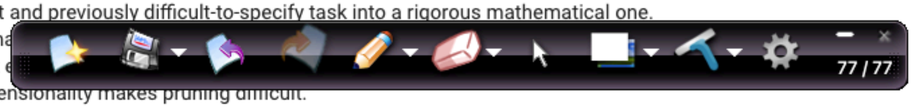
Announcing ScaNN: Efficient Vector Similarity Search

Tuesday, July 28, 2020

Posted by Philip Sun, Software Engineer, Google Research

Suppose one wants to search through a large dataset of literary works using queries that require an exact match of title, author, or other easily machine-indexable criteria. Such a task would be well suited for a relational database using a language such as SQL. However, if one wants to support more abstract queries, such as "Civil War poem," it is no longer possible to rely on naive similarity metrics such as the number of words in common between two phrases. For example, the query "science fiction" is more related to "future" than it is to "earth science" despite the former having zero, and the latter having one, word in common with the query.

Machine learning (ML) has greatly improved computers' abilities to understand language semantics and therefore answer these abstract queries. Modern ML models can transform inputs such as text and images into embeddings, high dimensional vectors trained such that more similar inputs cluster closer together. For a given query, we can therefore compute its embedding, and find the literary works whose embeddings are closest to the query's. In this manner, ML has transformed an abstract and previously difficult-to-specify task into a rigorous mathematical one. However, a computationa
find the nearest dataset e
search and its high dimensionality makes pruning difficult.



In our ICML 2020 paper, "Accelerating Large-Scale Inference with Anisotropic Vector Quantization," we address this problem by focusing on how to compress the dataset vectors to enable fast approximate distance computations, and propose a new compression technique that significantly boosts accuracy compared to prior works. This technique is utilized in our recently open-sourced vector similarity search library (ScaNN), and enables us to outperform other vector similarity search libraries by a factor of two, as measured on ann-benchmarks.com.

The Importance of Vector Similarity Search

Embedding-based search is a technique that is effective at answering queries that rely on semantic