Northeastern University
College *of* Engineering

IE7374 Project Presentation

Adult Income Classification Model

Reported by :

Yuxi Chen, Haotian Chen, Rundong Xu

Overall framework

Three Part

Reasons and ''**Data**''

''Model'' Competition

Design ''Algorithm''

1. Introduction

Introduction

What is the Topic?

*"Adult **Income** Classification Model Based on **Census** "*

Census Data

Internet Platform

Acquire individuals' information

↓

Formatting User Profile

↓

Recording Behavior

↓

"Data Asset"

Classification Model

Income Prediction

High Income person?

>= 50K	VS	<50K
• Target User • VIP • High Credit • Better Content		• Bad Content • Growth • Monetization • DAU • Duration
• Retention • Conversion • Monetization		

Discrimination means

"Money $$"

Introduction

Why we choose this Topic ?

"Adult Income Classification Model Based on Census "

Business Value

- Monetization: Advertisement
- Pre-Evaluation of customer
- Default Fraud Risk : Finance

Society Value

- Government Policy&Law
- Provide Benefit for low income people

- Warning!! Avoid Information leak

"We are naked in Big data world.!!"

2. Data

Dataset

Data: Adult Income Data
Link: https://archive.ics.uci.edu/ml/datasets/adult.
Shape: 48842 Obervations , 15 attributes

Categorical Feature :
Workclass, Education, Marital-status, Occupation, Relationship, Race, Sex, Native-country:

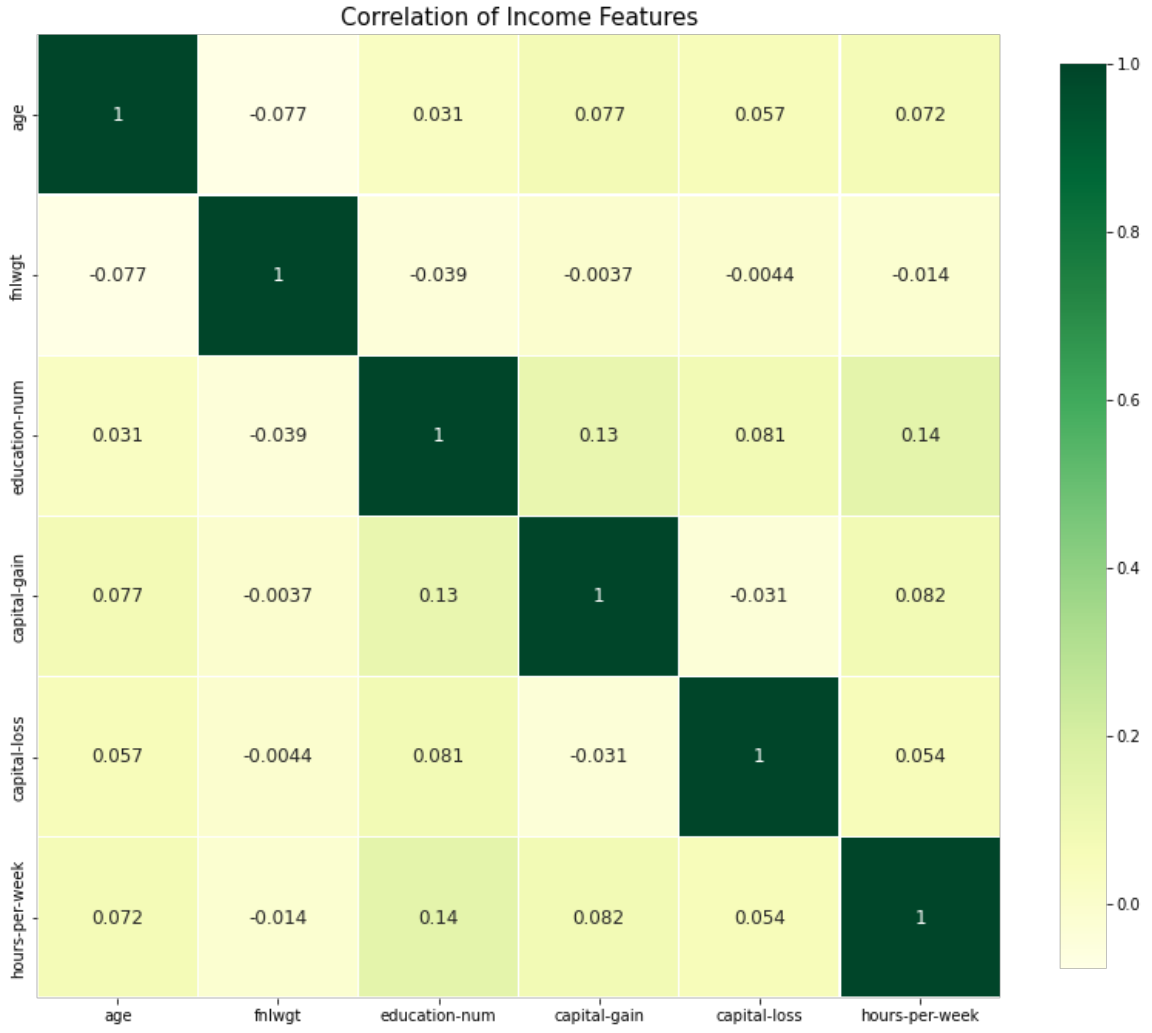
```
---  ------           ------
 0   age              48842
 1   workclass        48842
 2   fnlwgt           48842
 3   education        48842
 4   education-num    48842
 5   marital-status   48842
 6   occupation       48842
 7   relationship     48842
 8   race             48842
 9   sex              48842
 10  capital-gain     48842
 11  capital-loss     48842
 12  hours-per-week   48842
 13  native-country   48842
 14  income           48842
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

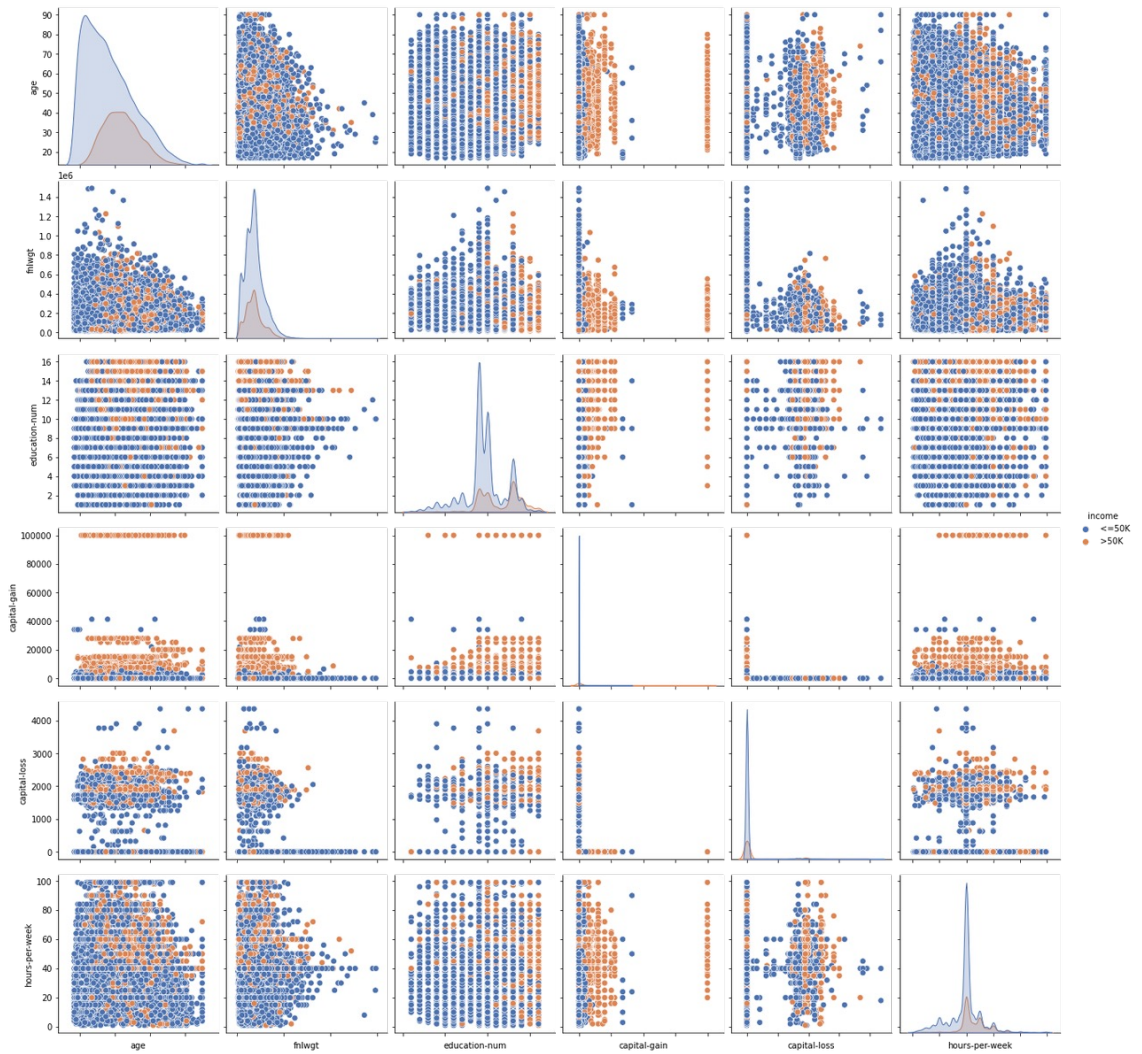
Numerical Feature:
Age, education-num, fnlwgt, Capital-gain, Capital-loss
Hours-per-week

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.422382
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.391444
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.000000

Observation:
Age: Range from 19 to 90 years, average is 37.
Education_num: from 1 and 16 ,the avg education level is 10 years.
hours.per.week:from 1 and 99, and the average is 40 hours.

EDA





1.Capital-gain and capital-loss:
there are too many '0' values here, especially for income<=50k, so both of them are the sparse features.
We need to deal with it. There is a lot of polarization, either clustered around 0 or very high-income groups,
which also reflects the '2:8' rule of social wealth distribution. **2.Age:** present right-skew tendency, especially for income<=50k

Feature Engineering

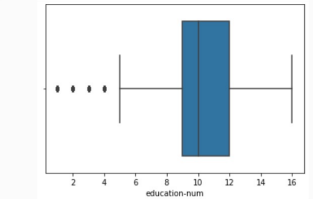
1. Feature discretization: Binning

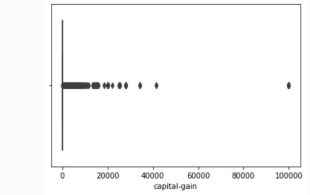
- Categorial feature combining: Map feature's content to 5 types
- Equal frequency binning: include as many values in each bin, because we want to keep same data in every binning.
- **Sparse Features：** We deal with the problem of too many '0' (sparse feature) by letting 0 is single bin, and others apply equal frequency binning to discretization the feature.

```
combining for baseline
lace([ 'Preschool', '1st-4th','5th-6th','7th-8th','9th', '10th','11th',  '12th'],
      'Drop_at_School_level', inplace = True)
lace(['Some-college','Assoc-acdm','Assoc-voc' ],'College',inplace=True)

|.replace(['Married-AF-spouse','Married-civ-spouse' ],'Married-spouse',inplace=True)

|.replace(['England','Scotland'],'UK',inplace=True)
|.replace(['Portugal','Germany','Italy','Yugoslavia','Hungary',
        'Greece', 'Poland','France' ,'Holand-Netherlands'],'Other-Europe',inplace=True)
|.replace(['Philippines','Vietnam','Thailand','Cambodia','Laos','Japan','India','Iran'],'Other-Asia',inplace=True)
|.replace(['United-States','Puerto-Rico','Outlying-US(Guam-USVI-etc)','Canada'],'North-America',inplace=True)
|.replace(['China','Taiwan','Hong'],'China',inplace=True)
|.replace(['Peru','Mexico','Dominican-Republic','Haiti',
       'Cuba','Guatemala','Columbia','Nicaragua', 'Honduras',
        'Jamaica' ,'Ecuador','El-Salvador','Trinadad&Tobago'] ,'Latin-America',inplace=True)
```

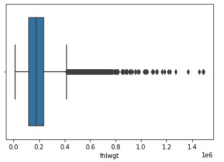


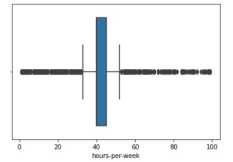


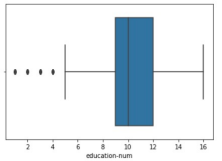
2. Processing Outlier

- Set the Low bound and high bound to get rid of outlier and form the **ad_df** dataset.

The shape shows that we get rid of 4842 outliers.





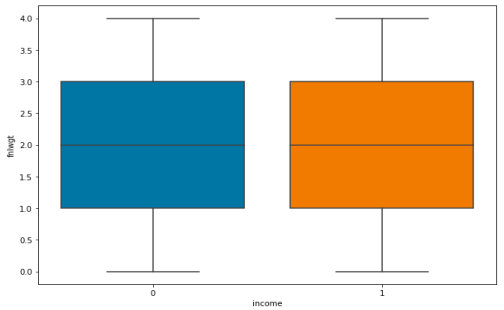


3. Hypothesis Testing

Null hypothesis: no difference
Alternative hypothesis: exist difference

```
ttest 0.8709825672046405
p-value 0.38482368112896215
we accept null hypothesis(no difference between mean of two group of income(>50k and<=50k.)
```

P value<0.05, we accept null hypothesis(No contribution to classification of income), drop flnwgt feature to form ad_df2.



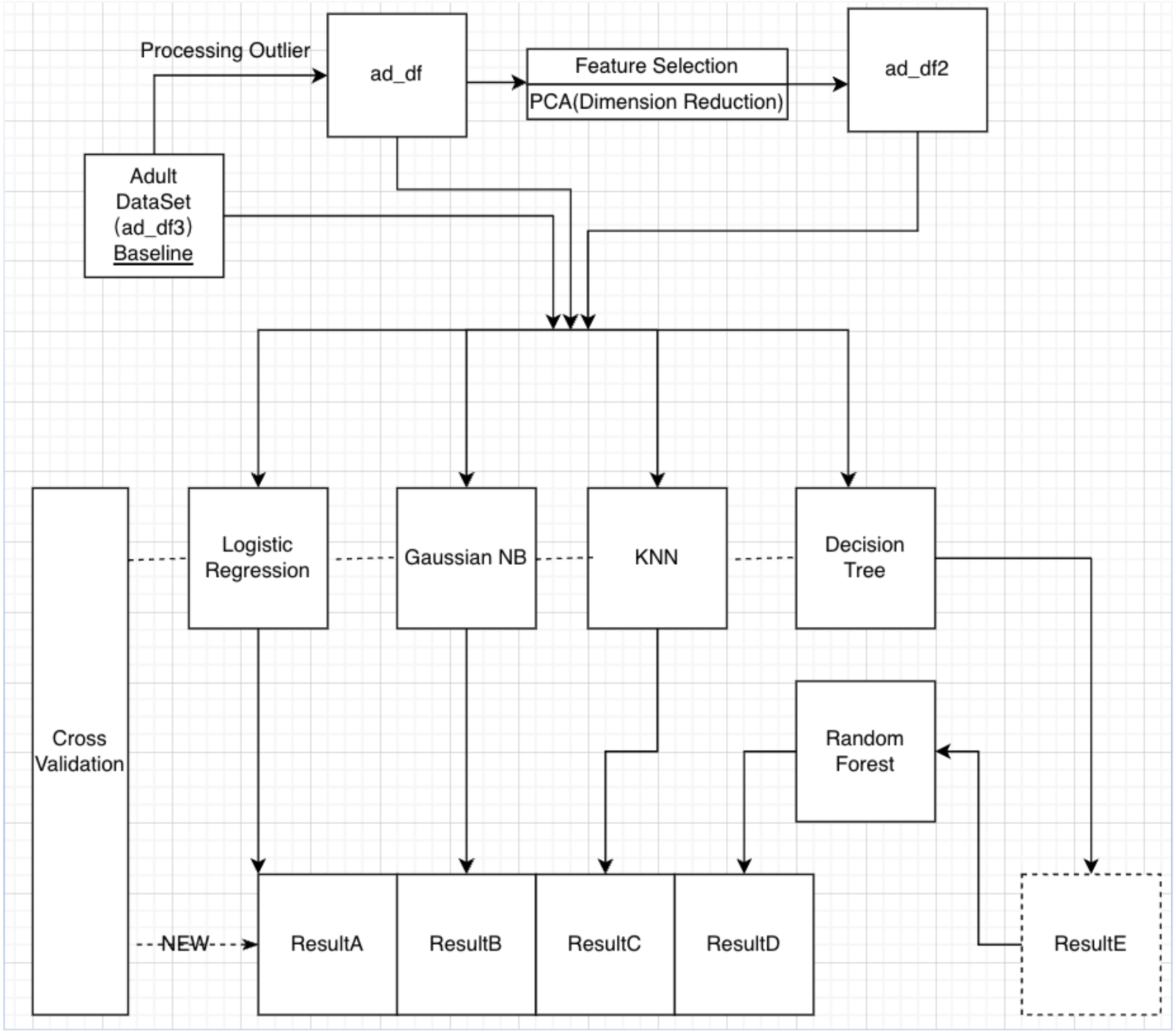
3. Method

Modeling Roadmap

Setting Three Scenario:

- **1. ad_df:** Get rid of Outlier
- **2. ad_df2:** Get rid of Outlier + feature selection by Hypothesis testing, domain knowledge
- **3. ad_df3(Baseline):** Do nothing improvement

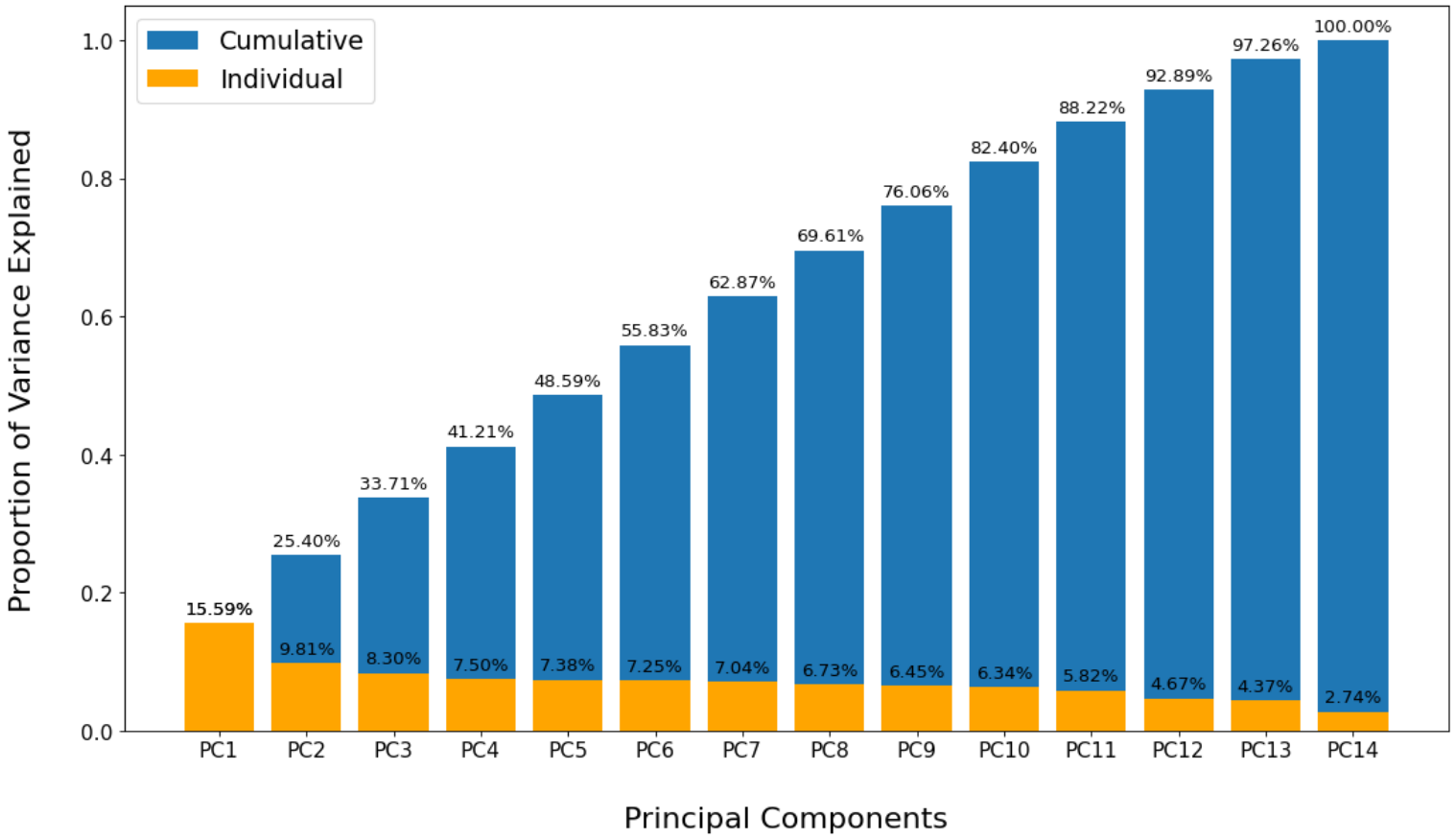
That's How we Modeling



How could you solve this problem?

PCA

Dimension Reduction



Input:

X: dataset

n_components: number of PCs

explained_variance_ratio: cumulative proportion of explained variance

Steps:

1. calculate covariance matrix
2. perform eigenvalue decomposition
3. sort eigenvalues and keep top k
4. return lower deminsion data

Plot:

orange bars: individual proportion of each PC

blue bars: cumulative proportions of top k PCs

We can see the first 12 PCs explained 92.89% variance(Over 90%)

Logistic Regression

How could you solve this problem?

1. Why do we use it ?

1.Simple to implement and widely used in industrial problems.
2.The amount of calculation is very small, the speed is very fast, and the storage resources are low.

```
[ ] lr = LogisitcRegression(X_train, y_train, X_test, y_test, learningRate = 0.01, tolerance = 0.001, maxIteration = 1000)
    lr2 = LogisitcRegression(X2_train, y2_train, X2_test, y2_test, learningRate = 0.01, tolerance = 0.001, maxIteration = 1000)
    lr3 = LogisitcRegression(X3_train, y3_train, X3_test, y3_test, learningRate = 0.01, tolerance = 0.001, maxIteration = 1000)
```

```
[ ] lr.fit()

    Solving using gradient descent
    100%|██████████| 1000/1000 [00:07<00:00, 134.26it/s]
    Accurancy is 0.6885606060606061.
    Precision is 0.4394485351715494.
    Recall is 0.8403235470341522.
    F1 Score is 0.5771011212838185
```

```
[ ] lr2.fit()

    Solving using gradient descent
    100%|██████████| 1000/1000 [00:07<00:00, 129.38it/s]
    Accurancy is 0.6962878787878788.
    Precision is 0.4457558609539208.
    Recall is 0.8259436788496105.
    F1 Score is 0.5790192166334138
```

```
[ ] lr3.fit()

    Solving using gradient descent
    100%|██████████| 1000/1000 [00:08<00:00, 112.59it/s]
    Accurancy is 0.5608407834573125.
    Precision is 0.20856137607505865.
    Recall is 0.30900666087460177.
    F1 Score is 0.2490372272143774
```

2. Design idea

1. The goal of Logistic Regression is to find the weight with sigmoid function.
2. Keep updating the w value and calculate the lowest cost function and predict label until it hit tolerance.

3. Problem encountered

cannot apply '(sig - self.y_train).dot(self.X_train)'

4. Improvement

change the type of input data into ndarray

How could you solve this problem?

Naive Bayes

1. Why do we use it ?

1.The theory is mature and the thinking is simple, which can be used for both classification and regression.
2.Can be used for nonlinear classification.
3.The training time complexity is O(n).
4.No assumptions about the data, high accuracy, not sensitive to outliers.

2. Design idea

1.fit distribution for each feature
2.calculate prior probability for each class

```
clf = GaussianNaiveBayes(X_train, y_train, X_test, y_test)
clf.fit()
clf.predict()

clf2 = GaussianNaiveBayes(X2_train, y2_train, X2_test, y2_test)
clf2.fit()
clf2.predict()

clf3 = GaussianNaiveBayes(X3_train, y3_train, X3_test, y3_test)
clf3.fit()
clf3.predict()
```

```
clf.score()
print('')
clf2.score()
print('')
clf3.score()
```

```
Accuracy is 0.7896212121212122.
Precision is 0.6355727404543258.
Recall is 0.3939484721390054.
F1 Score is 0.48640651007952646

Accuracy is 0.7891666666666667.
Precision is 0.6660682226211849.
Recall is 0.3334331935290593.
F1 Score is 0.44440007985625873

Accuracy is 0.8051593530335085.
Precision is 0.6792565947242206.
Recall is 0.32812047494931945.
F1 Score is 0.4424917008396798
```

3. Problems encountered

1.There are many features and it is confusing to write code to fit distribution for each feature and each class one by one.
2.self.fitDistribution can't be used after being saved into a list.

4. Improvement

1.We write a 'for' loop to fit distribution for each feature of two classes.
2.Store the results in an tarray.

How could you solve this problem?

KNN

1. Why do we use it ?

1.Simple, easy to understand, high precision; Insensitive to outliers; No data input settings

2.Downside: High computational complexity and space complexity

2. Design idea

1.set the k values and calculate the euclidean distances between the sample point and all the train points.

2.pick top k distances and count the occurrence times of various labels in these K distances

3.classify the SAMPLE as the label with the most occurrence times.

3. Problems encountered

1.The model used up all the RAM of Colab for the first time, resulting in a forced stop

2.The second model run took too long

```
[ ]  knn1 = KNNClassifier(X_train, y_train, X_test, y_test)
     knn2 = KNNClassifier(X2_train, y2_train, X2_test, y2_test)
     knn3 = KNNClassifier(X3_train, y3_train, X3_test, y3_test)
```

```
[ ]  knn1.KNN_classifier(4)
     knn2.KNN_classifier(4)
     knn3.KNN_classifier(4)

     100%|██████████| 13200/13200 [00:21<00:00, 604.08it/s]
     100%|██████████| 13200/13200 [00:16<00:00, 788.98it/s]
     100%|██████████| 14653/14653 [00:26<00:00, 552.69it/s]
```

```
     knn1.score()
     print('')
     knn2.score()
     print('')
     knn3.score()

     Accuracy is 0.8197727272727273.
     Precision is 0.7245901639344262.
     Recall is 0.46345116836429.
     F1 Score is 0.5653206650831354

     Accuracy is 0.8234848484848485.
     Precision is 0.7163090128755365.
     Recall is 0.5.
     F1 Score is 0.5889202540578687

     Accuracy is 0.8200368525216679.
     Precision is 0.6917293233082706.
     Recall is 0.4262959745149146.
     F1 Score is 0.527504031535567
```

4. Improvement

1.The code itself has many 'For loops' for the second time causing the model to run too long

2.Through the optimization and rewriting of code, the for loops are reduced without making any errors in the algorithm, thus shortening the time to 3 minutes

Decision Tree

Why do we use it?

- simple to calculate, easy to understand and has strong interpretability
- more suitable for processing samples with missing attributes
- ability to handle irrelevant features
- ability to produce feasible and well-executed results on large data sources in relatively short time

Input:

Y : actual class of training samples
X : training sample points
min_samples_split : the minimum number of samples required to split
max_depth : the maximum depth of tree
depth : default current depth of node
X_features_fraction : the proportion of features to find the best split
node_type : type of node
rule : the rule to measure the quality of a split

ad_df3	Decision Tree
Accuracy	0.856480
Precision	0.808501
Recall	0.512308
F1 Score	0.627194

Steps:

1. split data into two parts :
   1.1 calculate GINI impurity and GINI gain
   1.2 choose split point with the highest GINI gain
   - stop split if negative

2. grow a tree
   - stop when reaching the maximum depth
     or the minimum samples for split

Random Forest

Why do we use it?

- Usually, random forest is often the winner of many classification problems.
- Random forest can greatly reduce the risk of overfitting caused by decision trees.

Input:

n_trees : the number of trees in the forest

X_obs_fraction : the proportion of samples we select from original dataset with replacement

Steps:

1. choose the number of the decision trees we use - k trees
2. randomly sample the rows from the original dataset with replacement for each tree
3. select a random subset of features from all features for each tree
4. use class *DecisionTree* to grow trees

Compare:

ad_df3	Accuracy
Decision Tree	0.856480
Random Forest	0.854289

ad_df3	Random Forest
Accuracy	0.854289
Precision	0.787609
Recall	0.522908
F1 Score	0.628382

4. Result

Model Comparison

Comparing algorithm within Group

Algorithm VS Algorithm for Metrics in same dataset

Comparing between groups after Improvement

ad_df3 VS ad_df2 VS ad_df for same Algorithm's Metrics

Generalization Ability Perspective

VS

Bias-Variance-Tradeoff

Optimization Perspective

Best Accuracy model VS Model applied Cross Validation

Consumption Perspective

Time Consumption & RAM Consumption

Result

Algorithm VS Algorithm for Metrics in same dataset

Baseline: ad_df3

ad_df3	Accuracy	Precision Rate	Recall Rate	F-1 Score
Logistic Regression	0.56084	0.208561376	0.30901	0.30901
Gaussian NB	0.80515935	0.67925659	0.3281205	0.4424917
KNN	0.82004	0.691729	0.426296	0.527504
Decision Tree	0.85648	0.8085009	0.5123081	0.5123081
Random Forest(Avg-Score)	0.85377	0.7734177	0.5390964	0.6343827

Obeservision:

- Random Forest seems to be the best model among the above 5 algorithms . It has the best F-1 Score. Although its accuracy is 0.85377, the decision tree is 0.85648 .
- If we pursue high accuracy, we could choose the decision tree, but we still need to see bias-variance-trade off in order to evaluate it's generalization ability, given that usually the decision tree is easy to be overfitting.

Result

Algorithm VS Algorithm for Metrics in same dataset

After Processing outlier
ad_df

ad_df	Accuracy	Precision Rate	Recall Rate	F-1 Score
Logistic Regression	0.68856	0.4394485	0.84032	0.84032
Gaussian NB	0.7896212	0.78962	0.3939485	0.4864065
KNN	0.82153	0.636818	0.827569	0.624196
Decision Tree	0.8356818	0.755798687	0.51737567	0.6142628
Random Forest(Avg-Score)	0.83675	0.7620979	0.52366687	0.61694412

- After dealing with Outlier by EDA boxplot, we could see almost all metrics decreased a bit, but it has the remarkable improvement in logistic Regression . We can see huge improvement here .
- The reason is that LR are more **Outlier sensitive** than others, because LR is a parametric model (Logistic regression is assumed to obey Bernoulli distribution).

Baseline VS Processing Outlier	Accuracy	Precision Rate	Recall Rate	F-1 Score
Logistic Regression(ad_df3)	0.56084	0.20856	0.30901	0.30901
Logistic Regression(ad_df)	0.68856	0.439449	0.84032	0.84032

Result

Algorithm VS Algorithm for Metrics in same dataset

After Feature Selection
ad_df2

ad_df2	Accuracy	Precision Rate	Recall Rate	F-1 Score
Logistic Regression	0.696288	0.445756	0.82594	0.579019
Gaussian NB	0.789167	0.666069	0.666068	0.44444
KNN	0.827567	0.647734	0.602309	0.624196
Decision Tree	0.833636	0.766325	0.766324	0.599416
Random Forest(Avg-Score)	0.83178	0.749153	0.508628	0.603604

Obeservision:

- After applying Feature Selection, For LR, the accuracy and precision rate continue to improve a lot, but F-1 Score has decreased a lot.

Baseline VS Processing OutlierVS Feature Selection	Accuracy	Precision Rate	Recall Rate	F-1 Score
Logistic Regression(ad_df3)	0.56084	0.20856	0.30901	0.30901
Logistic Regression(ad_df)	0.68856	0.439449	0.84032	0.84032
Logistic Regression(ad_df)	0.696288	0.445756	0.82594	0.579019

Result

ad_df3 VS ad_df2 VS ad_df for same Algorithm's Metrics

		Logistic Regression	Gaussian Naive Bayes	Decision Tree	Random Forest	KNN
ad_df	Accuracy	0.688561	0.789621	0.835682	0.836242	0.819773
	Precision	0.439449	0.635573	0.755799	0.739229	0.724590
	Recall	0.840324	0.393948	0.517376	0.553775	0.463451
	F1 Score	0.577101	0.486407	0.614263	0.629451	0.565321
ad_df2	Accuracy	0.696288	0.789167	0.833636	0.835394	0.823485
	Precision	0.445756	0.666068	0.766325	0.740039	0.716309
	Recall	0.825944	0.333433	0.492211	0.544787	0.500000
	F1 Score	0.579019	0.444400	0.599416	0.624704	0.588920
ad_df3	Accuracy	0.560841	0.805159	0.856480	0.853614	0.820037
	Precision	0.208561	0.679257	0.808501	0.787708	0.691729
	Recall	0.309007	0.328120	0.512308	0.518853	0.426296
	F1 Score	0.249037	0.442492	0.627194	0.625431	0.527504

Conclusion(Algorithm Perspective)

	Best Acc	Dataset
Logistic Regression	0.696288	ad_df2
Gaussian Naive Bayes	0.805159	ad_df3
Decision Tree	0.856480	ad_df3
Random Forest	0.853614	ad_df3
KNN	0.823485	ad_df2

3 Conclusion(Algorithm Perspective):

i. Accuracy between algorithm

- The Decision tree, Ramdom forest and KNN usually occupy the Top 3.
- The lowest accuracy is Logistic Regression, because LR have these cons:
  - when the feature space is large, the performance of logistic regression is not very good;
  - It is easy to under-fit, and the general accuracy is not very high
  - Does not handle a large number of multi-class features or variables well;
- **We could choose Decision tree, which has relatively good metrics, in general. But we still need to look at it's generalization ability, because it easily causes overfitting.**

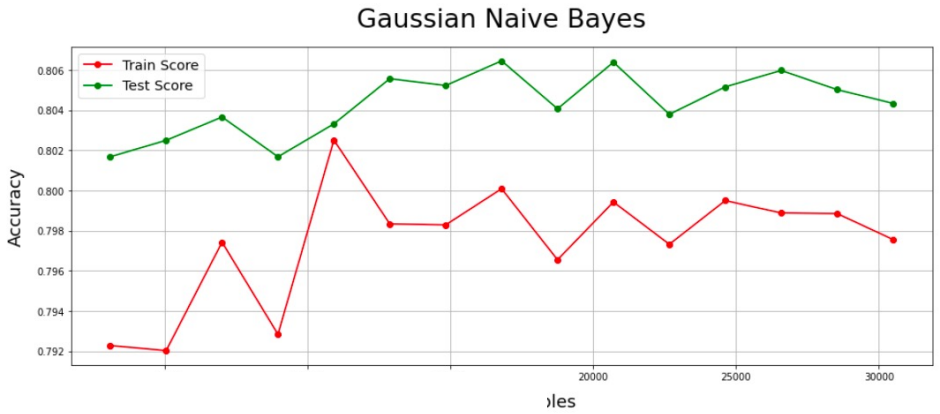
Conclusion(Algorithm Perspective)

Other Metrics Comparation

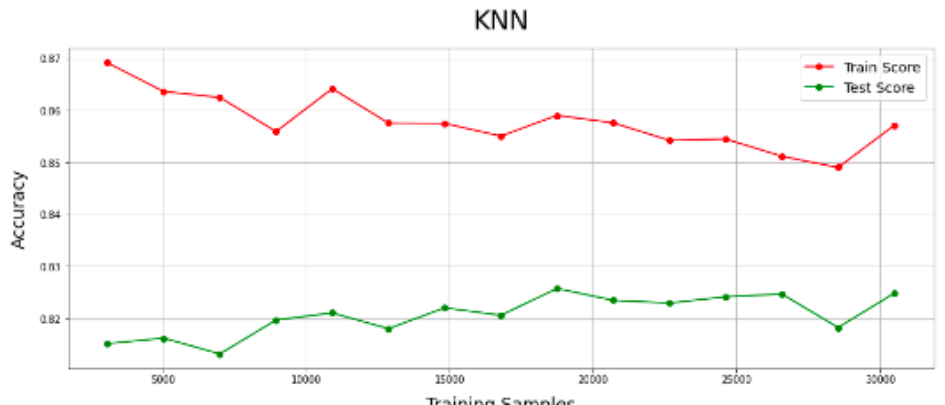
- **Precision rate**
  - When the cost of False Positive (FP) is very high (the consequences are serious), that is, when it is expected to avoid generating FP as much as possible, it should focus on improving the Precision index. We should choose **decision tree 0.8085009 precision rate**
- **Recall：**
  - When the cost of False Negative (FN) is very high (the consequences are serious), and you want to avoid generating FN as much as possible, you should focus on improving the Recall indicator. We should choose logistic regression **0.84 recall rate, or decision tree 0.766324 recall rate.**
- **F1-Score:**
  - If we want to keep Precision rate and Recall tradeoff, usually, F1- Score are important in common knowledge. So we should choose **logistic regression 0.84032 F1-Score, or choose random forest 0.6343827 F1-Score.**

5.Bias-Variance-tradeoff

Learning curve



a. Fitting situation
   i. Low bias, high variance---- overfitting
   ii. High bias, low variance ----underfitting
   iii. High bias, high variance ----- underfitting
   iv. Low bias, low variance ---- good fitting
b. For accuracy and error:
   i. high bias：train accuracy is low
   ii. high variance：trian accuracy is high, test accuracy is low.



Bias-variance-trade-off thinking

3.1.1 Takeaway：

- Based on plot, we can see the best fitting model is Random Forest and Decision tree.
- We can see Naive bayes test rate is high. It may be because of sampling bias, such as the samples in the test set are easy to predict . It is coincidence.

3.1.2 Generalization Ability Thinking：

Usually the model is trained by minimizing the training error, but the real concern is the test error. Therefore, the generalization ability of the model is evaluated by test error.

- The training error is the average loss of the model over the training set, and its size is meaningful, but not intrinsically important.
- The test error is the average loss of the model on the test set, which reflects the predictive ability of the model on the unknown test data set. The high 'mean value' means it changes the least with test size

Therefore, based on the curve, we could conclude that Random Forest has the best generalization ability than others.

Learning curve

	Mean Acc - Train	Mean Acc - Test
Logistic Regression	0.699932	0.701692
Gaussian Naive Bayes	0.798592	0.804686
Decision Tree	0.848898	0.849601
Random Forest	0.848366	0.847212
KNN	0.859423	0.821965

3.2 Conclusion2(bias-variance-tradeoff Perspective):

i. Naive Bayes: high bias, low variance.

ii. KNN high variance, and it seems overfitting, but it learning curve not sharp,

iii. Overfitting is usually caused by three reasons: 1. The model is too complex and there are too many parameters, 2. The generalization ability of the model is not enough, 3. The data is too small

iv. **For example: KNN>NB in bias-variance-tradeoff**

1. In small dataset, high bias/low variance classifier (e.g., Naive Bayes NB)> Low bias/high variance classifier (KNN), cause it would cause overfitting.
2. With Training data improved, ability of prediction would be improved, then bias would be lower, so Low bias/high variance classifier (KNN)(It has low asymptotic error)>NB

Cross Validation

	Best Acc	Dataset
Logistic Regression	0.696288	ad_df2
Gaussian Naive Bayes	0.805159	ad_df3
Decision Tree	0.856480	ad_df3
Random Forest	0.853614	ad_df3
KNN	0.823485	ad_df2

After Cross Validation

	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.709273	0.598386	0.512280	0.417035
Gaussian Naive Bayes	0.798526	0.662899	0.321926	0.433363
Decision Tree	0.851188	0.795933	0.509542	0.620708
Random Forest	0.851286	0.783692	0.523697	0.627317
KNN	0.830159	0.729267	0.499862	0.592932

Comparsion

	K-fold Acc	Best Acc	Dataset
Logistic Regression	0.709273	0.696288	ad_df2
Gaussian Naive Bayes	0.798526	0.805159	ad_df3
Decision Tree	0.851188	0.856480	ad_df3
Random Forest	0.851286	0.853614	ad_df3
KNN	0.830159	0.823485	ad_df2

4.3 Takeaway：

- We can see that accuracy is increased after CV in Logistic Regression.
- Others would be decreased a little, maybe because we use 5 fold, so the sample size is decreased.

Consumption

Time-Consumption

	ad_df1 run time	ad_df2 runtime	ad_df3 runtime
Logistic Regression	7 sec	7 sec	8 sec
Gaussian Naive Bayes	46 sec	38 sec	50 sec
Decision Tree	7 sec	5 sec	14 sec
Random Forest	53sec	47 sec	56 sec
KNN	1h56min -> 23 sec	17 sec	27 sec
K-Fold Cross Validation	5 min	4 min	7 min

RAM-Consumption

5.2 RAM Space

	All ad_df RAM
Logistic Regression	2 gb
Gaussian Naive Bayes	3 gb
Decision Tree	2 gb
Random Forest	2.5gb
KNN	2gb

Takeaway

KNN：

1. It has large amount of calculation,and face sample imbalance problem (i.e. some classes have a large number of samples while others have a small number). It requires a large amount of memory(RAM).

5.Conclusion & Summary

Conclusion & Summary

OKR

Based on above 4 perspective

1. Metrics
2. Bias-Variance-trade-off
3. generalization ability
4. K-fold cross validation
5. Consumtion

In general, We would Choose Random Forest to be the classification model first because it have better accuracy and generalization ability, and then we need to do HyperParameter optimization such as Grid Search in order to compare all model !

Q&A

Questions from Prof: Ramin

Q1. How would you perform PCA for categorical data. Did you guys use autoencoder?

A1: Sure, we applied labelencoder, standardEncoder before the PCA.

Q2. For Gaussian NBC, you guys talk about the assumption, why would you use that.

A2: Yes, we could see some weird tendency in plot, so like I say, we want to emphasize that is the red flag, usually it could be interpreted, may be because the sample bias, may be is the **Gaussian assumption**(Do not to use Binning to transform the data to be categorical), I hope everyone could pay more attention about it.

Questions from Prof: Ramin

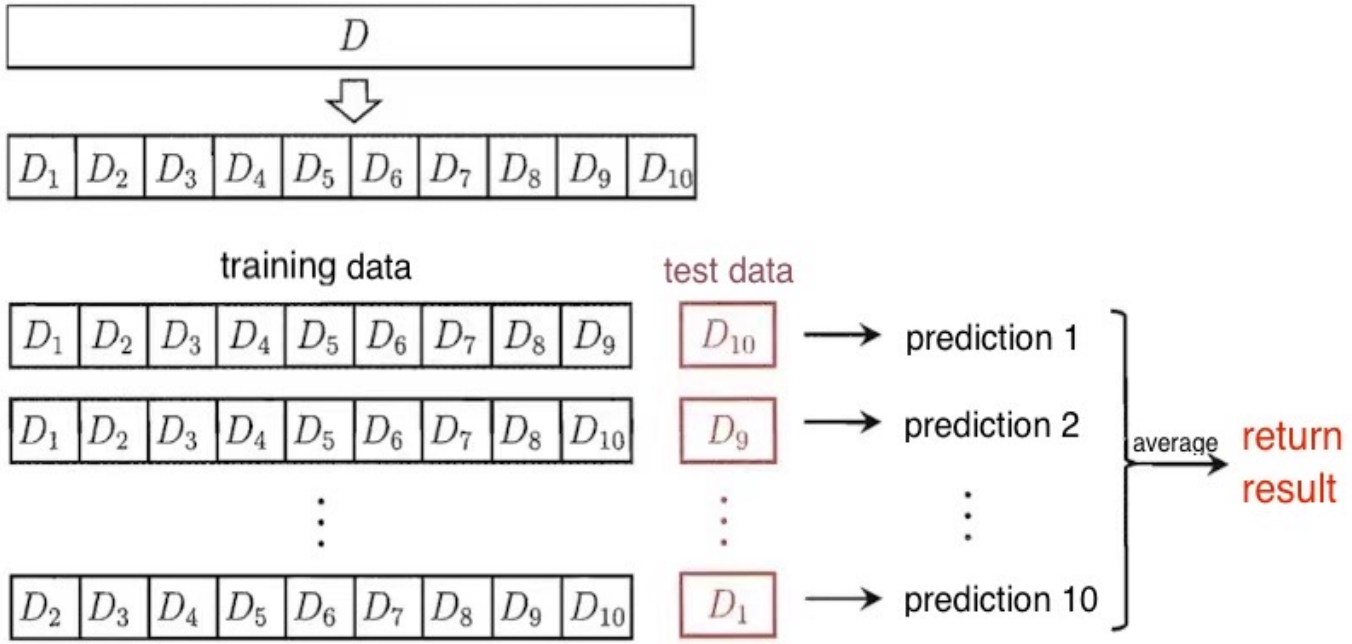
Q3. For Gaussian NBC, you guys talk about the assumption, why would you use that.

A3:Because of limited time, we could see difference after K-Fold Cross Validation, the random forest have more good generalization ability, we should choose it

Q&A

Q4. why you guys use only train data and test data, no validation data.

First of all, k-fold cross validation serves two usages: model evaluation and model selection.
The former usage is used in our project. Compared with the traditional method of model evaluation (dividing dataset into fixed training data and test data), the advantage of cross validation is that it avoids overfitting caused by unreasonable division of the dataset, which may not be caused by the model.
Therefore, we use K-fold cross validation to divide the dataset multiple times, average the results of multiple evaluations (such as accuracy), and use this to evaluate the performance of the model.
This can eliminate the imbalanced data division during a single division resulting adverse effects. However, since we did not tune the parameters of the differen models, we only used cross-validation for model evaluation. At this point, cross-validation cannot solvethe overfitting problem and can only be used to evaluate the performance of different kinds of model use din our project.



Citation: Zhou Zhihua, "Machine Learning".

Questions

&

Answers

Thanks for Everyone to Listening !!

Thanks for Professor: Ramin