

Do not overfit!

Group 21








JIANG Yuxin	16096336D
WANG Jiashuo	16096527D
Xu Suyan	16097128D
Yu Jing	16098537D
WANG Bokang	16097234D

Final Score and Rank

Final Score:

Submission and Description	Private Score	Public Score	Use for Final Score
sub_RFE.csv 2 days ago by JYU999 add submission details	0.856	0.868	<input type="checkbox"/>

Final Rank: About 57 (Private Score)

56	▼ 6	Rob Freeman		0.857	118	7mo
57	▲ 3	Anshul Rai		0.856	20	7mo
58	▲ 15	KimStark		0.856	10	7mo
59	▲ 9	jinjingxu		0.856	47	7mo
60	▲ 10	dololo		0.856	8	7mo
61	▲ 28	Alexander Zinovev		0.856	32	7mo
62	▲ 55	Chun-Chao Lo		0.855	4	8mo

Outline

- Problem Definition
- Data Processing and Feature Engineering
- Model Design and Experiment
 - KNN
 - Linear SVM
 - Decision Tree and Random Forest
 - AdaBoost Classifier
 - Gradient Boosting Classifier
 - Generalized Linear Model
 - Linear Regression - Lasso, Ridge
 - Logistic Regression
- Performance Evaluation
 - Best Model
 - Feature Distribution
- Future Development

Problem Definition & Released Solution Analysis

- It was a really challenging competition which requires us to model a 20,000x300 matrix of continuous variables using only 250 training samples without overfitting.
- Since many participants release their solutions, we analyzed their solutions. We found that instead of finding the hyperplane that classifies the training data, the top two prefer to find the hyperplane that classifies the public test dataset. In other words, they want to find the feature of the public test dataset by trying again and again.
- But we think it violates the original intention of the competition. In real life, we may don't have that much test dataset or the test dataset is not unchangeable. Therefore, to lessen the chance of, or amount of, overfitting, we decide to use the traditional techniques such as model comparison, cross-validation, regularization, pruning, or dropout to solve this problem.

Data Processing and Feature Engineering

- **Output: 0/1**
 - Conclusion: This is a classification problem
- **Variables 0~299:**
 - Experiment 1: For each variable, we calculate max, 75%, 50%, 25%, min, standard deviation and mean.

	0	1	2	3	4	5
count	19750	19750	19750	19750	19750	19750
mean	-0.01404334177	0.0009718987342	0.005144860759	-0.003524658228	0.003393974684	0.002737924051
std	1.003778678	0.9939551617	1.00080895	1.008545076	1.002826096	1.002916891
min	-4.07	-3.664	-4.258	-4.14	-4.411	-3.586
25%	-0.68875	-0.667	-0.668	-0.686	-0.671	-0.679
50%	-0.006	0.001	0.017	-0.006	0.007	0.005
75%	0.664	0.676	0.681	0.682	0.676	0.68475
max	3.767	3.864	3.866	3.871	3.955	3.819

Data Processing and Feature Engineering

- **Variables 0~299:**

- Experiment 2: Find the contribution of each feature using random forest. The score is the negative value of accuracy with one feature. The larger the value is, more important the feature is.
 - Fact: There are only slight differences between features
 - Conclusion: We can select features according to their contribution.

191 -0.52
97 -0.524
164 -0.528
223 -0.532
64 -0.54
246 -0.54
83 -0.544
129 -0.544
265 -0.548
241 -0.548
136 -0.548
125 -0.548
70 -0.552
69 -0.552
3 -0.552
278 -0.552
258 -0.552
206 -0.552
167 -0.552
160 -0.552
115 -0.552
34 -0.556
297 -0.556
250 -0.556
203 -0.556
17 -0.556
156 -0.556

Data Processing and Feature Engineering

- **Variables 0~299:**

- Facts:

- Experiment results shows high similarity of these variables.
 - The number of variable is large.

- Conclusion:

- There is no critical feature of a variable itself that this variable should be deleted.
 - The number of variable is too large → we have a guess that using a few of them to do prediction works better than using all of them.

Data Processing and Feature Engineering

- **Recursive Feature Elimination (RFE):**

```
BestFeature = {};
```

```
FeatureSet = {all variables} ;
```

```
For i from 1-K:
```

```
    Find the best featur BF in FeatureSet;
```

```
    Add BF to BestFeature;
```

```
    Delete BF from FeatureSet;
```

```
Return BestFearture;
```


Data Processing and Feature Engineering

- **SelectKBest:**

BestFeature = {};

FeatureSet = {all variables} ;

Find K best features from FeatureSet;

Add the K feature to BestFeature;

Return BestFearture;

Model Design and Experiment: KNN

Description:

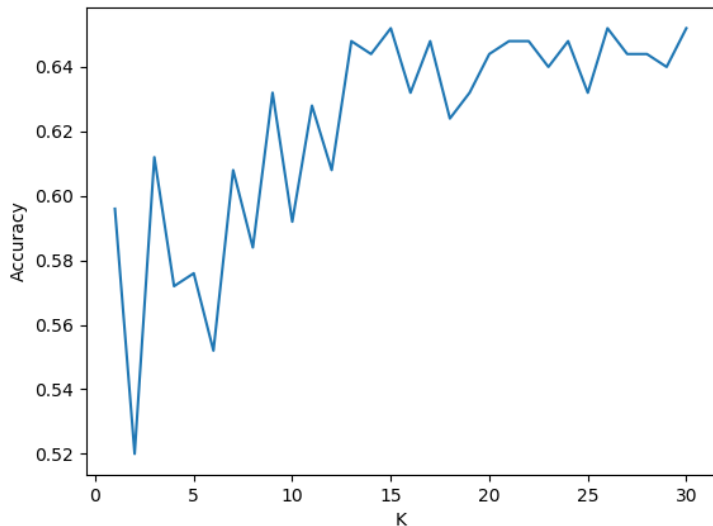
KNN is based on the idea that similar examples have similar label and classify new examples like similar training examples. It uses the algorithm that a new point is now assigned the **most frequent** label of its k nearest neighbors.

However the prediction accuracy can quickly degrade when number of attributes grows.

Model Design and Experiment: KNN

Experiment:

KNeighborsClassifier(n)



- Choose the most appropriate K between 1 to 31 firstly by comparing the accuracy.

The Best K = 14

- Use the SelectKBest (K=15) algorithm to select the attribute before modeling. And the cross validation (cv=3) also be used for preventing overfit.

Private Score: 0.672 Public Score: 0.664

Model Design and Experiment: Linear SVC

Description:

- Support Vector Classification (SVC) is

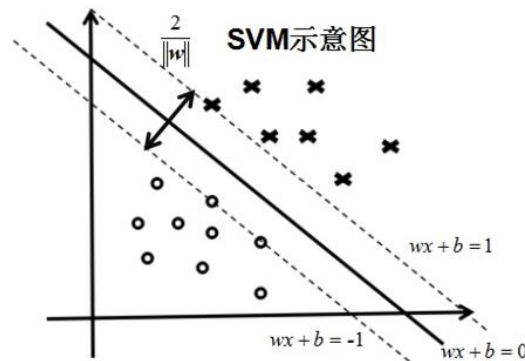
one classification of SVM:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^n \zeta_i$$

$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$

$$\zeta_i \geq 0, i = 1, \dots, n$$

- The main idea is that to **maximize** the 'distance' of the sample points **closest** to the hyperplane:

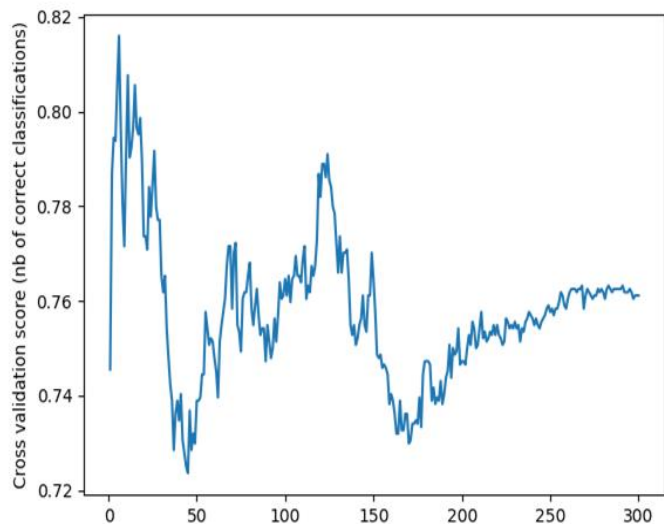


$$\begin{cases} \min \frac{1}{2} \|w\|^2 \\ \text{s.t. } y_i(w x_i + b) \geq 1, \quad \forall i \end{cases}$$

Model Design and Experiment: Linear SVC

Experiment:

SVC(kernel="linear")



Minimize the absolute value of hinge loss

- Use RFECV algorithm to do the feature selection and the cross-validation (cv=10).

N_feature = 6

Private Score: 0.689 Public Score: 0.700

- Use the KBest algorithm and cross-validation (cv=10)

SelectKBest(K=15): Private :0.725

Public: 0.734

SelectKBest(K=35): Private : 0.725

Public: 0.728

Model Design and Experiment: Decision Tree & Random Tree

Description:

Decision Tree:

Advantage: easy to interpret, we can know what variable and what value of that variable is used to split the data and predict outcome.

Disadvantage: easily overfit the data and doesn't know when we have crossed the line unless we are using cross validation.

Random Forest:

A Random Forest is essentially a collection of Decision Tree. It is similar with decision tree, but it is more accurate.

Model Design and Experiment: Decision Tree & Random Tree

Description: when to use this model?

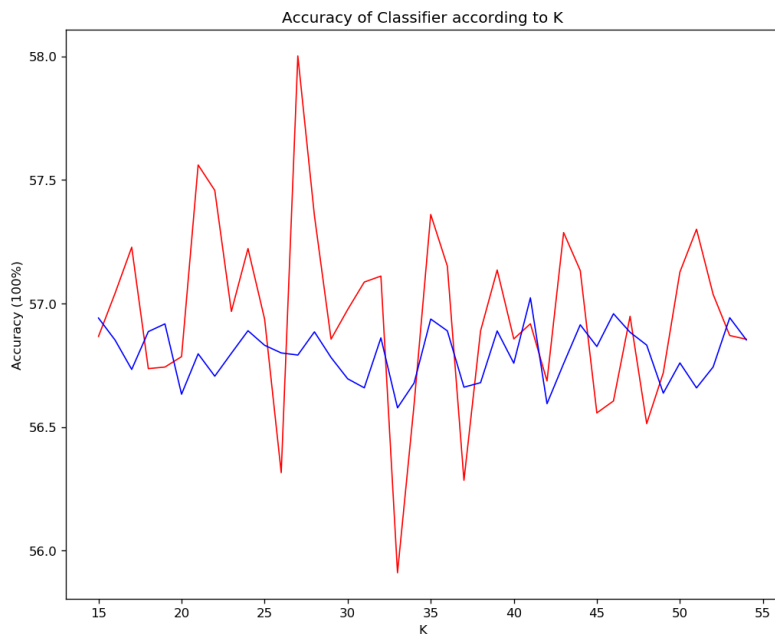
Decision Tree:

- Want a simple and explainable model
- Want a non parametric model
- Don't want to worry about feature selection or regularization or worry about multi-collinearity.

Random Forest:

- Don't bother much about interpreting the model but want better accuracy.

Model Design and Experiment: Decision Tree & Random Tree



Red: Random Forest; Blue: Decision Tree

100 iterations

Emperiment		
Function	DecisionTreeClassifier()	RandomForestClassifier()
# Attributes	41	27
# Cross Validation	3	3
Private Score	0.584	0.631
Public Score	0.598	0.641

Model Design and Experiment: AdaBoost Classifier

Description:

- Short for Adaptive Boosting, AdaBoost is sensitive to noisy data and outliers.

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.

Initialize: $D_1(i) = 1/m$ for $i = 1, \dots, m$.

For $t = 1, \dots, T$:

- Train weak learner using distribution D_t .
- Get weak hypothesis $h_t : \mathcal{X} \rightarrow \{-1, +1\}$.
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

- Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$.
- Update, for $i = 1, \dots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right).$$

- The idea is to set weights to both **classifiers** and **data points** in a way that forces classifiers to **concentrate** on observations that are **difficult** to correctly classify

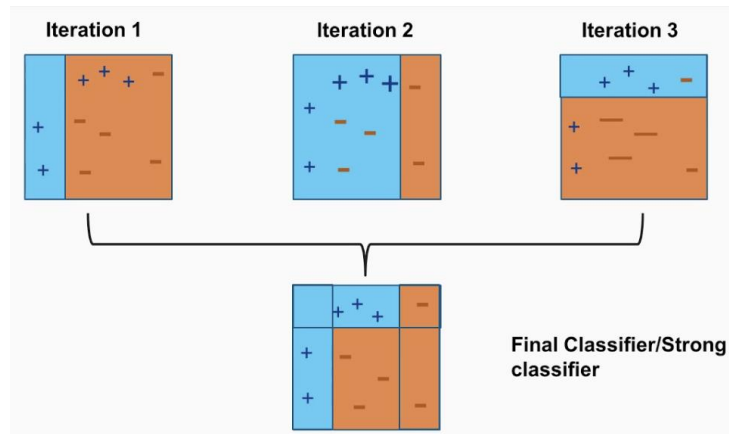
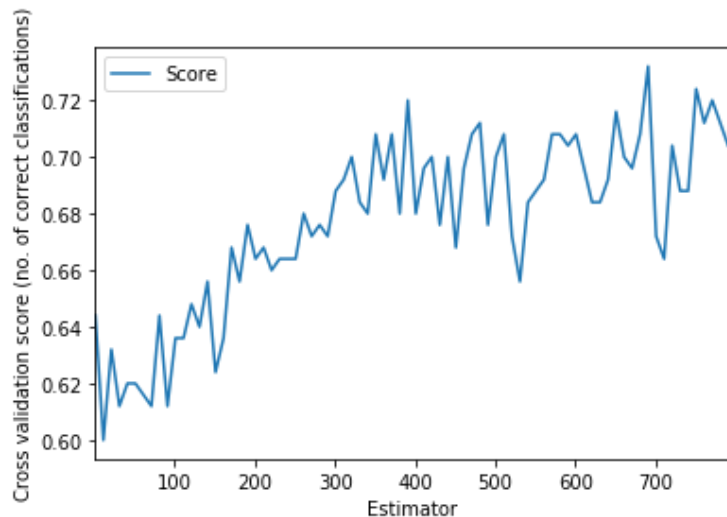


Fig. 1 The boosting algorithm AdaBoost.

Model Design and Experiment: AdaBoost Classifier

Experiment: AdaBoostClassifier(n)



Evaluate model generalization and performance

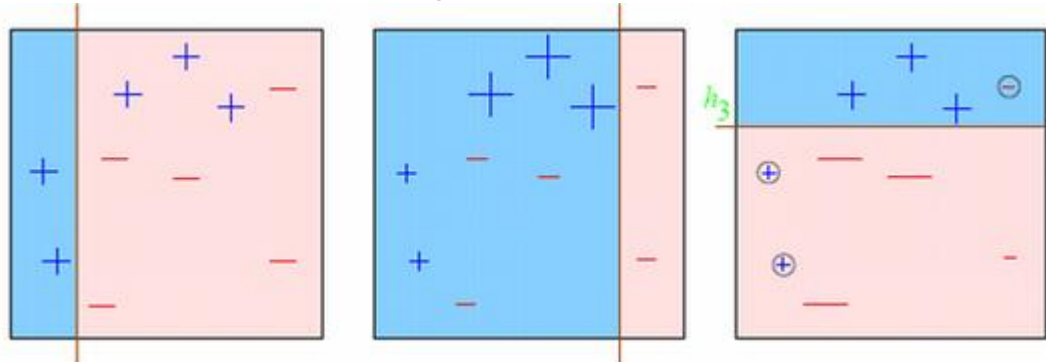
- Perform cross-validation (cv=10) on the AdaBoostClassifier model with different number of estimators
- Combine AdaBoost with Decision Tree Classifier as base estimator
- Choose the best number of estimators **max_depth=2, n_estimator=691**

Private Score: 0.741 Public Score: 0.754

Model Design and Experiment: Gradient Boosting Classifier

Description: Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

Algorithm: Gradient Boosting can combine a series of weak learners to improve the prediction accuracy of the overall model. At any time t , we give a weight to the current result according to the result obtained at $T-1$ time. The results of correct prediction get smaller weight, and the results of wrong classification get larger weight. For classification problem, the goal is let the model have a minimal logistic loss.



Model Design and Experiment: Gradient Boosting Classifier

Advantage: Fitting the training set too closely can lead to degradation of the model's generalization ability. By using some regularization method, such as shrinkage, early stopping, penalizing complexity of tree, we can reduce the overfitting effect by constraining the fitting procedure

Result: Private Score: 0.686 Public Score: 0.686

Model Design and Experiment: Generalized Linear Model

Description: The GLM generalizes linear regression by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value.

Algorithm: Since our response data, Y , are binary (taking on only values 0 and 1), the distribution function is generally chosen to be the Binomial distribution. And we use Logit link function.

Result: Private Score: 0.825 Public Score: 0.831

Model Design and Experiment: Linear Regression

Description:

Linear Regression is a linear approach to model the relationship between a scalar response and one or more explanatory variables.

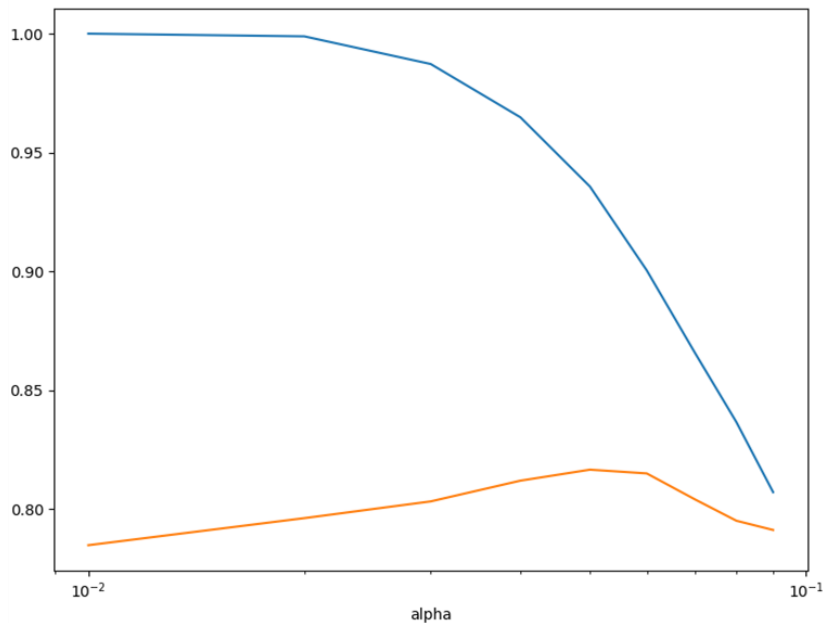
Tackle overfit using regularization:

- **Ridge Regression:** $Loss = \sum (\hat{Y}_i - Y_i)^2 + \lambda \sum \beta^2$
- **Lasso Regression:** $Loss = \sum (\hat{Y}_i - Y_i)^2 + \lambda \sum |\beta|$

Model Design and Experiment: Linear Regression

Experiment: Lasso Regression

- Use **LassoCV** to find out the best parameter **alpha**=0.03.
- Use a for-loop with **LassoCV** inside to find out the best cross-validation value **cv**=9.
- Use **RFECV** algorithm to do the feature selection and the cross-validation (cv=9)



Result: Private Score: 0.849 Public Score: 0.859

Model Design and Experiment: Linear Regression

Experiment: Ridge Regression

- Use **RidgeCV** to find out the best parameter **alpha**=2400.
ridge_model = Ridge(max_iter=10000, alpha=2400)
- Use a for-loop with **RidgeCV** inside to find out the best cross-validation value **cv**=6.
- Use **RFECV** algorithm to do the feature selection and the cross-validation (cv=6)
ridge = RFECV(ridge_model, step=1, cv=6, scoring='roc_auc')

Result: Private Score: 0.838 Public Score: 0.850

Model Design and Experiment: Logistic Regression

Description:

Logistic Regression is used to predict the probability of a categorical dependent variable.

Experiment:

- Use **LogisticRegressionCV** to find out the best parameter **C=0.2**.
- Use **L1 (Lasso)** as penalty for a better performance.

```
lr = LogisticRegression(class_weight='balanced', solver='liblinear', penalty='l1',  
C=0.2, max_iter=10000)
```

Result: Private Score: 0.852 Public Score: 0.862

Performance Evaluation - Best Model

Model ranking:

1. Lasso Regression	private 0.852 public 0.863	<div>Best</div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div>Worst</div>
2. Logistic Regression	private 0.837 public 0.849	
3. Ridge Regression	private 0.838 public 0.850	
4. Generalized Linear Model	private 0.825 public 0.831	
5. Ada Boost	private 0.741 public 0.754	
6. Linear SVC	private 0.689 public 0.700	
7. Gradient Boosting	private 0.686 public 0.686	
8. KNN	private 0.672 public 0.664	
9. Random Forest	private 0.631 public 0.641	
10. Decision Tree	private 0.584 public 0.598	

Results indicate an improvement in performance for linear models

Performance Evaluation – Final Model

Model ranking:

- | | |
|------------------------|----------------------------|
| 1. Logistic Regression | private 0.852 public 0.862 |
| 2. Lasso Regression | private 0.849 public 0.859 |
| 3. Ridge Regression | private 0.838 public 0.850 |

Select top-3 models and combine the results by assigning weight:

$$\text{preds} = 0.4 * \text{preds_logreg} + 0.2 * \text{preds_lasso} + 0.4 * \text{preds_ridge}$$

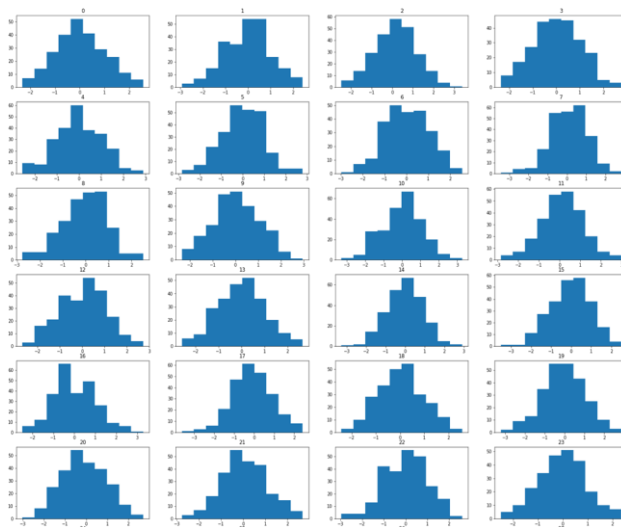
Final Result:

Submission and Description	Private Score	Public Score	Use for Final Score
sub_RFE.csv 2 days ago by JYU999 add submission details	0.856	0.868	<input type="checkbox"/>

Performance Evaluation - Feature Distribution

Observation

1. Target (output) is binary, 64% belong to 1 and 36% belong to 0
2. Features are most likely independent since correlation between features are low
3. Value range of different features are similar
4. Logistic and Lasso regression performing the best indicate a linear distribution of features

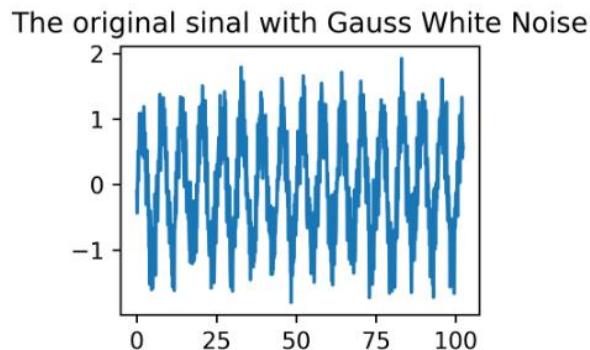
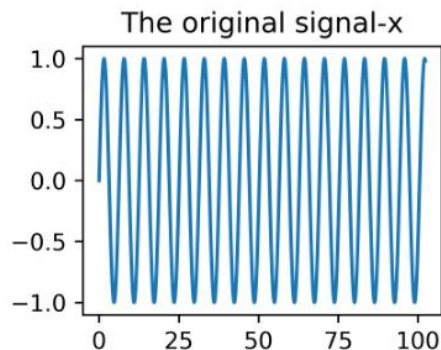


Future Development: White Gauss Noise

Description: The power spectral density of Gaussian white noise follows a uniform distribution, and the amplitude distribution obeys a Gaussian distribution.

$$\text{SNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}} = \left(\frac{A_{\text{signal}}}{A_{\text{noise}}} \right)^2$$

```
def wgn(x, snr):  
    snr = 10**(snr/10.0)  
    xpower = np.sum(x**2)/len(x)  
    npower = xpower / snr  
    return np.random.randn(len(x)) * np.sqrt(npower)
```



Future Development: Soft Margin SVM

Allow a small number of samples to not satisfy the constraints: $y_i(X_i^T W + b) \geq 1$

In order to make the number of sample points that do not meet the above conditions as small as possible, we need to add a penalty term for these points in the optimized objective function: $l_{hinge}(z) = \max(0, 1 - z)$

$$\min_{W, b, \xi} \quad \frac{1}{2} \|W\|^2 + C \sum_{i=1}^n \xi_i$$

$$s. t. \quad y_i(X_i^T W + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0, i = 1, 2, \dots, n.$$

C called penalty parameter