Python Data Analytics

OBA 410/510

Lundquist College of Business

Background image from www. givweb.com

Some of the slides and contents are adopted from Statistical Learning course by Dr. Trevor Hastie

Two more Classification Techniques

Classification

- Regression: Target variable (y) is numerical
- Classification: Target variable (y) is binary or categorical
- Examples:
  - Is an email legitimate or spam?
  - Will a flight be on time or late?
  - Will a student receive an High Pass, Pass, Low Pass or Fail?
  - Is it an appropriate time to buy or sell a stock?
  - Will a customer choose to buy or not buy?
- The prediction in many Classification techniques is the probability of categories (or labels) for given set of input variables (X)

Default data

- Default of individual credit card payment ; 10,000 records
- Target : default (y)
  - Yes, No
- Features (Predictors) (X)
  - student (Yes(1), No(0)), balance, income

	default	student	balance	income
0	No	0	729.526495	44361.625070
1	No	1	817.180407	12106.134700
2	No	0	1073.549164	31767.138950
3	No	0	529.250605	35704.493940
4	No	0	785.655883	38463.495880
5	No	1	919.588530	7491.558572
6	No	0	825.513330	24905.226580
7	No	1	808.667504	17600.451340
8	No	0	1161.057854	37468.529290
9	No	0	0.000000	29275.268290

UNIVERSITY OF OREGON
Lundquist College of Business

Descriptive Analysis

```
no=df_default['default']=='No'
yes=df_default['default']=='Yes'
```

- Individuals who default tend to have **higher** balance
- Individuals who default tend to have **lower** income

```
df_default[no].describe()
```

	student	balance	income
count	9667.000000	9667.000000	9667.000000
mean	0.291404	803.943750	33566.166625
std	0.454433	456.476236	13318.251249
min	0.000000	0.000000	771.967729
25%	0.000000	465.714646	21405.060665
50%	0.000000	802.857102	34589.488060

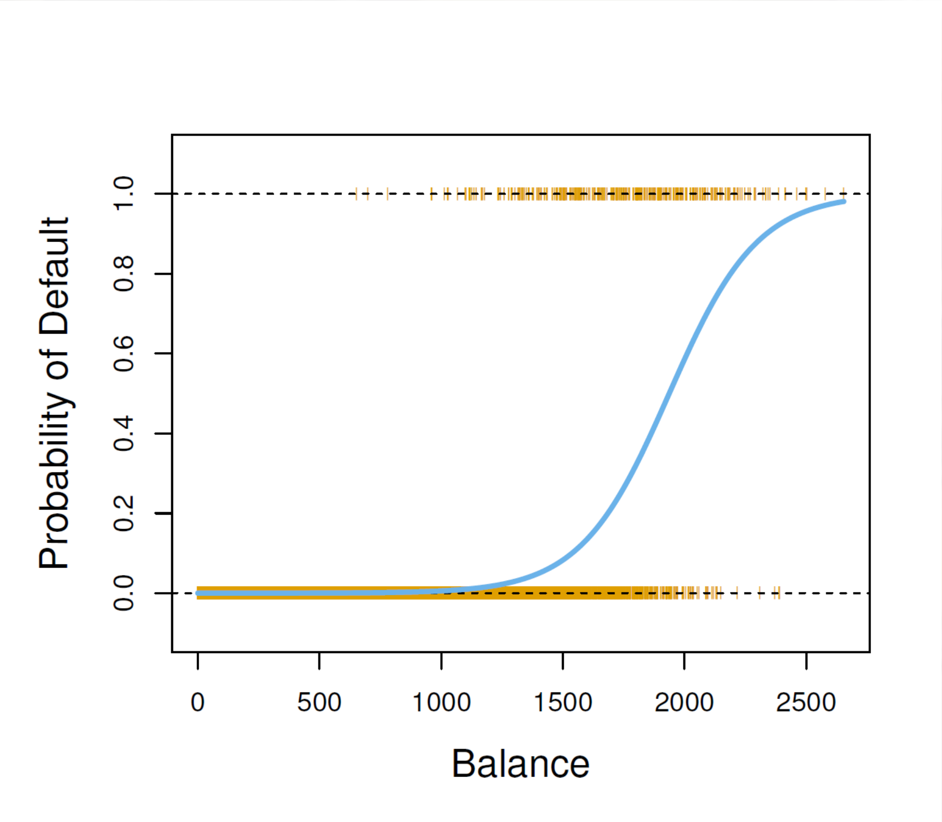
```
df_default[yes].describe()
```

	student	balance	income
count	333.000000	333.000000	333.000000
mean	0.381381	1747.821690	32089.147124
std	0.486457	341.266808	13804.221110
min	0.000000	652.397134	9663.788159
25%	0.000000	1511.610952	19027.508630
50%	0.000000	1789.093391	31515.344490

Simple Logistic Regression

UNIVERSITY OF OREGON
Lundquist College of Business

- Let's consider probability that an individual will default depends only on "balance"



$$\Pr(\text{default} = \text{Yes}) = \frac{e^{\beta_0 + \beta_1 \text{ balance}}}{1 + e^{\beta_0 + \beta_1 \text{ balance}}}$$

(Logistic function)

$$\Pr(\text{default} = \text{No}) = \frac{1}{1 + e^{\beta_0 + \beta_1 \text{ balance}}}$$

$$\Pr(\text{default} = \text{Yes}) + \Pr(\text{default} = \text{No}) = 1$$

Manipulation

$$\frac{\Pr(\text{default} = \text{Yes})}{1 - \Pr(\text{default} = \text{Yes})} = e^{\beta_0 + \beta_1 \text{ balance}}$$

- Left hand side of the above equation is call “odds”
- Odds close to **0** indicate **<u>low</u>** probability of default
- Odds close to **1** indicate **<u>high</u>** probability of default

$$\ln\left(\frac{\Pr(\text{default} = \text{Yes})}{1 - \Pr(\text{default} = \text{Yes})}\right) = \beta_0 + \beta_1 \text{ balance}$$

- Left hand side of the above example is call “log-odds”

Logistic Regression in scikit-learn

```
X,y=df_default[['balance']],df_default['default']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,random_state=0)
```

```
from sklearn.linear_model import LogisticRegression
```

```
log_reg=LogisticRegression()
log_reg.fit(X_train,y_train)
```

```
print('Log reg acc on train: {:.3f}'.format(log_reg.score(X_train,y_train)))
print('Log reg acc on test: {:.3f}'.format(log_reg.score(X_test,y_test)))
```

```
Log reg acc on train: 0.973
Log reg acc on test: 0.968
```

```
p1=[1000]
p2=[2000]
p3=[3000]
print('Predcited traget for p1, p2, and p3:',log_reg.predict([p1,p2,p3]))
```

```
Predcited traget for p1, p2, and p3: ['No' 'No' 'Yes']
```

```
log_reg.predict_proba([p1,p2,p3])
```

```
array([[0.98723001, 0.01276999],
       [0.54594197, 0.45405803],
       [0.01835677, 0.98164323]])
```

```
log_reg.classes_
```

```
array(['No', 'Yes'], dtype=object)
```

```
print('Probablities for p1, p2, and p3:',log_reg.predict_proba([p1,p2,p3])[:,1])
```

```
Probablities for p1, p2, and p3: [0.01276999 0.45405803 0.98164323]
```

Logistic Regression

$$\Pr(\text{target vriable}) = \text{logitic}(\beta_0 + \beta_1\, x_1 + \beta_2\, x_2 + \cdots + \beta_p\, x_p)$$

$$\Pr(\text{target vriable}) = \frac{e^{(\beta_0 + \beta_1\, x_1 + \beta_2\, x_2 + \cdots + \beta_p\, x_p)}}{1 + e^{(\beta_0 + \beta_1\, x_1 + \beta_2\, x_2 + \cdots + \beta_p\, x_p)}}$$

$$\ln\left(\frac{\Pr(\text{target vriable})}{1 - \Pr(\text{target vriable})}\right) = \beta_0 + \beta_1\, x_1 + \beta_2\, x_2 + \cdots + \beta_p\, x_p$$

- Coefficients (β_0, β_1,…) of the model are estimated using “Maximum Likelihood”

Logis

```
X,y=diabetes.iloc[:,:-1],diabetes['Outcome']
X_train, X_test, y_train, y_test=train_test_split(X,y,random_state=0)
```

```
log_reg2=LogisticRegression()
log_reg2.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
print('Log reg2 acc on train: {:.3f}'.format(log_reg2.score(X_train,y_train)))
print('Log reg2 acc on test: {:.3f}'.format(log_reg2.score(X_test,y_test)))
```

```
Log reg2 acc on train: 0.757
Log reg2 acc on test: 0.807
```

```
p1=[3,150,80,22,10,40,2.3,66]
log_reg2.predict([p1])
```

```
array([1], dtype=int64)
```

```
log_reg2.predict_proba([p1])
```

```
array([[0.14143716, 0.85856284]])
```

```
log_reg2.classes_
```

```
array([0, 1], dtype=int64)
```

- Diabetes dataset

Logistic Regression

- By default, logistic regression applies an L2 regularization (in the same way that Ridge does for regression)
- L2 regularization, forces coefficients' magnitude to be closer to zero
- For ***LogisticRegression***, the trade-off parameter that determines the strength of the regularization is called ***C***.
- Higher values of ***C*** correspond to <u>**less**</u> regularization
  - Higher C, LogisticRegression tries to fit the training set as best as possible
  - Lower C, more regularization, coefficients (β_is) closer to zero

Log

```
cancer=pd.read_csv('breast_cancer_data.csv',index_col=0)
cancer.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	sm
id						
842302	M	17.99	10.38	122.80	1001.0	
842517	M	20.57	17.77	132.90	1326.0	

```
X,y=cancer.iloc[:,1:],cancer['diagnosis']
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0)
```

```
log_reg3=LogisticRegression()
log_reg3.fit(X_train,y_train)
print('Log reg3 acc on train: {:.3f}'.format(log_reg3.score(X_train,y_train)))
print('Log reg3 acc on test: {:.3f}'.format(log_reg3.score(X_test,y_test)))
```

```
Log reg3 acc on train: 0.960
Log reg3 acc on test: 0.958
```

Logistic Regression in scikit-learn

```
log_reg4=LogisticRegression(C=100)
log_reg4.fit(X_train,y_train)
print('Log reg4 acc on train: {:.3f}'.format(log_reg4.score(X_train,y_train)))
print('Log reg4 acc on test: {:.3f}'.format(log_reg4.score(X_test,y_test)))
```

```
Log reg4 acc on train: 0.967
Log reg4 acc on test: 0.965
```

```
log_reg5=LogisticRegression(C=.1)
log_reg5.fit(X_train,y_train)
print('Log reg5 acc on train: {:.3f}'.format(log_reg5.score(X_train,y_train)))
print('Log reg5 acc on test: {:.3f}'.format(log_reg5.score(X_test,y_test)))
```

```
Log reg5 acc on train: 0.951
Log reg5 acc on test: 0.944
```

University of Oregon
Lundquist College of Business

Transforming the data (scaling)

- Some techniques are sensitive to the scale of variables in data
  - Different scales in various variables
  - High variability in a single variable (***default*** example with ***balance*** as the only feature)
- Therefore, before developing these models, it is a good idea to transform all the variables to the same scale
- One way to do this is using the *MinMaxScaler* function
- This function transforms all variables to a between 0 and 1 scale
- It uses this formula:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

UNIVERSITY OF OREGON
Lundquist College of Business

Transforming the data (scaling)

x1	x2
100	1.1
110	1
130	1.2
115	3
125	2.2

		x1	x1_new
max	130	100	0
min	100	110	0.3
		130	1
		115	0.5
		125	0.8

		x2	x2_new
max	3	1.1	0.05
min	1	1	0
		1.2	0.1
		3	1
		2.2	0.6

x1_new	x2_new
0.0	0.05
0.3	0
1.0	0.1
0.5	1
0.8	0.6

Evaluating the Effect of Parameters

- ***validation_curve*** function
    - Determines training and test scores for varying parameter values.
- Function inputs:
    - estimator: object type that implements the “fit” and “predict” methods
    - X: Training features
    - y: Target
    - param_name: Name of the parameter that will be varied.
    - param_range: The values of the parameter that will be evaluated.
    - cv : Determines the cross-validation splitting strategy

3 breast cancer data

```
# reading the data
cancer=pd.read_csv('breast_cancer_data.csv',index_col=0)
cancer.head(1)
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compa
id							
842302	M	17.99	10.38	122.8	1001.0	0.1184	

1 rows × 31 columns

```
X_cancer, y_cancer=cancer.iloc[:,1:], cancer['diagnosis']
display(X_cancer.head(1))
display(y_cancer.head(2))
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mea
id						
842302	17.99	10.38	122.8	1001.0	0.1184	0.277

1 rows × 30 columns

```
id
842302    M
842517    M
Name: diagnosis, dtype: object
```

```
]: from sklearn.preprocessing import MinMaxScaler
```

```
]: scaler1=MinMaxScaler()
   X_cancer_trns=scaler1.fit_transform(X_cancer)
   X_cancer_trns
```

```
]: array([[0.52103744, 0.0226581 , 0.54598853, ..., 0.91202749, 0.59846245,
            0.41886396],
           [0.64314449, 0.27257355, 0.61578329, ..., 0.63917526, 0.23358959,
            0.22287813],
           [0.60149557, 0.3902604 , 0.59574321, ..., 0.83505155, 0.40370589,
```

```python
from sklearn.model_selection import validation_curve
```

```python
C_range=[.1,.5,1,10,100,200]
```

```python
# validation_curve has two outputs: 1- score on traning sets 2- scores on test sets
# order is important
train_scores, test_scores=validation_curve(LogisticRegression(solver='liblinear'),
                                            X_cancer, y_cancer,
                                           param_name='C',param_range=C_range, cv=4)
```

```python
train_scores.round(4)
```

```
array([[0.9531, 0.9532, 0.9438, 0.9578],
       [0.9577, 0.9555, 0.9508, 0.9672],
       [0.9554, 0.9578, 0.9555, 0.9649],
       [0.9695, 0.9696, 0.9649, 0.9696],
       [0.9765, 0.9696, 0.9672, 0.9789],
       [0.9742, 0.9742, 0.9789, 0.9742]])
```

```python
print('ave cross val scores on train:',train_scores.mean(axis=1).round(4))
```

```
ave cross val scores on train: [0.952  0.9578 0.9584 0.9684 0.9731 0.9754]
```

```python
test_scores.round(4)
```

```
array([[0.9021, 0.9648, 0.9577, 0.9085],
       [0.9301, 0.9366, 0.9718, 0.9296],
       [0.9301, 0.9366, 0.9718, 0.9366],
       [0.9371, 0.9507, 0.9718, 0.9437],
       [0.9301, 0.9507, 0.9718, 0.9507],
       [0.9301, 0.9437, 0.9718, 0.9437]])
```

```python
print('ave cross val scores on test:',test_scores.mean(axis=1).round(4))
```

```
ave cross val scores on test: [0.9333 0.942  0.9438 0.9508 0.9508 0.9473]
```

Support Vector Machines

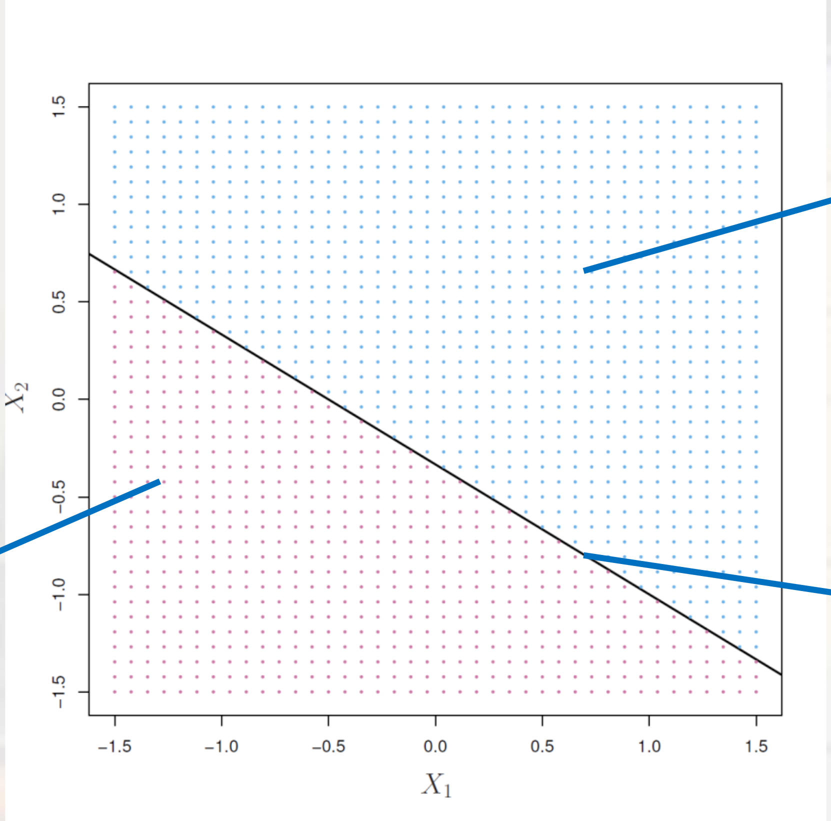
Support Vector Machine

- Approach for Classification (also for regression, but in this class we only use it for classification)
- Developed by Computer Scientists in 1990’s
- In short stated as “SVM”
- The are two type of SVMs:
  - Linear SVM
  - Non-linear SVM

Terminology

- Hyperplane
  - In two dimension space: a line
  - In three dimension space: a plane
- Equation of the hyperplane in a p-dimensional space:
  - $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$
- Example : A hyperplane in two-dimensional space $1 + 2X_1 + 3X_2 = 0$

Terminology



$1 + 2X_1 + 3X_2 > 0$

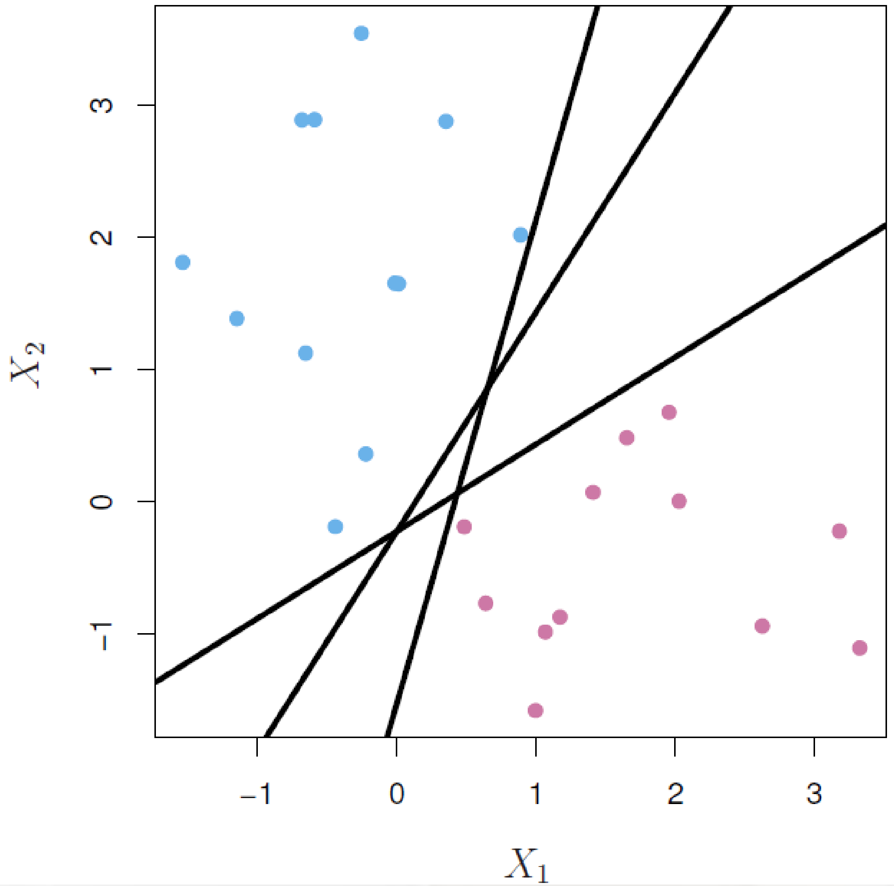
$1 + 2X_1 + 3X_2 = 0$

Separating Hyperplane
(decision boundary)

$1 + 2X_1 + 3X_2 < 0$

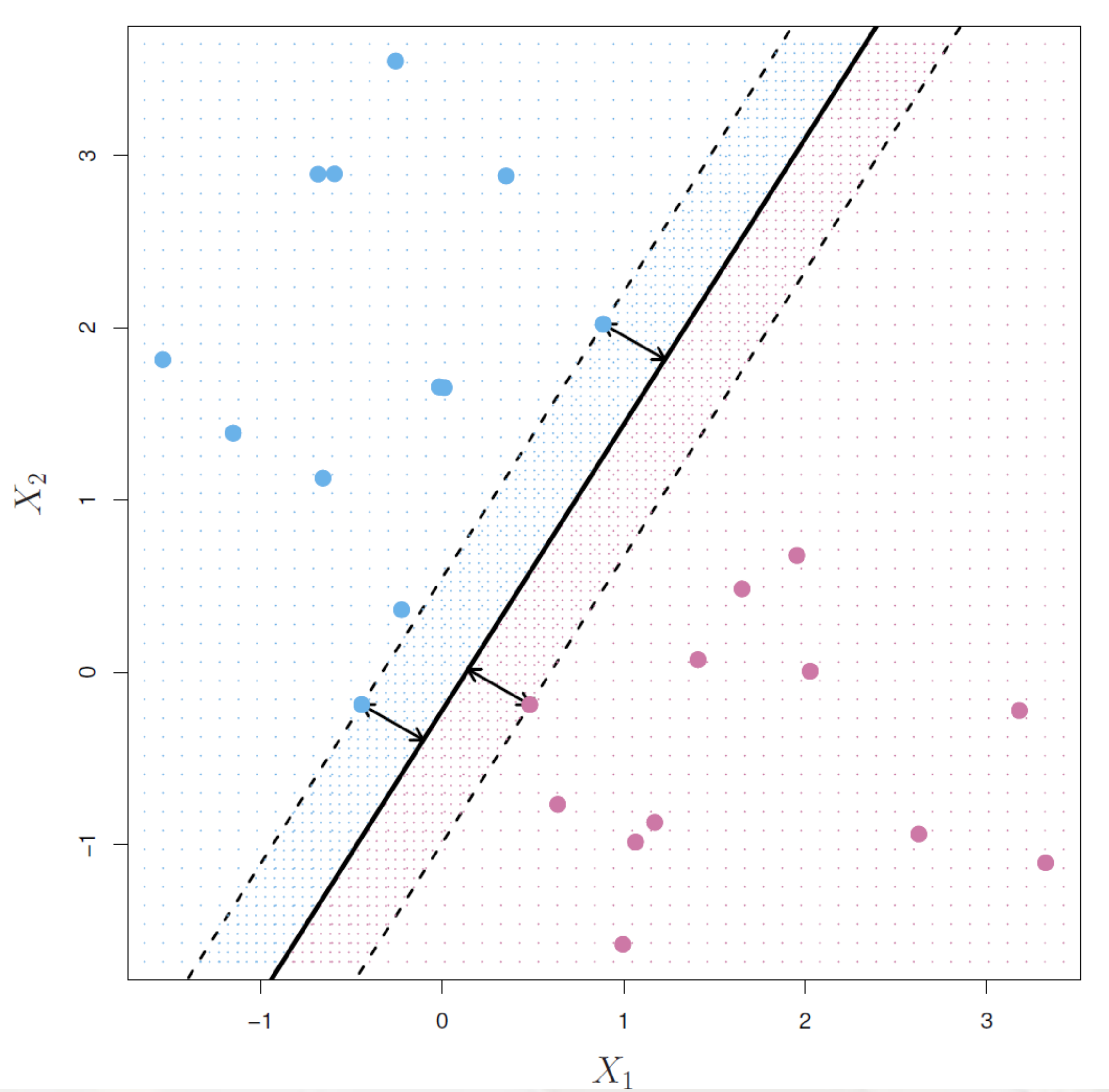
Classification using Separating Hyperplane

- Suppose we have training data of "n" observations and "p" variables
- Assume each of the observations fall into one of the two classes $\{-1, 1\}$
- Our goal is to classify a new test data, x^* with "p" variables into one of these classes
- All are hyperplanes shown in black are equally good



Maximum Margin Classifier

- Among all the hyperplanes, maximum margin classifier is the one that makes the biggest gap between the two classes
- The margin is the width that the decision boundary can be increased before hitting a data point.
- The objective in SVM
  - Minimizing the classification error and maximizing the margin between two classes



UNIVERSITY OF OREGON
Lundquist College of Business

Support Vector Machine

Feature vector   Class value (target prediction)

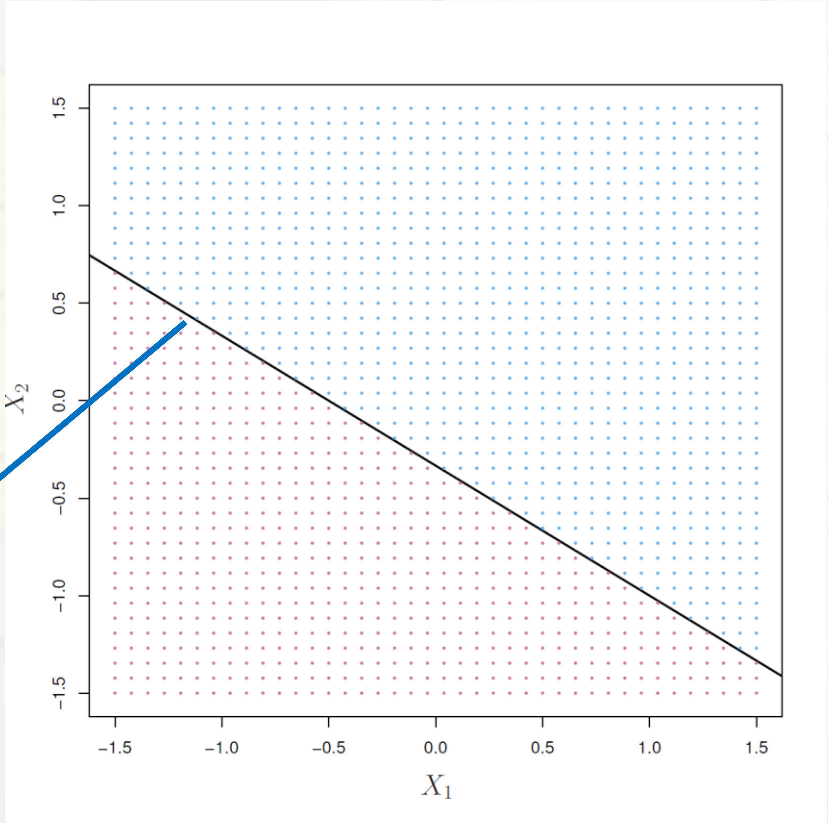
$$\boldsymbol{X} \longrightarrow \boxed{\boldsymbol{f}} \longrightarrow \hat{\boldsymbol{y}}$$

- $f(X, \beta) = sign(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p)$

- X=(-1, 0)   what is the class?
- X=(1, 1)

$$1 + 2X_1 + 3X_2 = 0$$

Separating Hyperplane
(decision boundary)



Transforming the data (scaling) before using SVM

- SVM is very sensitive to the scale of variables in data
    - Different scales in various variables
    - High variability in a single variable (***default*** example with ***balance*** as the only feature)
- Therefore, before developing SVM models, it is a good idea to transform all the variables to the same scale
- One way to do this is using the *MinMaxScaler* function
- This function transforms all variables to a between 0 and 1 scale
- It uses this formula:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Transforming the data (scaling) before using SVM

UNIVERSITY OF OREGON
Lundquist College of Business

x1	x2
100	1.1
110	1
130	1.2
115	3
125	2.2

		x1	x1_new
max	130	100	0
min	100	110	0.3
		130	1
		115	0.5
		125	0.8

		x2	x2_new
max	3	1.1	0.05
min	1	1	0
		1.2	0.1
		3	1
		2.2	0.6

x1_new	x2_new
0.0	0.05
0.3	0
1.0	0.1
0.5	1
0.8	0.6

UNIVERSITY OF OREGON
Lundquist College of Business

SVM in scikit-learn

```
: from sklearn.svm import LinearSVC
```

```
: df_default.head(n=2)
```

:

	default	student	balance	income
0	No	0	729.526495	44361.62507
1	No	1	817.180407	12106.13470

```
: X_def, y_def=df_default[['balance']],df_default['default']
```

```
91]: X_def.describe()
```

91]:

	balance
count	10000.000000
mean	835.374886
std	483.714985
min	0.000000
25%	481.731105
50%	823.636973
75%	1166.308387
max	2654.322576

SVM in scikit-learn

- LinearSVC, has its own random number generator
- If we do not specify the random_state option, every time we fit our model, we will have different scores

```
71]: from sklearn.preprocessing import MinMaxScaler

72]: scaler1=MinMaxScaler()

73]: X_def_trs=scaler1.fit_transform(X_def)
     X_def_trs

73]: array([[0.2748447 ],
            [0.30786778],
            [0.40445316],
            ...,
            [0.31850386],
            [0.59111468],
            [0.07569622]])

74]: X_train,X_test, y_train, y_test=train_test_split(X_def_trs,y_def,random_state=0)

92]: svm=LinearSVC(random_state=0)
     svm.fit(X_train,y_train)
```

SVM in scikit-learn

Test data points have to be transformed before making predictions

```
]: svm.coef_
]: array([[3.01440675]])

]: svm.intercept_
]: array([-2.41896078])

]: # -2.418+3.014x
```

```
93]: print('svm on train: {:.2%}'.format(svm.score(X_train,y_train)))
     print('svm on test: {:.2%}'.format(svm.score(X_test,y_test)))

     svm on train: 97.11%
     svm on test: 96.60%

94]: p=[[800],[5000]]
     p_trs=scaler1.transform(p)
     p_trs

94]: array([[0.30139517],
            [1.8837198 ]])

95]: svm.predict(p_trs)

95]: array(['No', 'Yes'], dtype=object)
```

Regularization Parameter

- Similar to LogisticRegression,
- By default, ***LinearSVC*** applies an L2 regularization (in the same way that Ridge does for regression)
- L2 regularization, forces coefficients' magnitude to be closer to zero
- For ***LinearSVC***, the trade-off parameter that determines the strength of the regularization is called ***C***.
- Higher values of ***C*** correspond to **less** regularization
  - Higher C, LogisticRegression tries to fit the training set as best as possible
  - Lower C, more regularization, coefficients (β_is) closer to zero

SVM in scikit-learn

1.2 breast cancer data

]:
```
cancer=pd.read_csv('breast_cancer_data.csv', index_col=0)
cancer.head(1)
```

]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	conca
id								
842302	M	17.99	10.38	122.8	1001.0	0.1184	0.2776	

1 rows × 31 columns

]:
```
# create features and traget sets
X_cancer, y_cancer=cancer.iloc[:,1:], cancer['diagnosis']
display(X_cancer.head(1))
display(y_cancer.head(2))
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
id							
842302	17.99	10.38	122.8	1001.0	0.1184	0.2776	0.3001

1 rows × 30 columns

```
id
842302    M
842517    M
Name: diagnosis, dtype: object
```

UNIVERSITY OF OREGON
Lundquist College of Business

SVM in scikit-learn

```
# transforming the features
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import LinearSVC
from sklearn.model_selection import validation_curve
```

```
scaler2=MinMaxScaler()
X_cancer_trns=scaler2.fit_transform(X_cancer)
X_cancer_trns[:1,:]
```

```
array([[0.52103744, 0.0226581 , 0.54598853, 0.36373277, 0.59375282,
        0.7920373 , 0.70313964, 0.73111332, 0.68636364, 0.60551811,
        0.35614702, 0.12046941, 0.3690336 , 0.27381126, 0.15929565,
        0.35139844, 0.13568182, 0.30062512, 0.31164518, 0.18304244,
        0.62077552, 0.14152452, 0.66831017, 0.45069799, 0.60113584,
        0.61929156, 0.56861022, 0.91202749, 0.59846245, 0.41886396]])
```

```
C_range=[.1,1,5,10,50,100]
```

```
train_scores, test_scores=validation_curve(LinearSVC(random_state=0, max_iter=100000)
                                           ,X_cancer_trns,y_cancer,
                                           param_name='C', param_range=C_range, cv=4)
```

SVM in scikit-learn

```
[42]:  train_scores.round(4)

[42]: array([[0.9624, 0.9578, 0.9672, 0.9696],
             [0.9812, 0.9789, 0.9859, 0.9766],
             [0.9883, 0.9859, 0.9883, 0.9813],
             [0.9883, 0.9883, 0.9906, 0.9813],
             [0.993 , 0.9906, 0.9906, 0.9836],
             [0.9953, 0.9906, 0.9906, 0.9836]])

[43]:  test_scores.round(4)

[43]: array([[0.951 , 0.9437, 0.9577, 0.9718],
             [0.972 , 0.9859, 0.9648, 0.9859],
             [0.965 , 0.9789, 0.9648, 0.9859],
             [0.972 , 0.9789, 0.9648, 0.9789],
             [0.958 , 0.9789, 0.9859, 0.9648],
             [0.951 , 0.9577, 0.9859, 0.9577]])

[44]:  print('ave cross val scores on train:',train_scores.mean(axis=1).round(3))

ave cross val scores on train: [0.964 0.981 0.986 0.987 0.989 0.99 ]

[45]:  print('ave cross val scores on test:',test_scores.mean(axis=1).round(3))

ave cross val scores on test: [0.956 0.977 0.974 0.974 0.972 0.963]
```

Linear Models
parameters, strengths, weaknesses

Linear Models

parameters, strengths, weaknesses

- Linear regression, Lasso, Ridge
- Logistic regression, Linear support vector machine
- The main parameter is the regularization parameter
    - ***alpha*** in regression models and ***C*** in LinearSVC and LogisticRegression
    - Large values for ***alpha*** or small values for ***C*** mean simpler models
    - In particular for regression models, tuning this parameter is very important

Linear Models

parameters, strengths, weaknesses

- The other parameter in linear models is the regularization type
  - L1 versus L2
  - Default for LinearSVC, LogisticRegression and Ridge is L2
  - Lasso uses L1
  - If you have many features and think only a few of them are important, you should use L1, otherwise use the default (L2)
  - As L1 only uses a few features, it is easier to explain

Linear Models

parameters, strengths, weaknesses

- Linear models are very fast to train, also fast to predict
- They are suitable for very large and sparse datasets
- They perform well when the number of features is large compared to the number of samples (records)
- Linear models are usually easy to understand because of their linear nature!
- Linear models might not perform well in lower-dimensional spaces (fewer features in data)