MITB ISSS610
Applied Machine Learning

Instructors:

Dr. Steven HOI

Dr. DAI Bing Tian

School of Information Systems

Singapore Management University

Outline

- Naïve Bayes Classifier
- Logistic Regression

Naïve Bayes Classifier

Naïve Bayes Classifier

- Given training data sampled from K classes:

$$(\mathbf{x}_i, y_i), i = 1, \ldots, n$$

- We are interested in $p(\mathcal{C}_k|\mathbf{x})$ not $p(\mathbf{x}|\mathcal{C}_k)$

$$p(\mathcal{C}_k|\mathbf{x}) \propto \boxed{p(\mathbf{x}|\mathcal{C}_k)}\boxed{p(\mathcal{C}_k)}$$

Density function for class C_k

Class prior

Naïve Bayes Classifier

- Assumption – All attributes are conditionally independent

$$p(\mathbf{x}|\mathcal{C}_k) \approx \prod_{j=1}^{d} p(x_j|\mathcal{C}_k)$$

- Naïve Bayes Classifier:

$$\mathcal{C}_{NB} = \arg\max_{\mathcal{C}_k} P(\mathcal{C}_k) \prod_j P(x_j|\mathcal{C}_k)$$

Iris Example (I)

```
from sklearn import datasets, naive_bayes

iris = datasets.load_iris()
x = iris.data
y = iris.target

gnb = naive_bayes.GaussianNB()
gnb.fit(x[:,:2], y)

print(gnb.class_prior_)
print(gnb.class_count_)
print(gnb.theta_)
print(gnb.sigma_)
```

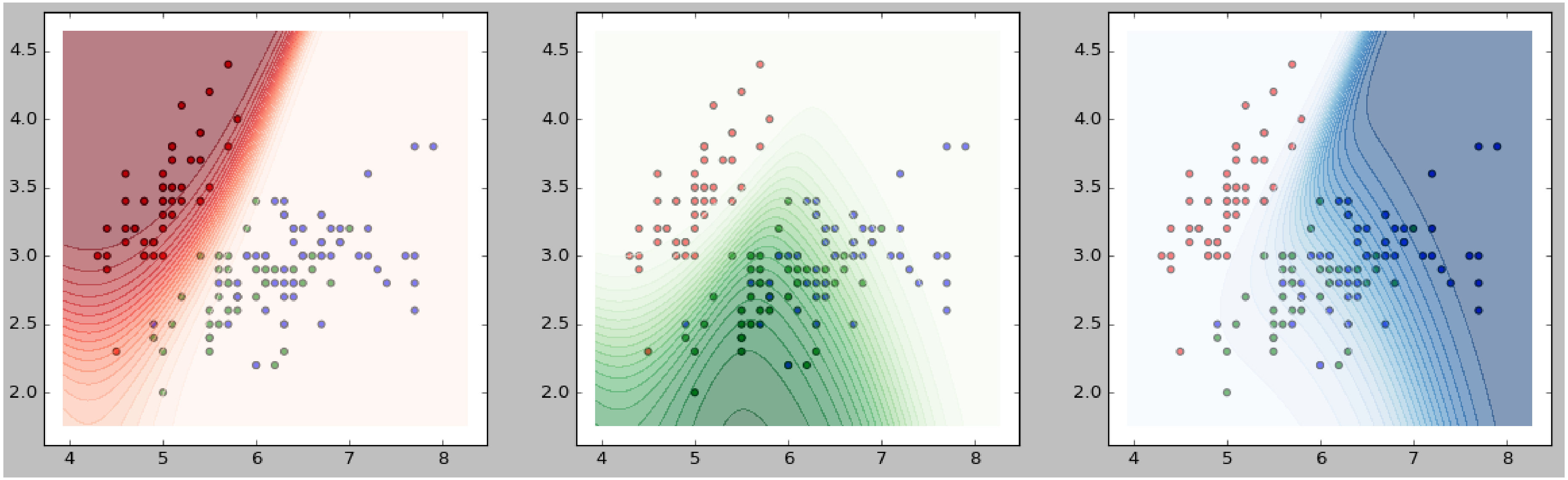
Iris Example (II)

```
import numpy as np

def my_linspace (min_value, max_value, steps):
    diff = max_value - min_value
    return np.linspace (min_value - 0.1 * diff, max_value + 0.1 * diff, steps)

steps = 200

x0 = my_linspace(min(x[:,0]), max(x[:,0]), steps)
x1 = my_linspace(min(x[:,1]), max(x[:,1]), steps)
xx0, xx1 = np.meshgrid(x0, x1)
mesh_data = np.c_[xx0.ravel(), xx1.ravel()]
mesh_class = gnb.predict(mesh_data)
mesh_proba = gnb.predict_proba(mesh_data).reshape(steps, steps, 3)
```

Iris Example (III)

```
import matplotlib.pyplot as plt

color = ['red', 'green', 'blue']
y_color = [color[i] for i in y]

plt.figure(figsize = (12, 3))
contour_color = [plt.cm.Reds, plt.cm.Greens, plt.cm.Blues]

for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.scatter(x[:, 0], x[:, 1], c=y_color)
    plt.contourf(xx0, xx1, mesh_proba[:,:,i], 20, cmap=contour_color[i],
alpha=0.5)

plt.show()
```

Iris Example (IV)



Iris Example (V)

- Build a naïve Bayes classifier for class 0 and class 1 only
- Are the parameters the same, and why?

```
x = iris.data[:100,:]
y = iris.target[:100]

gnb2 = naive_bayes.GaussianNB()
gnb2.fit(x[:,:2], y)

print(gnb2.class_prior_)
print(gnb2.class_count_)
print(gnb2.theta_)
print(gnb2.sigma_)
```

20newsgroups Dataset

- Start downloading the 20newsgroups dataset with the following commands – it takes time

```
from sklearn import datasets
data_train = datasets.fetch_20newsgroups(subset = 'train', shuffle = True, random_state = 2016, remove = ('headers', 'footers', 'quotes'))
data_test = datasets.fetch_20newsgroups(subset = 'test', shuffle = True, random_state = 2016, remove = ('headers', 'footers', 'quotes'))
```

Multinomial Naïve Bayes

$$p(\mathcal{C}_k|\mathbf{x}) \propto \boxed{p(\mathbf{x}|\mathcal{C}_k)}\, p(\mathcal{C}_k)$$

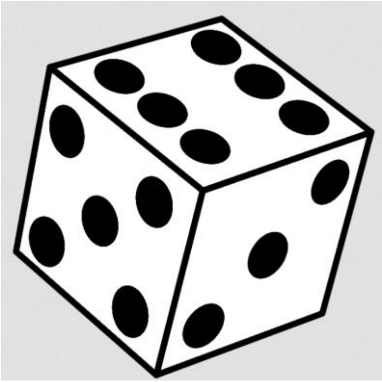
Gaussian Naïve Bayes: Gaussian distribution
Multinomial Naïve Bayes: multinomial distribution

- $p(\mathbf{x}|\mathcal{C}_k)$ follows multinomial distribution
- What is multinomial distribution?

Multinomial Distribution

- The multinomial distribution is a generalization of the binomial distribution
- Flip a coin – binomial distribution
  - P(H) = 0.6   P(T) = 0.4
  - P(H,H) =   P(H, T) =

    P(T, H) =   P(T, T) =



- Does the order matter?
- Throw a dice – multinomial distribution

Multinomial Distribution

- How does **x** look like if it follows a multinomial distribution?
- (applied, machine, learning, machine, learning)
- (applied:1, machine:2, learning:2)

Dimensions of the multinomial distribution

- **x** = (1, 2, 2)

$$\boxed{p_j = p(w_j|C_k)}$$

$$p(\mathbf{x}|C_k) = \frac{(x_1 + x_2 + \cdots + x_d)!}{x_1! x_2! \cdots x_d!} p_1^{x_1} p_2^{x_2} \cdots p_d^{x_d}$$

Parameter Estimation

$$p(\mathcal{C}_k|\mathbf{x}) \propto p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)$$

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{(x_1 + x_2 + \cdots + x_d)!}{x_1! x_2! \cdots x_d!} p_1^{x_1} p_2^{x_2} \cdots p_d^{x_d}$$

$$P(\mathcal{C}_k|\mathbf{x}) \propto p_1^{x_1} p_2^{x_2} \cdots p_d^{x_d} P(\mathcal{C}_k)$$

- How to compute $p_j = p(w_j|\mathcal{C}_k)$?

Parameter Estimation

- Learning by Maximum Likelihood Estimate
  - Simply count the frequencies in the data

$$P(w_j|\mathcal{C}_k) = \frac{count(w_j, \mathcal{C}_k)}{\sum_{w \in \mathcal{V}} count(w_j, \mathcal{C}_k)}$$

  - Create a mega-document for class k by concatenating all the docs in this class
  - Compute frequency of w in the mega-document

Problem with Maximum Likelihood

- What if there is a new word (e.g., any novel words created in internet) in a test document which never appears in the training data

$$\forall \mathcal{C}_k, \qquad P(\text{“}newword\text{”}|\mathcal{C}_k) = 0$$

$$p(\mathbf{x}|\mathcal{C}_k) = \prod_{j=1}^{d} p(x_j|\mathcal{C}_k) \propto \prod_{j=1}^{d} [p(w_j|\mathcal{C}_k)]^{x_j} = 0$$

Smoothing to Avoid Overfitting

- Smoothing to avoid Zero Probability

$$P(w_j|\mathcal{C}_k) = \frac{count(w_j, \mathcal{C}_k) + 1}{\sum_{w \in \mathcal{V}}(count(w_j, \mathcal{C}_k) + 1)}$$
$$= \frac{count(w_j, \mathcal{C}_k) + 1}{|\mathcal{V}| + \sum_{w \in \mathcal{V}} count(w, \mathcal{C}_k)}$$

20 newsgroups Example (I)

```
categories = data_train.target_names
target_map = {}
for i in range(len(categories)):
    if 'politics' in categories[i]:
        target_map[i] = 1
    else:
        target_map[i] = 0

y_train = [target_map[t] for t in data_train.target]
y_test = [target_map[t] for t in data_test.target]
print('num-of-class-0-instances in training data:', y_train.count(0)))
print('num-of-class-1-instances in training data:', y_train.count(1))
print('num-of-class-0-instances in test data:', y_test.count(0))
print('num-of-class-1-instances in test data:', y_test.count(1))
```

20 newsgroups Example (II)

```
from sklearn import feature_extraction
count_vectorizer = feature_extraction.text.CountVectorizer(min_df
= 0.01, max_df = 0.5, stop_words = 'english')
x_train = count_vectorizer.fit_transform(data_train.data)
x_test = count_vectorizer.transform(data_test.data)
feature_names = count_vectorizer.get_feature_names()
print(feature_names)
print(len(feature_names))
```

- What are stop words?
- Why do we set minimum and maximum document frequency (min_df, max_df)?
- analyzer = {‘word’, ‘char’, ‘char_wb’}

20 newsgroups Example (III)

```
from sklearn import naive_bayes, metrics
mnb = naive_bayes.MultinomialNB(alpha = 0.01)
mnb.fit(x_train, y_train)
y_pred = mnb.predict(x_test)
print('accuracy of Multinomial Naive Bayes: ',
metrics.accuracy_score(y_test, y_pred))
```

- alpha: smoothing parameter
- accuracy of Multinomial Naïve Bayes = ?

20 newsgroups Example (IV)

```
import operator
diff = mnb.feature_log_prob_[1,:] - mnb.feature_log_prob_[0,:]
name_diff = {}
for i in range(len(feature_names)):
    name_diff[feature_names[i]] = diff[i]
names_diff_sorted = sorted(name_diff.items(), key = operator.ite
mgetter(1), reverse = True)
for i in range(20):
    print(names_diff_sorted[i])
```

- feature_log_prob_[k][j]: the log probability on each feature j given class k
- What are the most distinctive features?

Logistic Regression

Logistic Regression

- Linear discriminatory model
  - Directly model the linear decision boundary

$$\ln \frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})} = \mathbf{w}^\top \mathbf{x} + b \rightarrow \mathbf{w}^\top \mathbf{x}$$

  - **w** is the parameter to be decided

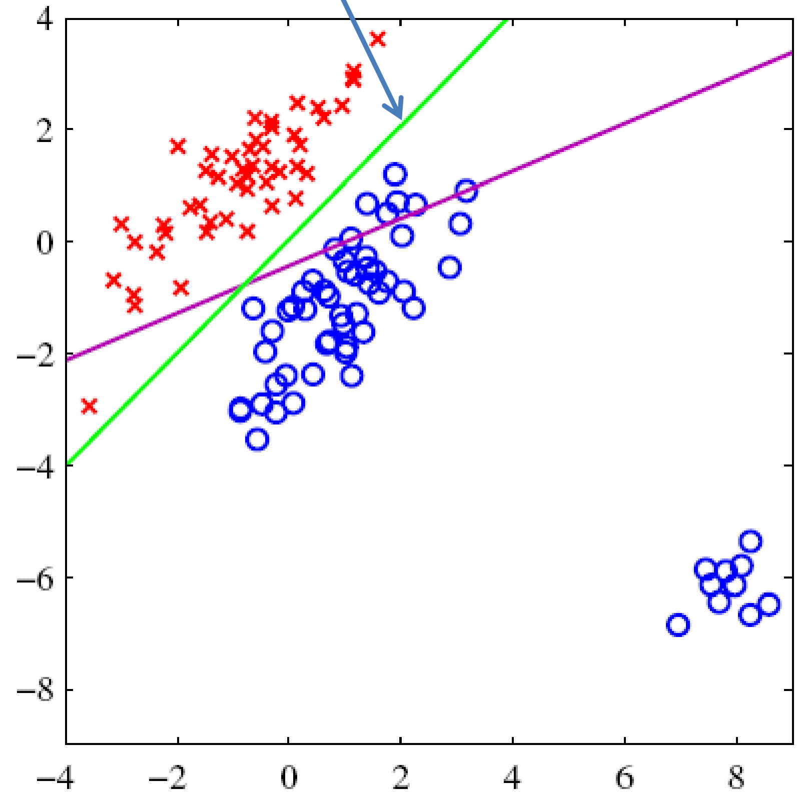
Logistic Regression

$$\ln \frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})} = \mathbf{w}^\top \mathbf{x}$$

$$\begin{aligned} p(y|\mathbf{x}) &= \frac{1}{\exp(-y\mathbf{w}^\top \mathbf{x}) + 1} \\ &= \sigma(y\mathbf{w}^\top \mathbf{x}) \end{aligned}$$

logistic / sigmoid function

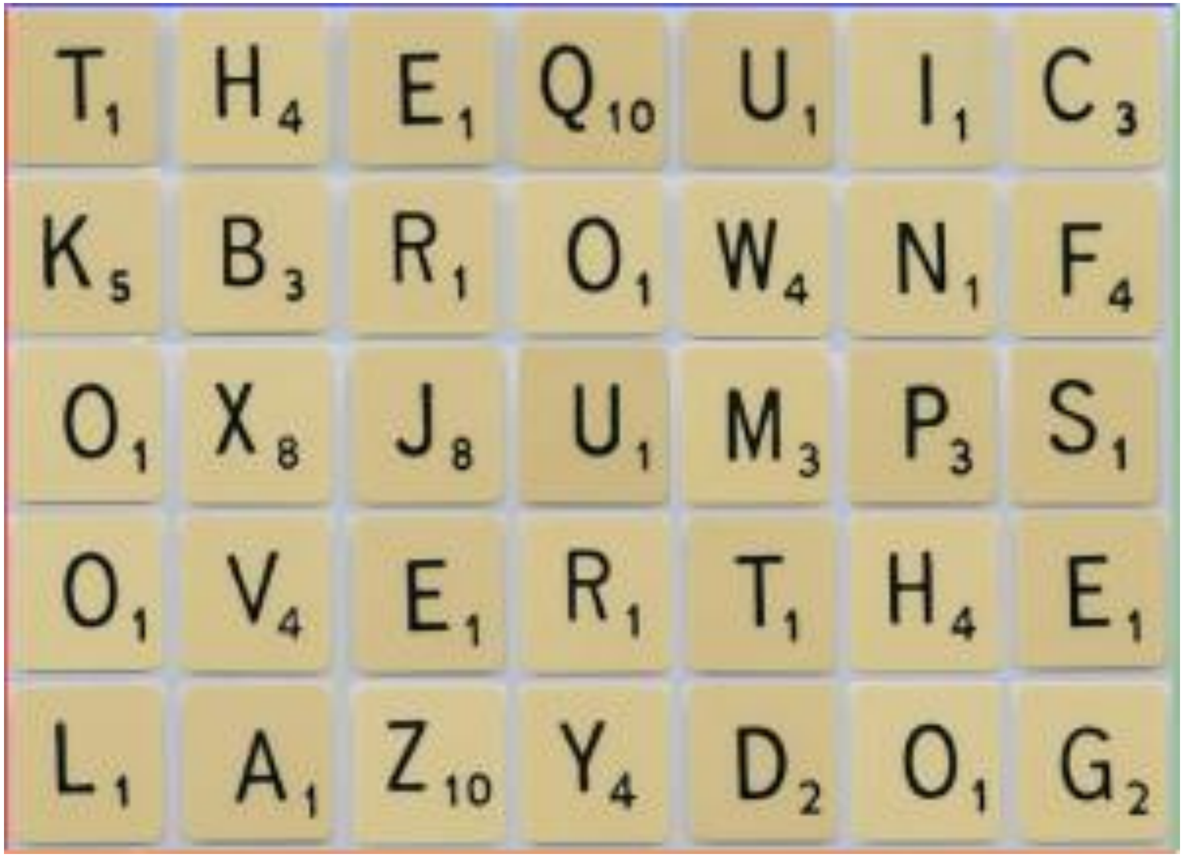
$$\mathbf{w}^\top \mathbf{x} + b = 0$$



20 newsgroups Example

```
from sklearn import feature_extraction, linear_model
count_vectorizer = feature_extraction.text.CountVectorizer(min_df = 0.01,
 max_df = 0.5, stop_words = 'english')
tfidf_vectorizer = feature_extraction.text.TfidfVectorizer(min_df = 0.01,
 max_df = 0.5, sublinear_tf = True, stop_words = 'english')
x_count_train = count_vectorizer.fit_transform(data_train.data)
x_tfidf_train = tfidf_vectorizer.fit_transform(data_train.data)
x_count_test = count_vectorizer.transform(data_test.data)
x_tfidf_test = tfidf_vectorizer.transform(data_test.data)
logit_count = linear_model.LogisticRegression()
logit_tfidf = linear_model.LogisticRegression()
logit_count.fit(x_count_train, y_train)
logit_tfidf.fit(x_tfidf_train, y_train)
logit_count_pred = logit_count.predict(x_count_test)
logit_tfidf_pred = logit_tfidf.predict(x_tfidf_test)
print('accuracy of Logistic Regression on count: ', metrics.accuracy_scor
e(y_test, logit_count_pred))
print('accuracy of Logistic Regression on tfidf: ', metrics.accuracy_scor
e(y_test, logit_tfidf_pred))
```

tf-idf: term frequency–inverse document frequency



tf-idf

- tf: word *w* and document *d*

$$tf(w, d) = \#\text{-of-times } w \text{ appears in } d$$

- idf: a set of documents *D*

$$idf(w, D) = log \frac{|D|}{|\{d \in D : w \text{ appears in } d\}|}$$

20 newsgroups Example

```
from sklearn import feature_extraction, linear_model
count_vectorizer = feature_extraction.text.CountVectorizer(min_df = 0.01,
 max_df = 0.5, stop_words = 'english')
tfidf_vectorizer = feature_extraction.text.TfidfVectorizer(min_df = 0.01,
 max_df = 0.5, sublinear_tf = True, stop_words = 'english')
x_count_train = count_vectorizer.fit_transform(data_train.data)
x_tfidf_train = tfidf_vectorizer.fit_transform(data_train.data)
x_count_test = count_vectorizer.transform(data_test.data)
x_tfidf_test = tfidf_vectorizer.transform(data_test.data)
logit_count = linear_model.LogisticRegression()
logit_tfidf = linear_model.LogisticRegression()
logit_count.fit(x_count_train, y_train)
logit_tfidf.fit(x_tfidf_train, y_train)
logit_count_pred = logit_count.predict(x_count_test)
logit_tfidf_pred = logit_tfidf.predict(x_tfidf_test)
print('accuracy of Logistic Regression on count: ', metrics.accuracy_scor
e(y_test, logit_count_pred))
print('accuracy of Logistic Regression on tfidf: ', metrics.accuracy_scor
e(y_test, logit_tfidf_pred))
```

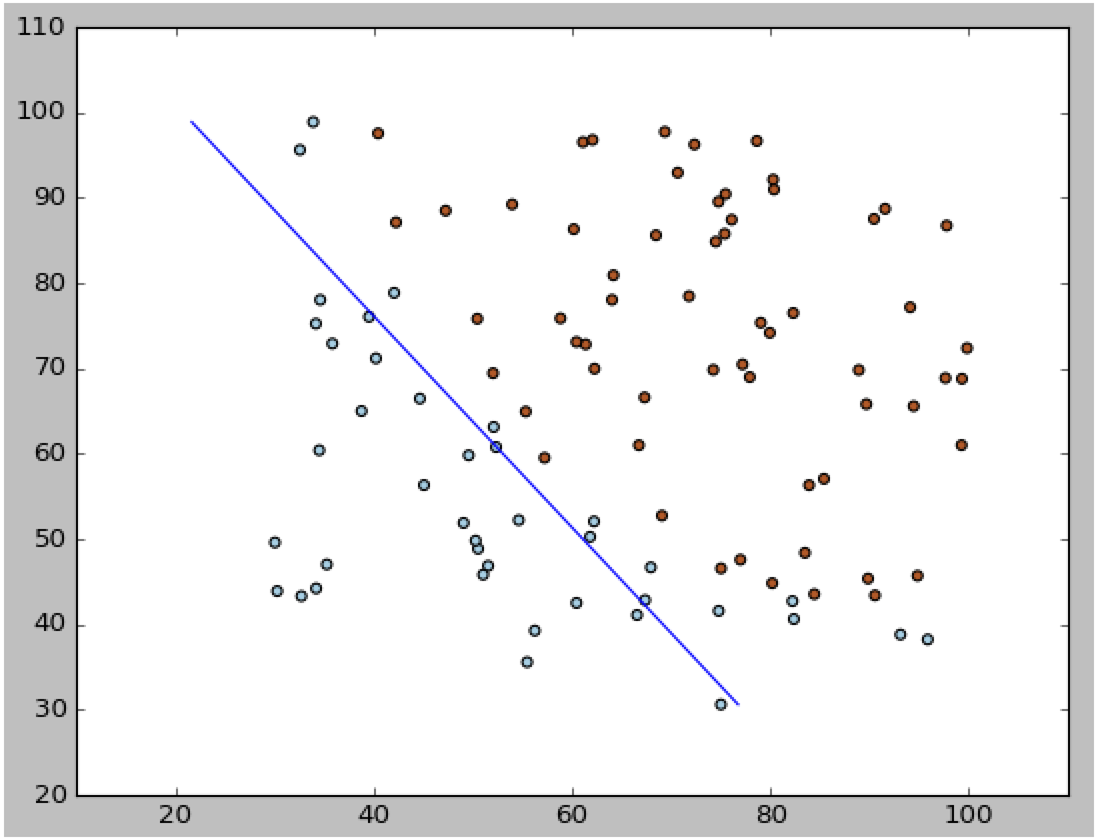
Example – Admission (I)

- Each row of the dataset consists of two exam scores (column 1 and 2), and a Boolean variable on the 3rd column, with 1 for the student's been admitted, and 0 for the student's been rejected
- Build a logistic regression model for the discriminant boundary of admitted class and rejected class

Example – Admission (II)

```
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
data = np.loadtxt('exam_score.txt', delimiter = ',')
logit = linear_model.LogisticRegression()
logit.fit(data[:,:2], data[:,2])
plt.scatter(data[:,0], data[:,1], c=data[:,2], cmap=plt.cm.Paired)
y = np.linspace(min(data[:,1]), max(data[:,1]), 200)
x = (- logit.coef_[0][1] * y - logit.intercept_[0]) / logit.coef_[0][0]
plt.plot(x, y)
plt.show()
```

Example – Admission (III)



Regularization

- Regularization: to prevent overfitting
- Instead of minimizing the original cost function $\mathrm{f}(X, Y)$, minimize $\mathrm{C} * \mathrm{f}(X, Y) + ||w||$, where $||w||$ represents the complexity of the model
  - Large C: trade-off model complexity for better classification accuracy
  - Small C: trade-off classification accuracy for less model complexity
- In linear models, $||w||$ is often the L_1-norm or L_2-norm of the model parameter w

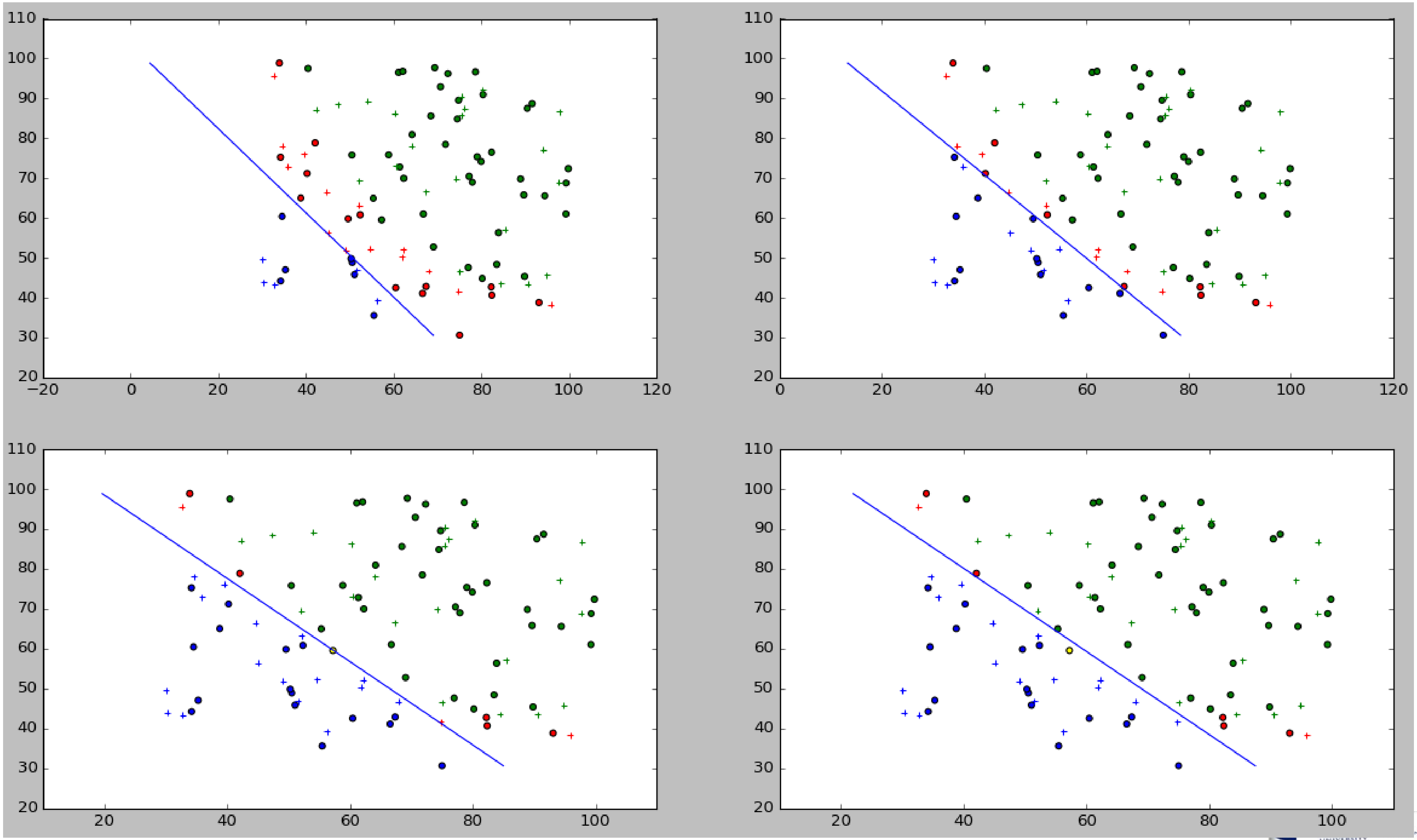
Example – Admission (IV)

```
import numpy as np
from sklearn import linear_model, metrics
import matplotlib.pyplot as plt
data = np.loadtxt('exam_score.txt', delimiter = ',')

np.random.seed(2016)
train = np.random.choice([True, False], len(data), replace=
True, p=[0.5, 0.5])
x_train = data[train,:2]
y_train = data[train,2]
x_test = data[~train,:2]
y_test = data[~train,2]
plt.figure(figsize = (18,18))
C = [1, 2, 5, 10]
```

Example – Admission (IV)

```
for i in range(4):
    plt.subplot(2, 2, i+1)
    logit = linear_model.LogisticRegression(C=C[i])
    logit.fit(x_train, y_train)
    y_train_pred = logit.predict(x_train)
    y_test_pred = logit.predict(x_test)
    color = [['blue', 'red'], ['yellow', 'green']]
    print('accuracy for training data at C=%f is %f' % (C[i], metrics.accur
acy_score(y_train, y_train_pred)))
    print('accuracy for test data at C=%f is %f' % (C[i], metrics.accuracy_
score(y_test, y_test_pred)))
    print('model complexity:', np.square(logit.coef_[0][0]) + np.square(log
it.coef_[0][1]) + np.square(logit.intercept_[0]))
    plt.scatter(data[train,0], data[train,1], c=[color[int(j1)][int(j2)] fo
r j1, j2 in zip(y_train, y_train_pred)], marker='o')
    plt.scatter(data[~train,0], data[~train,1], c=[color[int(j1)][int(j2)]
for j1, j2 in zip(y_test, y_test_pred)], marker='+')
    y = np.linspace(min(data[:,1]), max(data[:,1]), 200)
    x = (- logit.coef_[0][1] * y - logit.intercept_[0]) / logit.coef_[0][0]
    plt.plot(x, y)
plt.show()
```

Example – Admission (V)



Example – Admission (VI)

C	Accuracy on Training	Accuracy on Test	Model Complexity
C=1	0.767	0.650	7.218
C=2	0.867	0.750	17.635
C=5	0.900	0.925	45.190
C=10	0.900	0.950	79.071

Generative vs. Discriminative (I)

- Build a naïve Bayes classifier for class 0 and class 1 only
- Are the parameters the same, and why?

```
x = iris.data[:100,:]
y = iris.target[:100]

gnb2 = naive_bayes.GaussianNB()
gnb2.fit(x[:,:2], y)

print(gnb2.class_prior_)
print(gnb2.class_count_)
print(gnb2.theta_)
print(gnb2.sigma_)
```

Generative vs. Discriminative (II)

- Build a logistic regression classifier for class 0 and class 1 only
- Are the parameters the same, and why?

```
import numpy as np
from sklearn import datasets, linear_model
import matplotlib.pyplot as plt

iris = datasets.load_iris()
x = iris.data
y = iris.target
x2 = iris.data[:100,:]
y2 = iris.target[:100]

logit = linear_model.LogisticRegression()
logit.fit(x[:,:2], y)
logit2 = linear_model.LogisticRegression()
logit2.fit(x2[:,:2], y2)
```

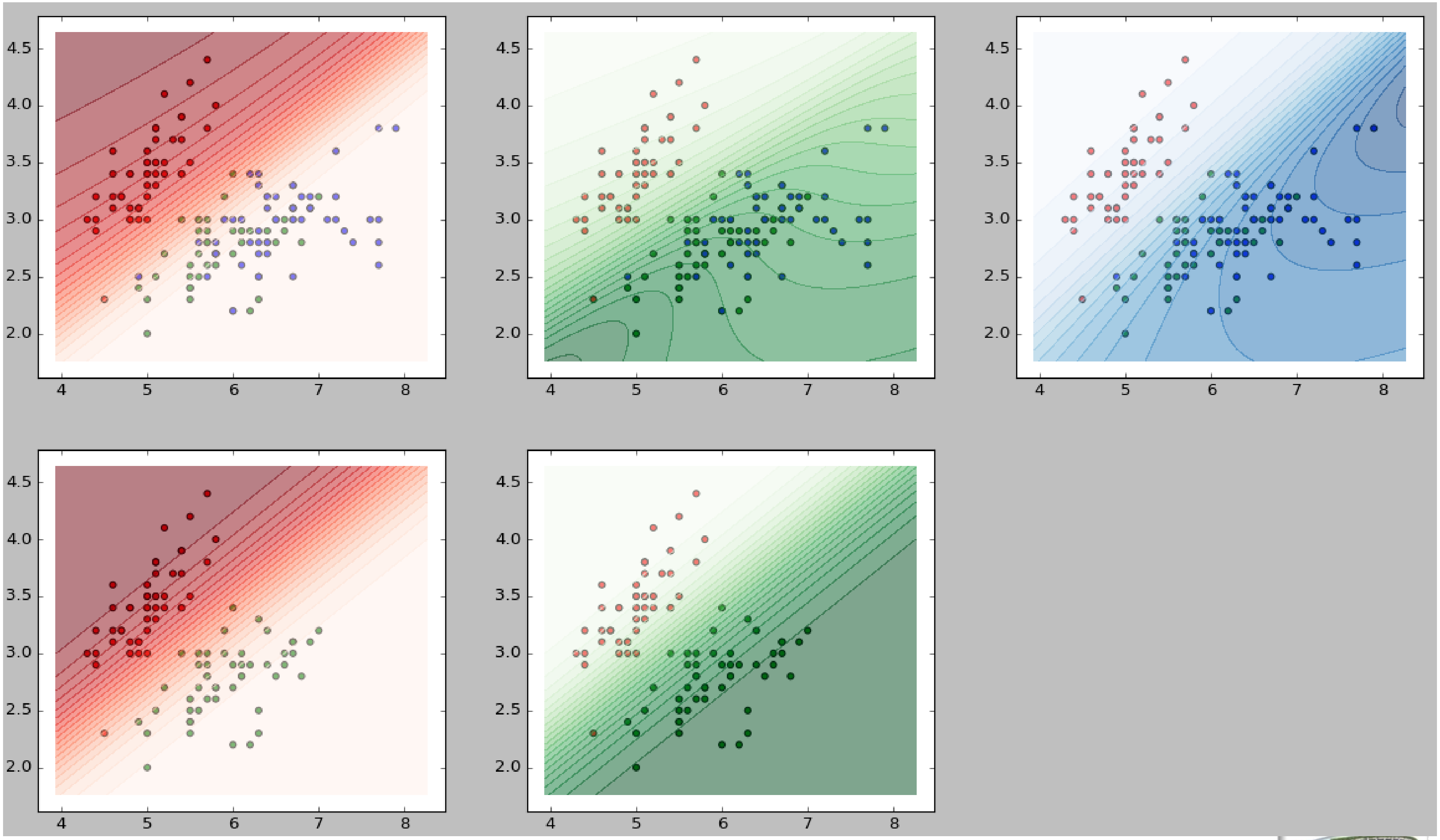
Generative vs. Discriminative (III)

```
x0 = my_linspace(min(x[:,0]), max(x[:,0]), steps)
x1 = my_linspace(min(x[:,1]), max(x[:,1]), steps)
xx0, xx1 = np.meshgrid(x0, x1)
mesh_data = np.c_[xx0.ravel(), xx1.ravel()]
mesh_proba = logit.predict_proba(mesh_data).reshape(steps, steps, 3)
mesh_proba2 = logit2.predict_proba(mesh_data).reshape(steps, steps, 2)

color = ['red', 'green', 'blue']
y_color = [color[i] for i in y]
y2_color = [color[i] for i in y2]
```

Generative vs. Discriminative (IV)

```
plt.figure(figsize = (18, 10))
contour_color = [plt.cm.Reds, plt.cm.Greens, plt.cm.Blues]

for i in range(3):
    plt.subplot(2, 3, i+1)
    plt.scatter(x[:,0], x[:,1], c=y_color)
    plt.contourf(xx0, xx1, mesh_proba[:,:,i], 20, cmap=contour_color[i],
 alpha=0.5)

for i in range(2):
    plt.subplot(2, 3, i+4)
    plt.scatter(x2[:,0], x2[:,1], c=y2_color)
    plt.contourf(xx0, xx1, mesh_proba2[:,:,i], 20, cmap=contour_color[i]
, alpha=0.5)

plt.show()
```

Generative vs. Discriminative (V)



Framework of Classifiers

- Classifier Declaration
  - clf = some_classifier()
- Train classifier
  - clf.fit(x_train, y_train)
- Test on datasets
  - clf.score(x_test, y_test)
- Predict new data
  - clf.predict(x_new)
- Classifier Visualization