MITB ISSS610
Applied Machine Learning

Instructors:

Dr. Steven HOI

Dr. DAI Bing Tian

School of Information Systems

Singapore Management University

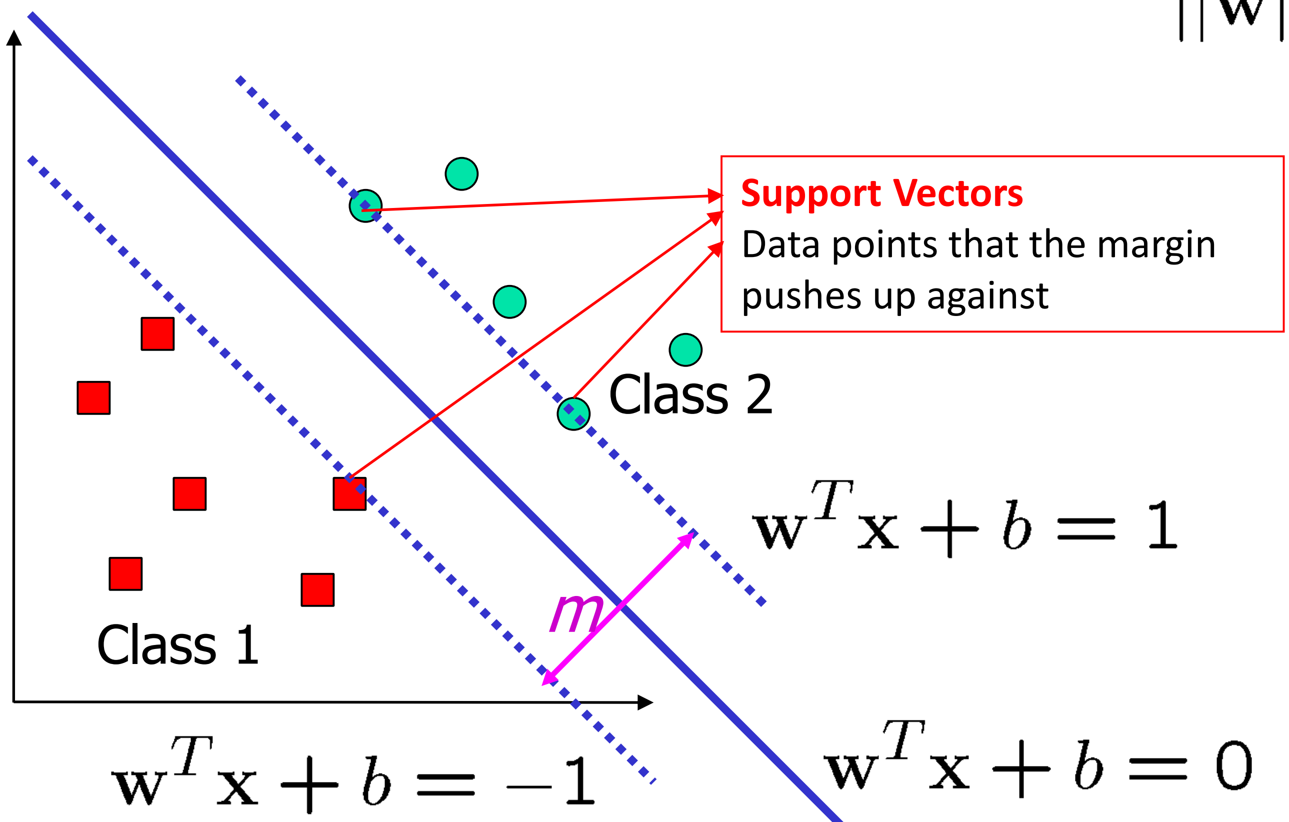
Outline

- SVM
  - Binary
  - Multi-class
- Ensemble Learning
  - Bagging
  - AdaBoost
- Cross Validation
- Grid Search

Support Vector Machines

Support Vector Machines

- The decision boundary should be as far away from the data of both classes as possible
  - We should maximize the margin: $m = \frac{2}{||\mathbf{w}||}$



SVM – Linear Kernel

- Two implementations in *scikit-learn*
  - svm.LinearSVC
  - svm.SVC (kernel='linear')

```
iris = datasets.load_iris()
x = iris.data[:100,:2]
y = iris.target[:100]
clf = []
for C in [1, 10, 100, 1000]:
    clf.append(svm.LinearSVC(C=C))

for C in [1, 10, 100, 1000]:
    clf.append(svm.SVC(kernel='linear', C=C))

for i in range(8):
    clf[i].fit(x, y)
```

SVM – Linear Kernel

- Plotting the decision boundary
  - Class Divider: $\mathbf{w}^T\mathbf{x} + b = 0$
  - Margin at positive class: $\mathbf{w}^T\mathbf{x} + b = 1$
  - Margin at negative class: $\mathbf{w}^T\mathbf{x} + b = -1$

```
   div = my_line(x0[0], x0[-1], x1[0], x1[-1], clf[i].coef_[0],
clf[i].intercept_[0])
   pos = my_line(x0[0], x0[-1], x1[0], x1[-1], clf[i].coef_[0],
clf[i].intercept_[0] - 1)
   neg = my_line(x0[0], x0[-1], x1[0], x1[-1], clf[i].coef_[0],
clf[i].intercept_[0] + 1)
   plt.plot(div[0], div[1], color='black')
   plt.plot(pos[0], pos[1], color='blue', ls='--')
   plt.plot(neg[0], neg[1], color='red', ls='--')
```

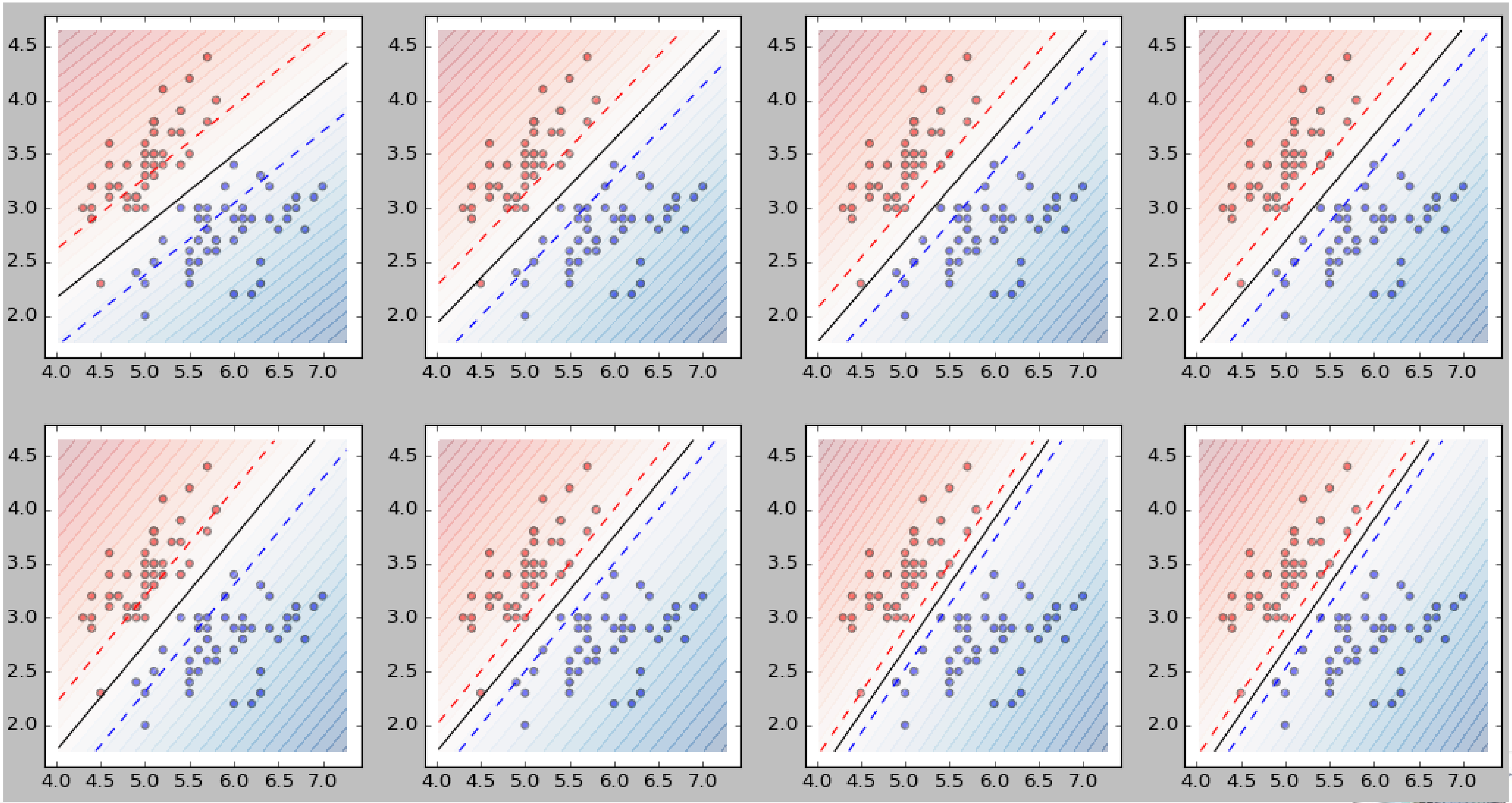
SVM – Linear Kernel

- Plot $\mathbf{w}^T\mathbf{x} + b = 0$
  – $w_0 = 0$ ➔ horizontal line; w_1=0 ➔ vertical line
  – General case: $y = -\frac{w_0 \cdot x + b}{w_1}$

```
def my_line (x_min, x_max, y_min, y_max, w, b):
    if w[0] == 0.0:
        if w[1] == 0.0:
            print('impossible line')
            return [0.0, 0.0], [0.0, 0.0]
        else:
            return [x_min, x_max], [- b / w[1], - b / w[1]]
    elif w[1] == 0.0:
            return [- b / w[0], - b / w[0]], [y_min, y_max]
    else:
        xn = - (w[1] * np.asarray([y_min, y_max]) + b)  / w[0]
        x_min, x_max = max(x_min, min(xn)), min(x_max, max(xn))
        return [x_min, x_max], [- (w[0] * x_min + b) / w[1], -
(w[0] * x_max + b) / w[1]]
```

SVM – Linear Kernel

C = 1 | C = 10 | C = 100 | C = 1000



SVM – Polynomial Kernel

- Polynomial kernel

$$\phi(\mathbf{x}) \;=\; (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, x_1^2,\; x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3)$$

```python
def map(orig_data, terms):
    mapped = []
    for x in orig_data:
        xx = []
        for d in terms:
            v = 1.0
            for pos, exponent in d.items():
                v *= math.pow(x[pos], exponent)
            xx.append(v)
        mapped.append(xx)
    return np.asarray(mapped)
```

Kernel Tricks

- **Idea:** Replacing dot product with a kernel $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j)$$

- **Kernel Functions:**
  - Linear Kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle = \mathbf{x}_i^\top \mathbf{x}_j$
  - Polynomial Kernel (degree d)

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + 1)^d$$

  - Gaussian kernel / RBF Kernel

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\Big(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\Big) \quad \kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\Big(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2\Big)$$

SVM – Non-linear Kernel

- Polynomial kernel
- RBF kernel

```
clf = []
for kern in ['linear', 'poly', 'rbf']:
    clf.append(svm.SVC(kernel=kern, gamma='auto'))

for kern in ['linear', 'poly', 'rbf']:
    clf.append(svm.SVC(kernel=kern, gamma=1))

for i in range(6):
    clf[i].fit(x, y)
```

Visualization of Linear Kernel

- Class boundary visualization with linear kernel

```
    div = my_line(x0[0], x0[-1], x1[0], x1[-1], clf[i].coef_[0],
clf[i].intercept_[0])
    pos = my_line(x0[0], x0[-1], x1[0], x1[-1], clf[i].coef_[0],
clf[i].intercept_[0] - 1)
    neg = my_line(x0[0], x0[-1], x1[0], x1[-1], clf[i].coef_[0],
clf[i].intercept_[0] + 1)
    plt.plot(div[0], div[1], color='black')
    plt.plot(pos[0], pos[1], color='blue', ls='--')
    plt.plot(neg[0], neg[1], color='red', ls='--')
```

Visualization of Non-linear Kernel

- Class boundary with non-linear kernel?

coef_ : array, shape = [n_class-1, n_features]

Weights assigned to the features (coefficients in the primal problem). This is only available in the case of a linear kernel.

coef_ is a readonly property derived from dual_coef_ and support_vectors_.

intercept_ : array, shape = [n_class * (n_class-1) / 2]

Constants in decision function.

decision function!

SVM – Decision Function

- What is the decision function?
  - Linear case:

$$\mathbf{w}^T\mathbf{x} + b$$

  - Non-linear case:

$$\phi(\mathbf{x})$$

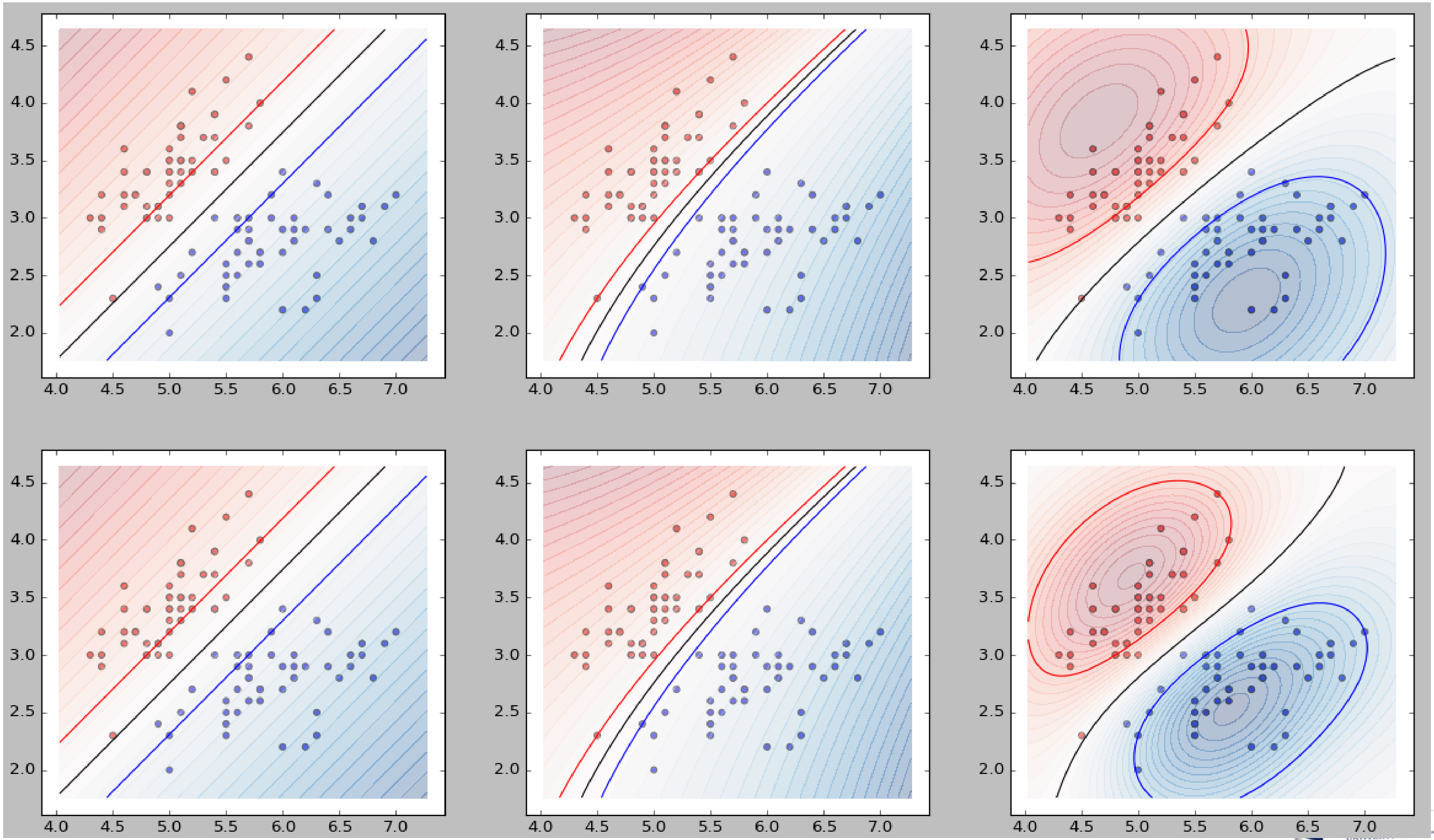
SVM – Non-linear Kernel

- How to visualize the class divider?

```
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_deci = [0] * 6
for i in range(6):
    mesh_deci[i] = clf[i].decision_function(mesh_data).reshape(steps, steps)
```

```
for i in range(6):
    plt.subplot(2, 3, i+1)
    plt.scatter(x[:,0], x[:,1], c=y_color)
    plt.contourf(xx0, xx1, np.maximum(-mesh_deci[i], 0.0), 20,
 cmap=contour_color[0], alpha=0.3)
    plt.contourf(xx0, xx1, np.maximum(mesh_deci[i], 0.0), 20,
cmap=contour_color[1], alpha=0.3)
    plt.contour(xx0, xx1, mesh_deci[i], levels=[0.0,-1.0,1.0],
 colors=['black','red','blue'])
```

SVM – Non-linear Kernel



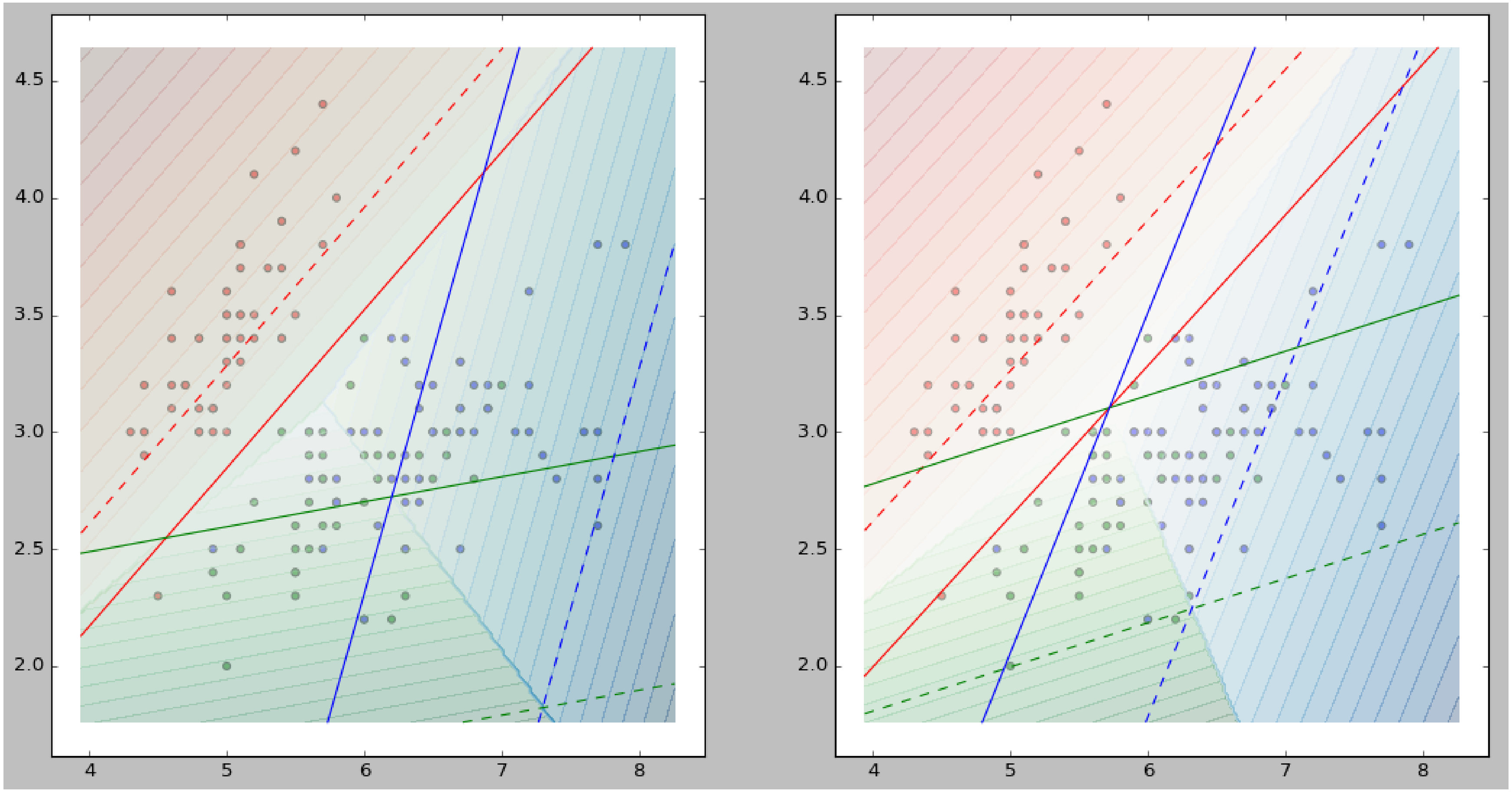
Multi-Class SVM

Linear SVM – Multi-class

- liblinear implementation:
  - ovr – one v.s. rest
  - crammer_singer
    - Constraint: sum of the decision = 0

```
clf = [svm.LinearSVC(multi_class='ovr'), svm.LinearSVC
(multi_class='crammer_singer')]
for i in range(2):
    clf[i].fit(x, y)
```

Linear SVM – Multi-class



General SVM – Multi-class

- libsvm implements ovo – one v.s. one
  - as of now
- Default option for decision_function_shape
  - ovo
  - From sklearn 0.18 onwards: ovr

```
clf = [svm.SVC(kernel='linear', probability=True),
       svm.SVC(kernel='poly', probability=True),
       svm.SVC(probability=True)]
for i in range(3):
    clf[i].fit(x, y)
```

SVM – Multi-class

- Add one more class to Iris data

- How many ovo classifier?

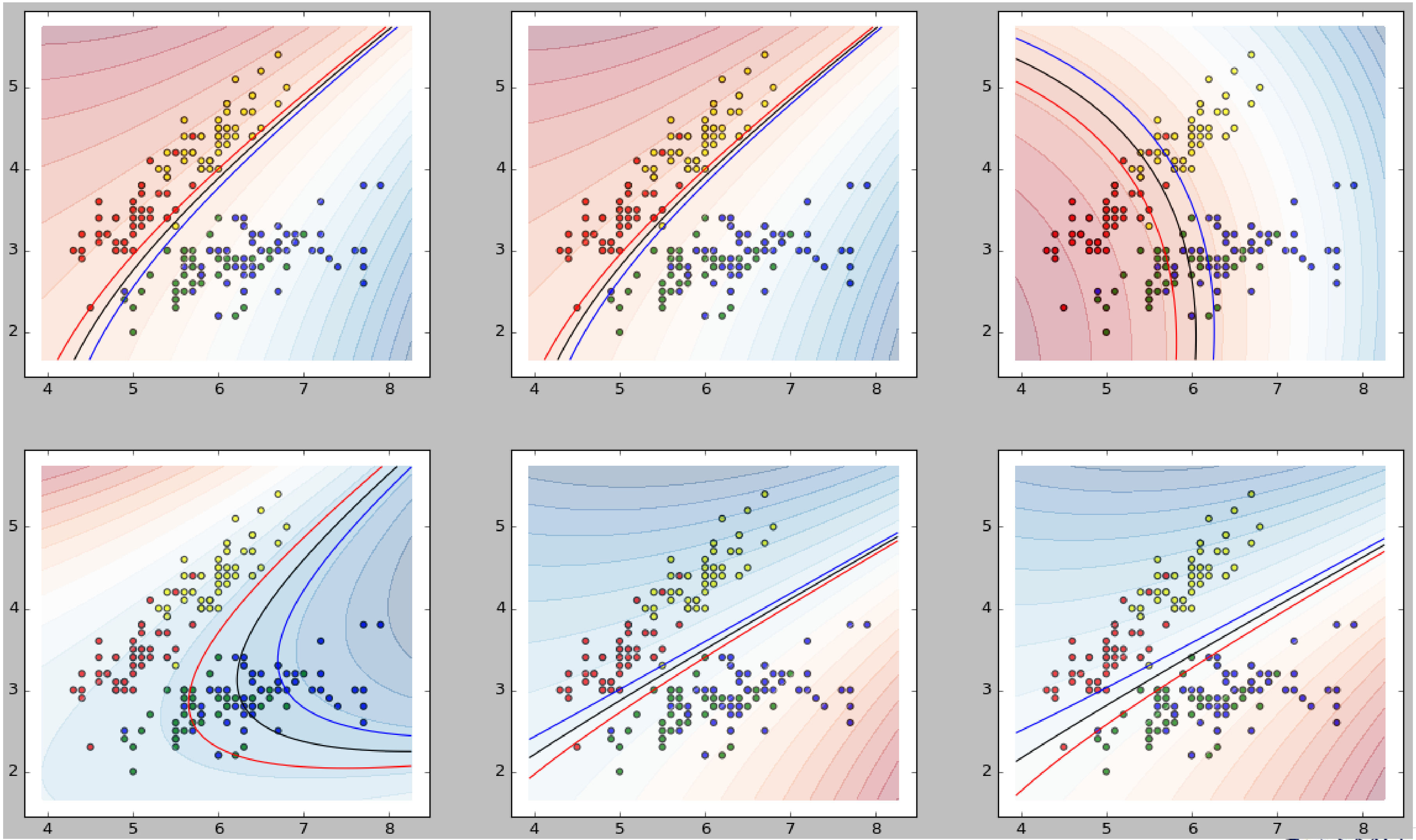
```
iris = datasets.load_iris()
x = iris.data[:,:2]
y = iris.target
x = np.append(x, iris.data[:50,:2] + [1.0, 1.0], 0)
y = np.append(y, [3] * 50, 0)

clf = [svm.SVC(kernel='linear', probability=True),
       svm.SVC(kernel='poly', probability=True),
       svm.SVC(probability=True)]
for i in range(3):
    clf[i].fit(x, y)
```

SVM – Multi-class

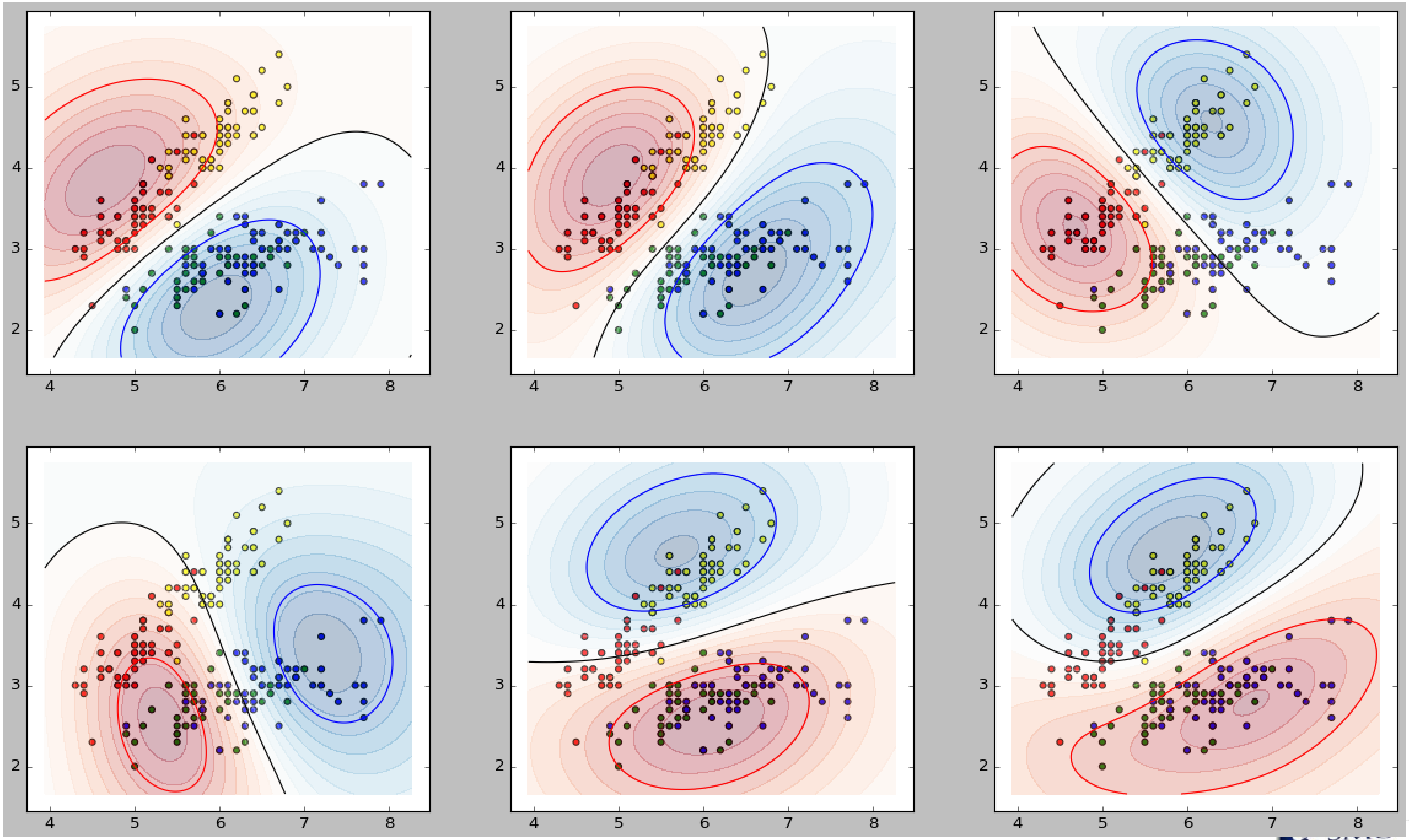
- number-of-classes = 4
- number-of-ovo-classifiers = $4 \times (4 - 1) \div 2 = 6$

```
j = 0
for k in range(n_class):
    for l in range(k+1, n_class):
        plt.subplot(2, 3, j + 1)
        plt.scatter(x[:,0], x[:,1], c=y_color)
        plt.contourf(xx0, xx1, -deci[:,j].reshape(steps, steps)
, 20, cmap=plt.cm.RdBu, alpha=0.3)
        plt.contour(xx0, xx1, deci[:,j].reshape(steps, steps),
levels=[0.0,1.0,-1.0], colors=['black','red','blue'])
        j += 1
plt.show()
```

SVM – Multi-class



SVM – Multi-class



Prediction and Decision Function in Multi-class SVM

- Prediction in multi-class SVM: vote
- sklearn/multiclass.py, line 595

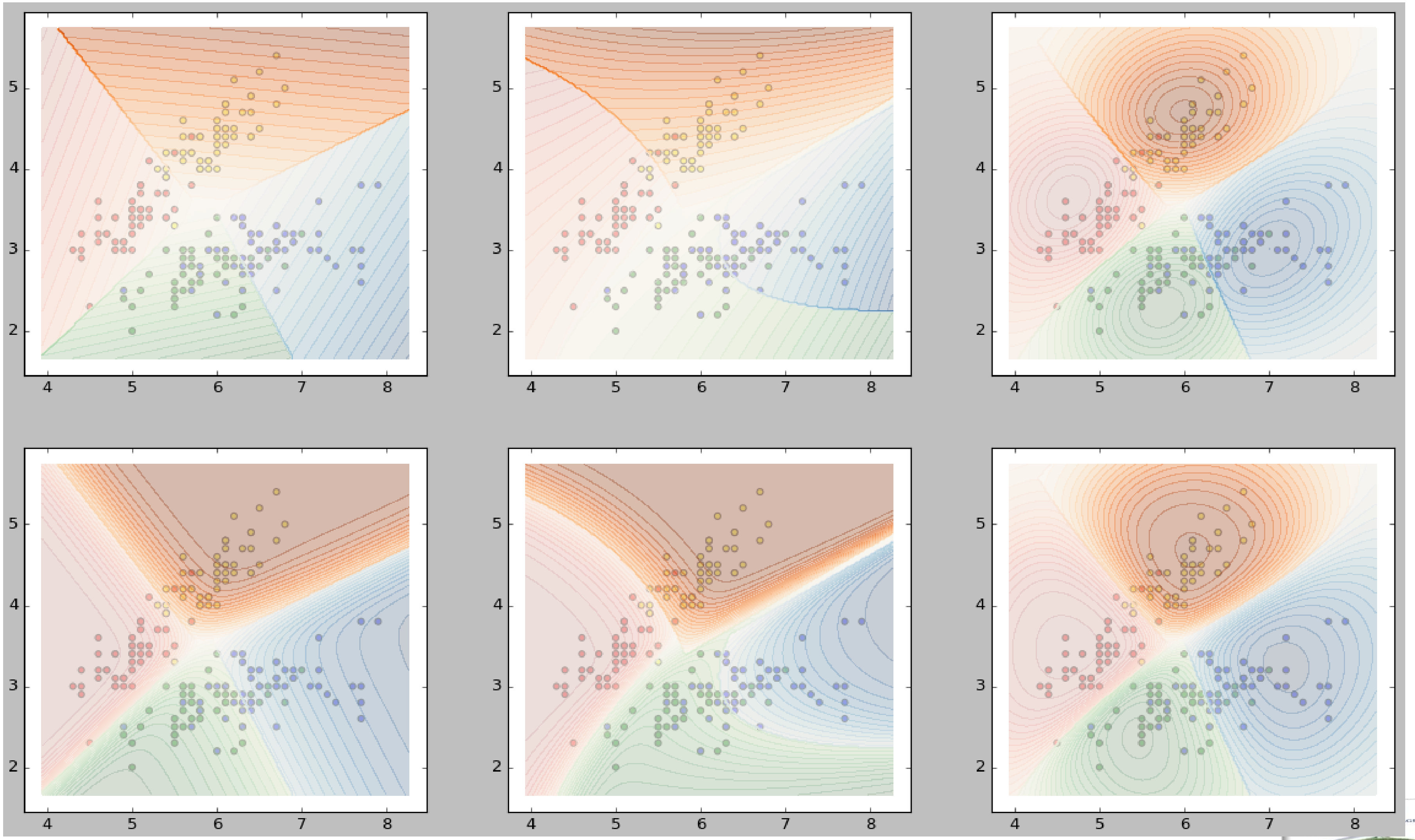
```
votes = np.zeros((n_samples, n_classes))
sum_of_confidences = np.zeros((n_samples, n_classes))

k = 0
for i in range(n_classes):
    for j in range(i + 1, n_classes):
        sum_of_confidences[:, i] -= confidences[:, k]
        sum_of_confidences[:, j] += confidences[:, k]
        votes[predictions[:, k] == 0, i] += 1
        votes[predictions[:, k] == 1, j] += 1
        k += 1
```

Prediction and Decision Function in Multi-class SVM

- Sum decision functions in ovo case
- class k v.s. class l

```
    j = 0
    for k in range(n_class):
        for l in range(k+1, n_class):
            mesh_deci[i][:,k] += deci[:,j]
            mesh_deci[i][:,l] -= deci[:,j]
            plt.subplot(2, 3, j + 1)
            plt.scatter(x[:,0], x[:,1], c=y_color)
            plt.contourf(xx0, xx1, -deci[:,j].reshape(steps, steps)
, 20, cmap=plt.cm.RdBu, alpha=0.3)
            plt.contour(xx0, xx1, deci[:,j].reshape(steps, steps),
levels=[0.0,1.0,-1.0], colors=['black','red','blue'])
            j += 1
    plt.show()
```

SVM – Multi-class



Bagging and adaBoost

Bagging Algorithm

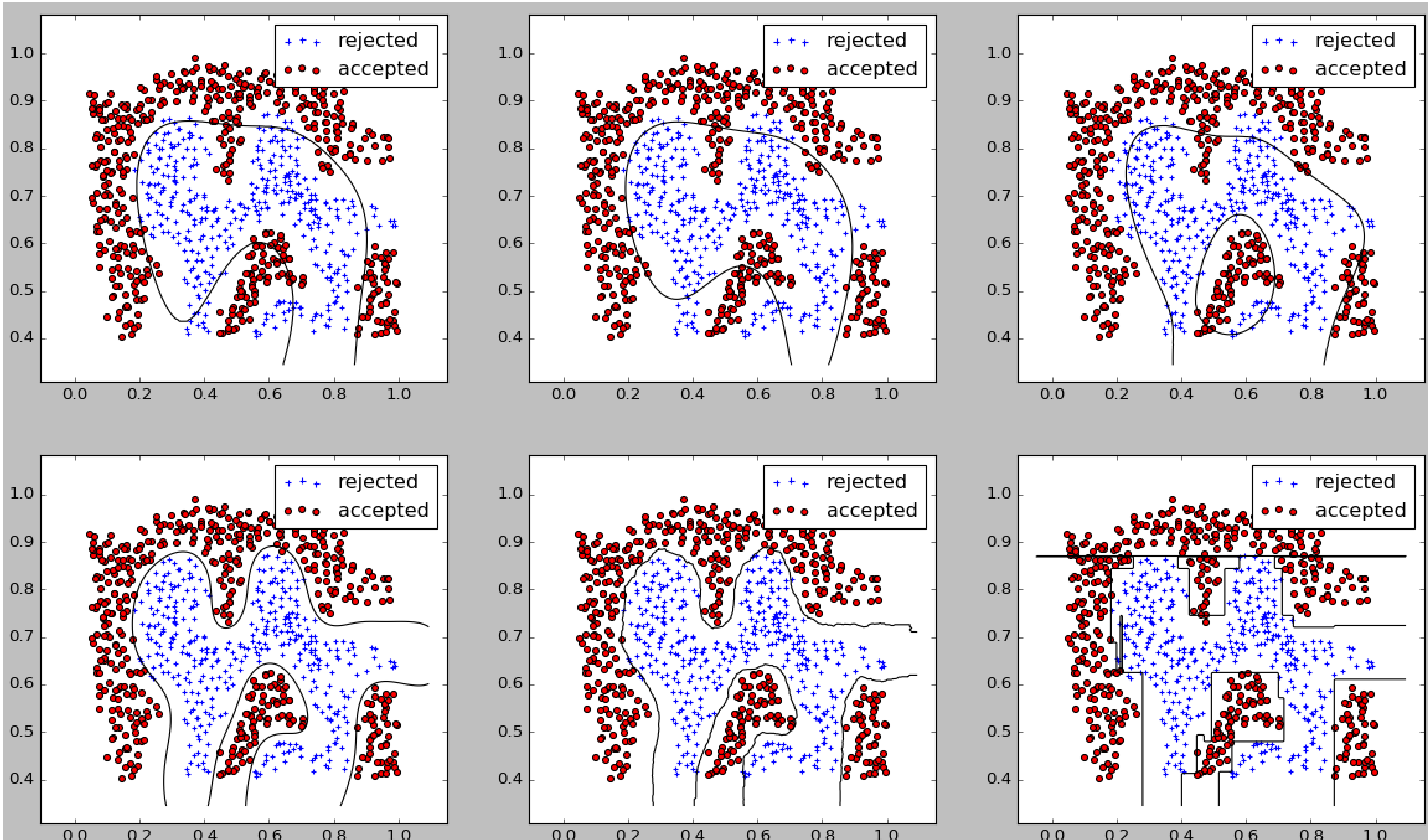
- Create k boostrap samples $D^1, D^2, \dots, D^k$
- Train distinct classifier h_i on each D^i
- Classify a new instance $\mathbf{x}$ by classifier vote with equal weights

$$c^*(\mathbf{x}) = \arg\max_c \sum_{i=1}^{k} p(c|h_i, \mathbf{x})$$

Bagging and adaBoost

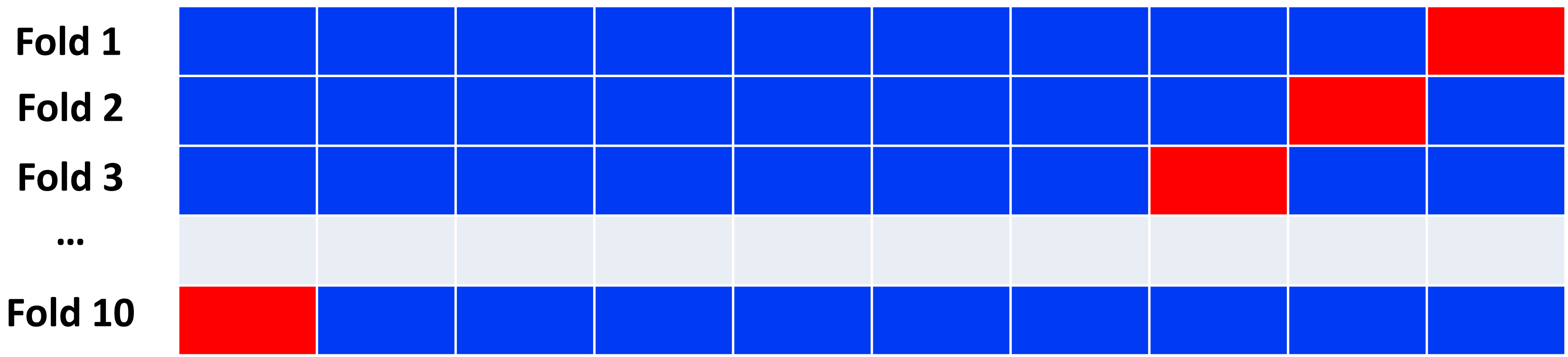
```
clf = [0] * 6
clf[0] = svm.SVC(gamma=10, probability=True)
clf[1] = ensemble.BaggingClassifier(svm.SVC(gamma=10, probability=Tr
ue), n_estimators=10, max_samples=0.5, random_state=2016)
clf[2] = ensemble.AdaBoostClassifier(svm.SVC(gamma=10, probability=T
rue), n_estimators=10, random_state=2016)
clf[3] = svm.SVC(gamma=100, probability=True)
clf[4] = ensemble.BaggingClassifier(neighbors.KNeighborsClassifier()
, n_estimators=10, max_samples=0.5, random_state=2016)
clf[5] = ensemble.AdaBoostClassifier(tree.DecisionTreeClassifier(),
n_estimators=10, random_state=2016)
for i in range(6):
    clf[i].fit(x_train, y_train)
    print(clf[i].score(x_test, y_test))
```

Bagging and adaBoost



Cross Validation

Cross Validation



- Divide the entire data to *k* portions
- Do k times
  - Train the classifier with the *k-1* portions
  - Evaluate the classifier on the remaining 1 portion
- Take the average score as the overall performance

Cross Validation

```
for i in range(6):
    score = cross_validation.cross_val_score(clf[i], x, y, cv=10)
    pred = cross_validation.cross_val_predict(clf[i], x, y, cv=10)
    print(score)
    print('    mean:', np.mean(score), 'standard deviation:', np.std(score))
    print('    mean:', len(y[pred==y]) / len(y))
```

- KFold and StratifiedKFold
- StratifiedKFold:
  - Maintain class ratio in each portion
  - Prevents classes come in sequence
- Any more problem?

Cross Validation

```
for i in range(6):
    cv = cross_validation.ShuffleSplit(len(x), n_iter=10, test_size=0.1, ran
dom_state=2016)
    score = cross_validation.cross_val_score(clf[i], x, y, cv=cv)
    print(score)
    print('    mean:', np.mean(score), 'standard deviation:', np.std(score))
```

Grid Search

Grid Search

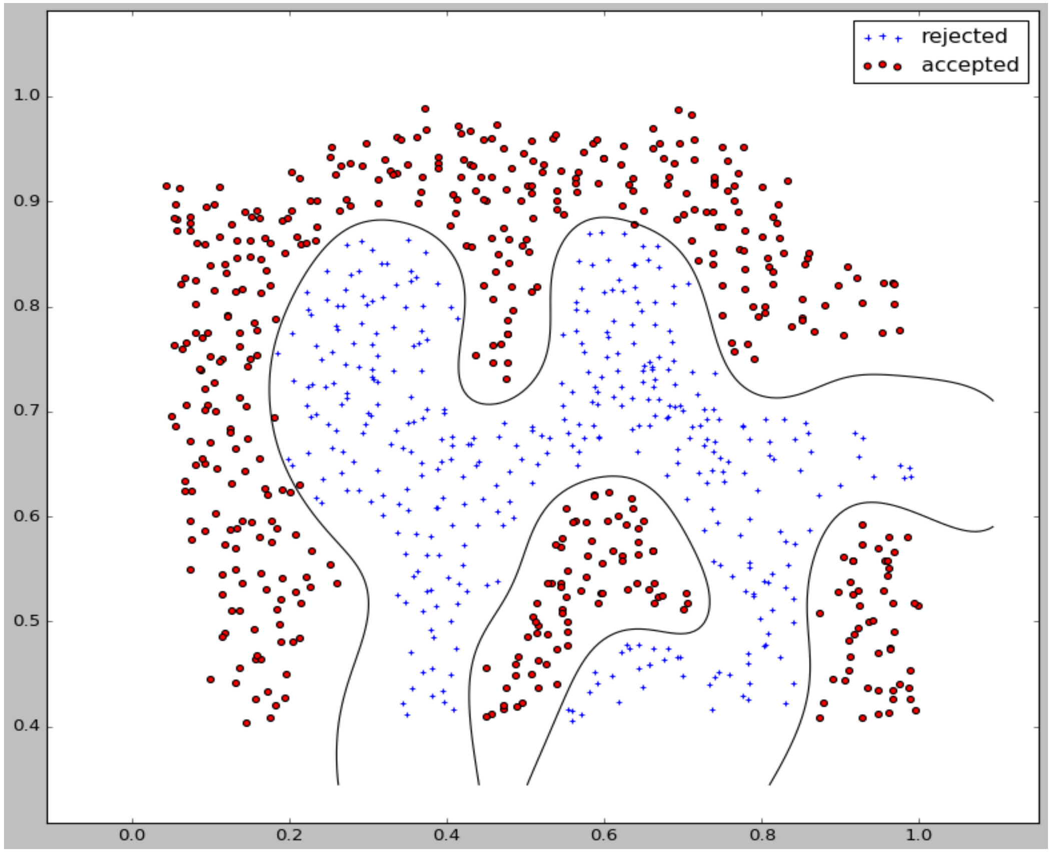
- Feed a set of parameter combinations to grid search
- Grid search evaluates based on cross validation

```
parameters = [
    {'kernel':['poly'],'degree':[2,3],'C':[1,10,100,1000]},
    {'kernel':['rbf'],'gamma':[1,10,100,1000,'auto'],'C':[1, 10, 100,
 1000]}]
clf = grid_search.GridSearchCV(svm.SVC(), parameters, cv=10)
clf.fit(x, y)
print(clf.best_score_)
print(clf.best_params_)
```

Grid Search

- best score
- best parameters:

```
0.929316338355
{'gamma': 100, 'kernel': 'rbf', 'C': 10}
```



Take-aways

- SVM
  - biclassification v.s. multiclassification
  - linear kernel, polynomial kernel, rbf kernel
  - decision function and visualization
- Combining classifiers
  - bagging and adaBoost
- Parameter selection
  - cross validation
  - grid search

Answer to Assignment 1

Question 1

```
import time
import numpy as np
from sklearn import linear_model

seed = time.time()
print('the seed is ', seed)
for subj in ['mat', 'por']:
    data = np.loadtxt('student/processed-%s.csv' % subj, delimiter =
',', skiprows = 1)
    regr = linear_model.LinearRegression()
    score_sum = 0.0
    for i in range(5):
        train = np.random.choice([True, False], len(data), replace =
True, p = [0.8, 0.2])
        x_train = data[train,:39]
        y_train = data[train,39]
        x_test = data[~train,:39]
        y_test = data[~train,39]
        regr.fit(x_train, y_train)
        score = regr.score(x_test, y_test)
        print('the R^2 score for run', i + 1, 'is', score)
        score_sum += score
    print('the average R^2 score is', score_sum / 5)
```

Question 2

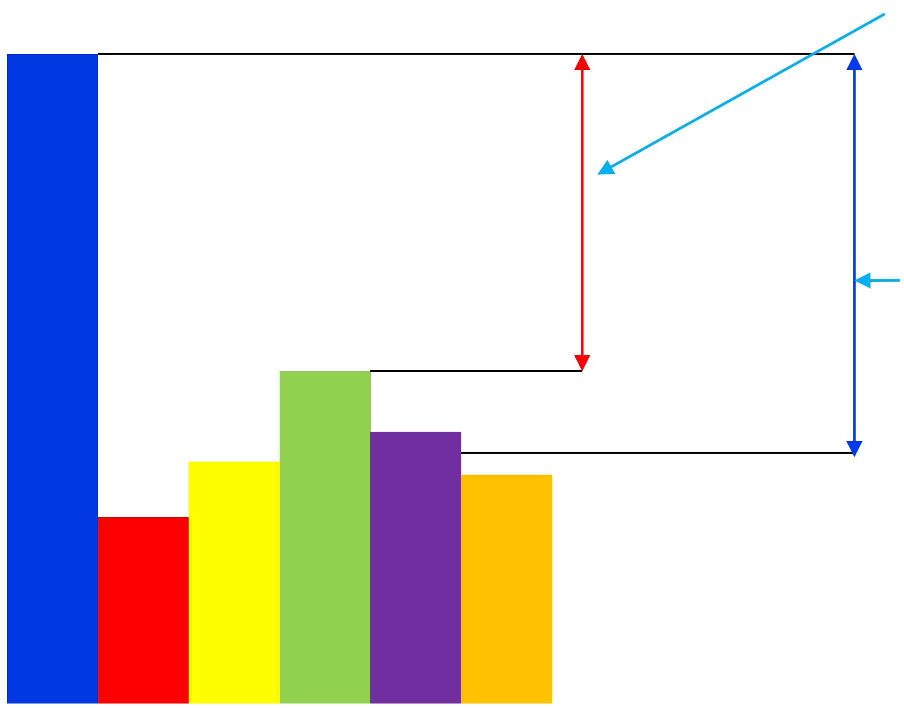
- Use feature_log_prob_
  - if word *w* is distinguishable to class *i* then to class *j*
  - $\mathrm{P}(\mathrm{w}|C_i) \gg \mathrm{P}(\mathrm{w}|C_j)$
  - $\mathrm{w}: argmax\left(\frac{\mathrm{P}(\mathrm{w}|C_i)}{\mathrm{P}(\mathrm{w}|C_j)}\right) = argmax\left(log\left(\frac{\mathrm{P}(\mathrm{w}|C_i)}{\mathrm{P}(\mathrm{w}|C_j)}\right)\right) =$
    $argmax(log\,\mathrm{P}(\mathrm{w}|C_i) - log\,\mathrm{P}(\mathrm{w}|C_j))$

```
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])
```

Question 2

- Extend from bi-classification to multi-classification

$$\min_{j \neq i} [\log P(w|C_i) - \log P(w|C_j)]$$



$$\underset{j \neq i}{avg} [\log P(w|C_i) - \log P(w|C_j)]$$

Question 2

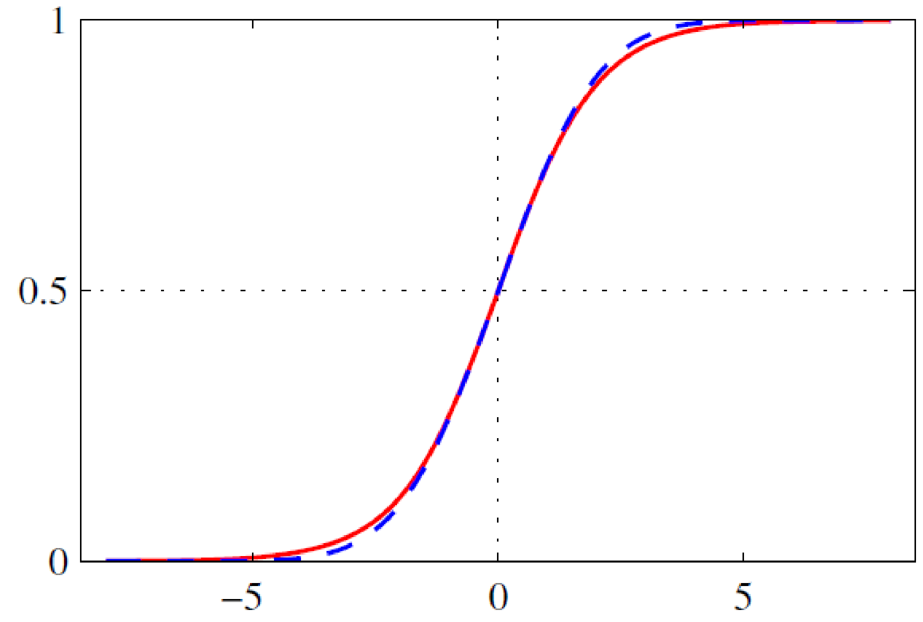
- Extend from bi-classification to multi-classification

```
for i in range(6):
    print('for class', i)
    diff = np.zeros((5, len(feature_names)))
    for j in range(5):
        diff[j,:] = mnb.feature_log_prob_[i,:] - mnb.feature_log_prob
_[(i + j + 1) % 6,:]
    diffmin = np.amin(diff, axis = 0)
    name_diff = {}
    for i in range(len(feature_names)):
        name_diff[feature_names[i]] = diffmin[i]
    names_diff_sorted = sorted(name_diff.items(), key = operator.item
getter(1), reverse = True)
    for j in range(10):
        print('    ', names_diff_sorted[j])
```

Question 3

- Probabilities in logistic function
- Find t_1 for x, s.t.
  - $t_1 \approx \mathrm{f}(x)$, where f is linear
  - $\frac{P(x|C_1)}{P(x|C_0)} = e^{t_1}$
  - $P(x|C_1)+P(x|C_o)=1$
  - $P(x|C_1) = \frac{e^{t_1}}{e^{t_1}+1}$
  - $P(x|C_1) = \frac{1}{1+e^{-t_1}}$

$$\sigma(a) = \frac{1}{1+\exp(-a)}$$



Question 3

- $log\left(\frac{P(x|C_1)}{P(x|C_0)}\right) = t_1$ where $t_1 \approx \mathrm{f}(x)$
- $\mathrm{f}(x) = \sum_i coef_{[w_i]} * count(w) + intercept$
- w: $argmax\left(coef_{[w_i]}\right)$

logit.coef_

```
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])
```

Question 3

- Extend from bi-classification to multi-classification
- Find t_1, t_2, t_3, t_4 for x, s.t.
  - $t_1 \approx f_1(x),\ t_2 \approx f_2(x),\ t_3 \approx f_3(x),\ t_4 \approx f_4(x)$ where $f_1,\ f_2,\ f_3,\ f_4$ are linear
  - $P(x|C_1): P(x|C_2): P(x|C_3): P(x|C_4) = e^{t_1}: e^{t_2}: e^{t_3} : e^{t_4}$
  - $P(x|C_1)+P(x|C_2)+P(x|C_3)+P(x|C_4)=1$
  - $P(x|C_1) = \frac{e^{t_1}}{e^{t_1}+e^{t_2}+e^{t_3}+e^{t_4}}$ Softmax Function

Question 3

- Extend from bi-classification to multi-classification

```
for i in range(6):
    print('for class', i)
    diff = np.zeros((5, len(feature_names)))
    for j in range(5):
        diff[j,:] = logit.coef_[i,:] - logit.coef_[(i + j + 1) % 6,:]
    diffmin = np.amin(diff, axis = 0)
    name_diff = {}
    for i in range(len(feature_names)):
        name_diff[feature_names[i]] = diffmin[i]
    names_diff_sorted = sorted(name_diff.items(), key = operator.item
getter(1), reverse = True)
    for j in range(10):
        print('    ', names_diff_sorted[j])
```