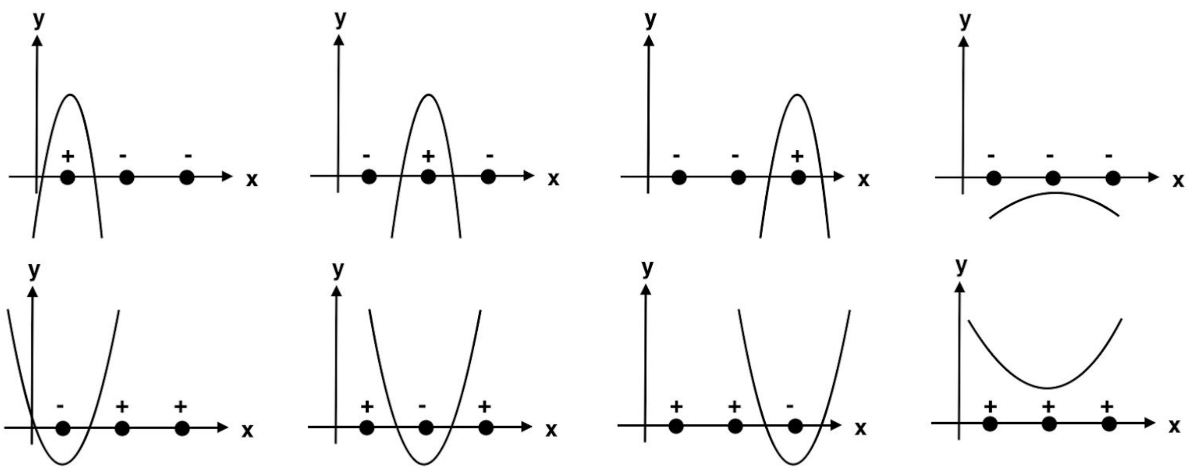
# CS 146 - HW3

Jie-Yun Cheng
004460366

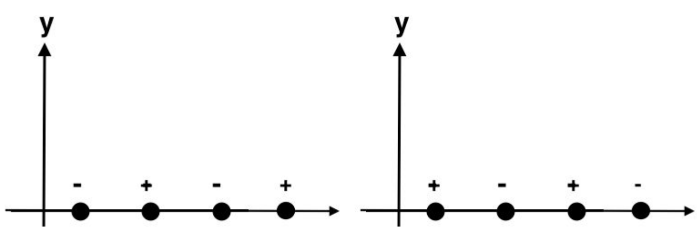
//////////////////////////
// Problem #1 //
//////////////////////////

The inputs are one dimensional points. The function $ax^2 + bx + c$ is a concave or convex function in a two dimensional space. In this dimension, 2 points are always shatterable. Let's check what happens for 3 points. There are $2^3 = 8$ possible combinations to check. It's easy to visualize when we put feature space on the x-axis and $y = ax^2 + bx + c$ to separate data points.




In all eight figures above, all the negative data points are above the $ax^2 + bx + c$ function curve, and all positive data points are below the curve. Hence, VC(H) = 3 works.

Let's see what happens with 4 data points. With the two combinations below, no $ax^2 + bx + c$ function curve can shatter the data points.




To conclude, VC(H) = 3.

///////////////////////////
// Problem #2 //
///////////////////////////
Expand the kernel $K_\beta(x,z)$ first, using the cubic formula: $(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$.

$$K_\beta(x,z) = (1+\beta x\cdot z)^3 = (1+\beta x^T z)^3 = \beta^3(x^T z)^3 + 3\beta^2(x^T z)^2 + 3\beta(x^T z) + 1$$

$$= \beta^3(\sum_i x_i z_i)^3 + 3\beta^2(\sum_i x_i^2 z_i^2 + 2\sum_i\sum_{j>i} x_i z_i x_j z_j)^2 + 3\beta(\sum_i x_i z_i) + 1$$

$$= \beta^3(\sum_i x_i^3 z_i^3 + 6\sum_i\sum_{j>i}\sum_{k>j} x_i x_j x_k z_i z_j z_k + 3\sum_i\sum_{j>i} x_i^2 x_j z_i^2 z_j) + 3\beta^2(\sum_i x_i^2 z_i^2 + 2\sum_i\sum_{j>i} x_i z_i x_j z_j)^2 + 3\beta(\sum_i x_i z_i) + 1$$

The question said vectors x and z are in $\mathbb{R}^2$, so with two dimensions:
To make $K_\beta(x,z)$ map like a 3rd degree polynomial kernel, scale linear terms of x by $\sqrt{\beta}$, scale terms with 2nd order by $\sqrt{\beta^2}$, scale terms with 3rd order by $\sqrt{\beta^3}$, and scale terms with nth order by $\beta^{n/2}$. This would give

$$\varphi_\beta(x) = (1, \sqrt{3\beta}x_1, \sqrt{3\beta}x_2, \sqrt{3\beta^2}x_1{}^2, \sqrt{3\beta^2}x_2{}^2, \sqrt{6\beta^2}x_1x_2, \sqrt{3\beta^3}x_1{}^2x_2, \sqrt{3\beta^3}x_1x_2{}^2, \sqrt{\beta^3}x_1{}^3, \sqrt{\beta^3}x_2{}^3)^T$$

, where $\varphi_\beta(x) * \varphi_\beta(z) = K_\beta(x,z)$.

The point of using kernel is to map models to higher dimension models. Depending on what values of $\beta$ we choose, we can make different kinds of separators that put different weights on different terms, as shown below:

When $\beta = 0$, $K_\beta$ is a linear separator.
When $0 < \beta < 1$, $K_\beta$ puts more weight on lower order terms than higher order terms.
When $\beta = 1$, $K_\beta(x,z) = K(x,z)$.
When $\beta > 1$, $K_\beta$ puts more weight on higher order terms than lower order terms.

////////////////////////////
//   Problem #3  //
////////////////////////////

(a)

For this Support Vector Machine, the two examples are the support vectors on the margins. The classifier should be a linear line that passes the origin and the midpoint of $x_1$ and $x_2$.

$y_1 w^T x_1 = 1 \quad \Rightarrow \quad 1 \times [w_1,\ w_2]\ [1,\ 1]^T = w_1 + w_2 = 1 \quad \Rightarrow \quad w_1 =- 1$

$y_2 w^T x_2 = 1 \quad \Rightarrow \quad -1 \times [w_1, w_2]\ \ [1,\ 0]^T = -w_1 = 1 \quad \Rightarrow \quad w_2 = 2$

Hence, $w^* = [-1,\ 2]^T$

(b)

The classifier should be a linear line that is the bisector between $x_1$ and $x_2$ that has the maximum margin with minimum $||w||$.

$y_1 w^T x_1 + b = 1 \quad \Rightarrow \quad 1 \times [w_1,\ w_2]\ [1,\ 1]^T + b = w_1 + w_2 + b = 1 \quad \Rightarrow \quad w_1 = 0$

$y_2 w^T x_2 + b = 1 \quad \Rightarrow \quad -1 \times [w_1, w_2]\ \ [1,\ 0]^T + b = -(w_1 + b) = 1 \quad \Rightarrow \quad w_2 = 2$

$\Rightarrow \quad b = \ -1$

Hence, $w^* = [0,\ 2]^T$ and $b^* = \ -1$.

Comparing the solutions with and without an offset where $b \neq 0$ , it appears that when there is an offset, the classifier margin is bigger, $w_1$ becomes smaller, and there is no change in $w_2$.

//////////////////////////////
//   Problem #4.1   //
//////////////////////////////

(a) Done.

```python
        ### ========== TODO : START ========== ###
        # part 1a: process each line to populate word_list
        i = 0
        for line in fid:
            for word in extract_words(line):
                if word not in word_list:
                    word_list[word] = i
                    i += 1
        ### ========== TODO : END ========== ###
```

(b) Done.

```python
        ### ========== TODO : START ========== ###
        # part 1b: process each line to populate feature_matrix
        lineNumber = 0
        for line in fid:
            for word in extract_words(line):
                if word in word_list:
                    feature_matrix[lineNumber, word_list[word]] = 1
            lineNumber += 1
        ### ========== TODO : END ========== ###
```

(c) Done.

```python
    ### ========== TODO : START ========== ###
    # part 1: split data into training (training + cross-validation) and testing set
    trainingSetX = X[:560,:]
    trainingSety = y[:560]
    testSetX = X[560:,:]
    testSety = y[560:]
```

(d) As shown in parts 4.1 (a),(b),(c), I have finished the feature extraction and generated the train/test splits.

//////////////////////////////
//   Problem #4.2  //
//////////////////////////////

(a) Done.

```
    ### ========== TODO : START ========== ###
    # part 2a: compute classifier performance
    score = 0
    if metric == "accuracy":
        score = metrics.accuracy_score(y_true, y_label)
    elif metric == "F1-Score":
        score = metrics.f1_score(y_true, y_label)
    elif metric == "AUROC":
        score = metrics.roc_auc_score(y_true, y_pred)
    return score
    ### ========== TODO : END ========== ###
```

(b) Done.

```
    ### ========== TODO : START ========== ###
    # part 2b: compute average cross-validation performance
    sum = 0
    i = 0
    for train_index, valid_index in kf.split(X, y):
        X_train, X_valid = X[train_index], X[valid_index]
        y_train, y_valid = y[train_index], y[valid_index]
        clf.fit(X_train, y_train)
        sum += performance(y_valid, clf.decision_function(X_valid), metric)
        i += 1
    mean = sum/i
    return mean
    ### ========== TODO : END ========== ###
```

When dividing the data into folds for CV, we have to try to keep the class proportions roughly the same across folds, because this may help prevent unlucky splits from happening. For example, if some folds have all positive labels, and some folds have all negative labels, then the performance would be a disaster. Maintaining class proportions across folds can reduce the discrepancy between training error and test error, because all the training sets would be a good representatives of the test set.

(c) Done.

```
    C_range = 10.0 ** np.arange(-3, 3)

    ### ========== TODO : START ========== ###
    # part 2: select optimal hyperparameter using cross-validation
    arrayC = np.array([cv_performance(SVC(C=c, kernel = 'linear'), X, y, kf,
        metric) for c in C_range])
    return arrayC
    ### ========== TODO : END ========== ###
```

(d) Done.

```
    # part 2: create stratified folds (5-fold CV)
    skf = StratifiedKFold(n_splits=5, random_state=1234)
    # part 2: for each metric, select optimal hyperparameter for linear-kernel
        SVM using CV
    for metric in ['accuracy', 'F1-Score', 'AUROC']:
        best = select_param_linear(trainingSetX, trainingSety, skf, metric)
        print(best)
```

```
Linear SVM Hyperparameter Selection based on accuracy:
[0.70894195 0.71074376 0.80603268 0.81462711 0.81818274 0.81818274]
Linear SVM Hyperparameter Selection based on F1-Score:
[0.82968282 0.8305628  0.87547268 0.87486483 0.87656215 0.87656215]
Linear SVM Hyperparameter Selection based on AUROC:
[0.81054948 0.81107835 0.85755274 0.87123274 0.86957902 0.86957902]
Jie-Yuns-MBP:src jycheng$
```

To put into tabular format (up to the fourth decimal place):

| C | accuracy | F1-score | AUROC |
|---|---|---|---|
| $10^{-3}$ | 0.7089 | 0.8296 | 0.8105 |
| $10^{-2}$ | 0.7107 | 0.8305 | 0.8110 |
| $10^{-1}$ | 0.8060 | 0.8754 | 0.8575 |
| $10^{0}$ | 0.8146 | 0.8748 | **0.8712** |
| $10^{1}$ | **0.8181** | **0.8765** | 0.8695 |
| $10^{2}$ | **0.8181** | **0.8765** | 0.8695 |
| best C | $10^1$ and $10^2$ | $10^1$ and $10^2$ | $10^0$ |

As C increases, both performance based on accuracy metric and F1-score metric become better. And based on AUROC, the rate of performance increase becomes slower.

//////////////////////////////
// Problem #4.3 //
//////////////////////////////

(a) Done. Choose C = 100 for accuracy and F1-score. Choose C = 1 for AUROC.

```
    ### ========== TODO : START ========== ###
    # part 3: return performance on test data by first computing predictions and then calling performance
    bestC = 100
    if metric == "AUROC":
        bestC = 1
    print ('Linear SVM test based on ' + str(metric) + ' with C = ' + str(bestC) + ':')
    score = performance(y, clf.decision_function(X), metric)
    return score
    ### ========== TODO : END ========== ###
```

```
    # part 3: train linear-kernel SVMs with selected hyperparameters
    clf100 = SVC(C = 100, kernel = 'linear')
    clf100.fit(trainingSetX, trainingSety)
    clf1 = SVC(C = 1, kernel = 'linear')
    clf1.fit(trainingSetX, trainingSety)
```

(b) Done.

```
    # part 3: report performance on test data
    for metric in ['accuracy', 'F1-Score']:
        print (performance_test(clf100, testSetX, testSety, metric))
    for metric in ['AUROC']:
        print (performance_test(clf1, testSetX, testSety, metric))
```

(c) Done.

```
Linear SVM test based on accuracy with C = 100:
0.7428571428571429
Linear SVM test based on F1-Score with C = 100:
0.43749999999999994
Linear SVM test based on AUROC with C = 1:
0.7405247813411079
Jie-Yuns-MBP:src jycheng$
```

| | accuracy | F1-score | AUROC |
|---|---|---|---|
| C | 100 | 100 | 1 |
| performance | 0.7428 | 0.4374 | 0.7405 |