Machine Learning and Computational Statistics
PRELIMINARY VERSION Homework 5: Generalized Hinge Loss and Multiclass SVM

Yidi Zhang (yz3464)

1 Introduction

The goal of this problem set is to get more comfortable with the multiclass hinge loss and multiclass SVM. In several problems below, you are asked to justify that certain functions are convex. For these problems, you may use any of the rules about convex functions described in our notes on Convex Optimization (https://davidrosenberg.github.io/mlcourse/Notes/convex-optimization.pdf) or in the Boyd and Vandenberghe book. In particular, you will need to make frequent use of the following result: If $f_1, \ldots, f_m : \mathbf{R}^n \to \mathbf{R}$ are convex, then their pointwise maximum

$$f(x) = \max\{f_1(x), \ldots, f_m(x)\}$$

is also convex.

2 Convex Surrogate Loss Functions

It's common in machine learning that the loss functions we really care about lead to optimization problems that are not computationally tractable. The 0/1 loss for binary classification is one such example[1]. Since we have better machinery for minimizing convex functions, a standard approach is to find a **convex surrogate loss function.** A convex surrogate loss function is a convex function that is an upper bound for the loss function of interest[2]. If we can make the upper bound small, then the loss we care about will also be small[3]. Below we will show that the multiclass hinge loss based on a class-sensitive loss Δ is a convex surrogate for the multiclass loss function Δ, when we have a linear hypothesis space. We'll start with a special case, that the hinge loss is a convex surrogate for the 0/1 loss.

2.1 Hinge loss is a convex surrogate for 0/1 loss

1. Let $f : \mathcal{X} \to \mathbf{R}$ be a classification score function for binary classification.

   (a) For any example $(x, y) \in \mathcal{X} \times \{-1, 1\}$, show that

   $$1(y \neq \text{sign}(f(x)) \leq \max\{0, 1 - yf(x)\},$$

   where $\text{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$.

   Answer 2.1.a:
   We can conclude that $\max\{0, 1 - yf(x)\} \geq 0$.
   So when $y = -1$:
   if $f(x) < 0$, $yf(x) > 0$, $\text{sign}(f(x)) = -1$, so $0 \leq \max\{0, 1 - yf(x)\}$
   if $f(x) = 0$, $yf(x) = 0$, $\max\{0, 1 - yf(x)\} = 1$, $1(y \neq \text{sign}(f(x)) = 1$, $1 \leq 1$, so $1(y \neq \text{sign}(f(x)) \leq \max\{0, 1 - yf(x)\}$.
   if $f(x) > 0$, $yf(x) < 0$, $\max\{0, 1 - yf(x)\} > 1$, $1(y \neq \text{sign}(f(x)) = 1$, so $1(y \neq \text{sign}(f(x)) \leq \max\{0, 1 - yf(x)\}$.

   So when $y = 1$:
   if $f(x) < 0$, $yf(x) < 0$, $\text{sign}(f(x)) = -1$, $\max\{0, 1 - yf(x)\} > 1$, $1(y \neq \text{sign}(f(x)) = 1$, so $1(y \neq \text{sign}(f(x)) \leq \max\{0, 1 - yf(x)\}$.
   if $f(x) = 0$, $yf(x) = 0$, $\max\{0, 1 - yf(x)\} = 1$, $1(y \neq \text{sign}(f(x)) = 1$, $1 \leq 1$, so $1(y \neq \text{sign}(f(x)) \leq \max\{0, 1 - yf(x)\}$.

[1] Interestingly, if our hypothesis space is linear classifiers and we are in the "realizable" case, which means that there is some hypothesis that achieves 0 loss (with the 0/1 loss), then we can efficiently find a good hypothesis using linear programming. This is not difficult to see: each data point gives a single linear constraint, and we are looking for a vector that satisfies the constraints for each data point.

[2] At this level of generality, you might be wondering: "A convex function of WHAT?". For binary classification, we usually are talking about a convex function of the margin. But to solve our machine learning optimization problems, we will eventually need our loss function to be a convex function of some $w \in \mathbf{R}^d$ that parameterizes our hypothesis space. It'll be clear in what follows what we're talking about.

[3] This is actually fairly weak motivation for a convex surrogate. Much better motivation comes from the more advanced theory of **classification calibrated** loss functions. See Bartlett et al's paper "Convexity, Classification, and Risk Bounds." http://www.eecs.berkeley.edu/~wainwrig/stat241b/bartlettetal.pdf

if $f(x) > 0$, $yf(x) > 0$, $\max\{0, 1 - yf(x)\} > 0$, $1(y \neq \text{sign}(f(x)) \leq 0$, so $1(y \neq \text{sign}(f(x)) \leq \max\{0, 1 - yf(x)\}$.
To sum up, we can conclude that $1(y \neq \text{sign}(f(x)) \leq \max\{0, 1 - yf(x)\}$.

(b) Show that the hinge loss $\max\{0, 1 - m\}$ is a convex function of the margin m.

Answer 2.1.b:

Since $1 - m$ is a affine function and convex function, so $1 - m$ is a convex function. 0 is a const and $1 - m$ is a convex function, according to:
If $f_1, \ldots, f_m : \mathbf{R}^n \to \mathbf{R}$ are convex, then their pointwise maximum

$$f(x) = \max\{f_1(x), \ldots, f_m(x)\}$$

is also convex, we can conclude that the hinge loss $\max\{0, 1 - m\}$ is a convex function of the margin m.

(c) Suppose our prediction score functions are given by $f_w(x) = w^T x$. The hinge loss of f_w on any example (x, y) is then $\max\{0, 1 - yw^T x\}$. Show that this is a convex function of w.

Answer 2.1.c:

Since $1 - yw_i x$ is a convex function, w_i is chosen from w, so $1 - yw^T x$ is a convex function. Since 0 is a const and $1 - yw^T x$ is a convex function, according to:
If $f_1, \ldots, f_m : \mathbf{R}^n \to \mathbf{R}$ are convex, then their pointwise maximum

$$f(x) = \max\{f_1(x), \ldots, f_m(x)\}$$

is also convex, thus we can conclude that the hinge loss $\max\{0, 1 - yw^T x\}$ is a convex function of the margin w.

2.2 Generalized Hinge Loss

Consider the multiclass output space $\mathcal{Y} = \{1, \ldots, k\}$. Suppose we have a base hypothesis space $\mathcal{H} = \{h : \mathcal{X} \times \mathcal{Y} \to \mathbf{R}\}$ from which we select a compatibility score function. Then our final multiclass hypothesis space is $\mathcal{F} = \left\{f(x) = \arg\max_{y \in \mathcal{Y}} h(x, y) \mid h \in \mathcal{H}\right\}$. Since functions in $\mathcal{F}$ map into $\mathcal{Y}$, our action space $\mathcal{A}$ and output space $\mathcal{Y}$ are the same. Nevertheless, we will write our class-sensitive loss function as $\Delta : \mathcal{Y} \times \mathcal{A} \to \mathbf{R}$, even though $\mathcal{Y} = \mathcal{A}$. We do this to indicate that the true class goes in the first slot of the function, while the prediction (i.e. the action) goes in the second slot. This is important because we do not assume that $\Delta(y, y') = \Delta(y', y)$. It would not be unusual to have this asymmetry in practice. For example, false alarms may be much less costly than no alarm when indeed something is going wrong.

Our ultimate goal would be to find $f \in \mathcal{F}$ minimizing the empirical cost-sensitive loss:

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \Delta\left(y_i, f(x_i)\right).$$

Since binary classification with 0/1 loss is both intractable and a special case of this formulation, we know that this more general formulation must also be computationally intractable. Thus we are looking for a convex surrogate loss function.

1. Suppose we have chosen an $h \in \mathcal{H}$, from which we get the decision function $f(x) = \arg\max_{y \in \mathcal{Y}} h(x, y)$. Justify that for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we have

   $$h(x, y) \leq h(x, f(x)).$$

   Answer 2.2.1:

   Since $f(x) = \arg\max_{y \in \mathcal{Y}} h(x, y)$, so when $y = f(x)$, $h(x, y)$ can obtain its maximum value. So $h(x, y) \leq h(x, f(x))$.

2. Justify the following two inequalities:

$$\begin{aligned}\Delta\left(y, f(x)\right) &\leq \Delta\left(y, f(x)\right) + h(x, f(x)) - h(x, y) \\ &\leq \max_{y' \in \mathcal{Y}} \left[\Delta\left(y, y'\right)\right) + h(x, y') - h(x, y)]\end{aligned}$$

The RHS of the last expression is called the **generalized hinge loss:**

$$\ell\left(h, (x, y)\right) = \max_{y' \in \mathcal{Y}} \left[\Delta\left(y, y'\right)\right) + h(x, y') - h(x, y)\right].$$

We have shown that for any $x \in \mathcal{X}, y \in \mathcal{Y}, h \in \mathcal{H}$ we have

$$\ell\left(h, (x, y)\right) \geq \Delta(y, f(x)),$$

where, as usual, $f(x) = \arg\max_{y \in \mathcal{Y}} h(x, y)$. [You should think about why we cannot write the generalized hinge loss as $\ell\left(f, (x, y)\right)$.]

Answer 2.2.2:

According to 2.2.1, we can conclude that $h(x, f(x)) - h(x, y) \geq 0$, thus

$$\Delta\left(y, f(x)\right) \leq \Delta\left(y, f(x)\right) + h(x, f(x)) - h(x, y)$$

Obviously, $\max_{y' \in \mathcal{Y}} \left[\Delta\left(y, y'\right)\right) + h(x, y') - h(x, y)] \geq \Delta\left(y, f(x)\right) + h(x, f(x)) - h(x, y)$, so we can conclude that:

$$\Delta\left(y, f(x)\right) \leq \max_{y' \in \mathcal{Y}} \left[\Delta\left(y, y'\right)\right) + h(x, y') - h(x, y)]$$

3. We now introduce a specific base hypothesis space $\mathcal{H}$ of linear functions. Consider a class-sensitive feature mapping $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbf{R}^d$, and $\mathcal{H} = \left\{h_w(x, y) = \langle w, \Psi(x, y)\rangle \mid w \in \mathbf{R}^d\right\}$. Show that we can write the generalized hinge loss for $h_w(x, y)$ on example (x_i, y_i) as

$$\ell\left(h_w, (x_i, y_i)\right) = \max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right)) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i)\rangle\right].$$

Answer 2.2.3:

$$\begin{aligned}
&\ell\left(h, (x, y)\right) = \max_{y' \in \mathcal{Y}} \left[\Delta\left(y, y'\right)) + h(x, y') - h(x, y)\right] \\
&= \max_{y' \in \mathcal{Y}} \left[\Delta\left(y, y'\right)) + \langle w, \Psi(x_i, y)\rangle - \langle w, \Psi(x_i, y_i)\rangle\right] \\
&\quad = \max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right)) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i)\rangle\right].
\end{aligned}$$

4. We will now show that the generalized hinge loss $\ell\left(h_w, (x_i, y_i)\right)$ is a convex function of w. Justify each of the following steps.

   (a) The expression $\Delta(y_i, y) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i)\rangle$ is an affine function of w.

   Answer 2.2.4.a:

   $w^T(\Psi(x_i, y) - \Psi(x_i, y_i)$ is a affine function, so we can conclude that $\Delta(y_i, y) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i)\rangle$ is a affine function.

   (b) The expression $\max_{y\in\mathcal{Y}}\left[\Delta\left(y_i, y\right)\right) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i)\rangle]$ is a convex function of w.

   Answer 2.2.4.b:

   According 2.1.c, since $\langle w, \Psi(x_i, y) - \Psi(x_i, y_i)\rangle$ is convex function, so we can get $\max_{y\in\mathcal{Y}}\left[\Delta\left(y_i, y\right)\right) + \langle w, \Psi(x_i, y$ is a convex function of w.

5. Conclude that $\ell\left(h_w, (x_i, y_i)\right)$ is a convex surrogate for $\Delta(y_i, f_w(x_i))$.

   Answer 2.2.5:

   From above prove, we know $\ell\left(h_w, (x_i, y_i)\right)$ is a convex function. Besides,

$$\Delta(y_i, f_w(x_i)) \leq \ell\left(h_w, (x_i, y_i)\right)$$

   Thus $\ell\left(h_w, (x_i, y_i)\right)$ is a convex surrogate for $\Delta(y_i, f_w(x_i))$.

3 SGD for Multiclass SVM

Suppose our output space and our action space are given as follows: $\mathcal{Y} = \mathcal{A} = \{1, \ldots, k\}$. Given a non-negative class-sensitive loss function $\Delta : \mathcal{Y} \times \mathcal{A} \to \mathbf{R}^{\geq 0}$ and a class-sensitive feature mapping $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbf{R}^d$. Our prediction function $f : \mathcal{X} \to \mathcal{Y}$ is given by

$$f_w(x) = \arg\max_{y \in \mathcal{Y}} \langle w, \Psi(x, y) \rangle$$

1. For a training set $(x_1, y_1), \ldots (x_n, y_n)$, let $J(w)$ be the ℓ_2-regularized empirical risk function for the multiclass hinge loss. We can write this as

$$J(w) = \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle \right].$$

   We will now show that $J(w)$ is a convex function of w. Justify each of the following steps. As we've shown it in a previous problem, you may use the fact that $w \mapsto \max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle \right]$ is a convex function.

   (a) $\frac{1}{n} \sum_{i=1}^{n} \max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle \right]$ is a convex function of w.

   Answer 3.1.a:

   We already know $w \mapsto \max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle \right]$ is a convex function, so $\frac{1}{n} \sum_{i=1}^{n} \max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle \right]$ is also a convex function.

   (b) $\|w\|^2$ is a convex function of w.

   Answer 3.1.b:

   We know that $|x|^p$ is convex on $\mathbf{R}$, thus we can conclude that $\|w\|^2$ is a convex function.

   (c) $J(w)$ is a convex function of w.

   Answer 3.1.c:

   From above question, we know $\frac{1}{n} \sum_{i=1}^{n} \max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle \right]$ is a convex function and $\|w\|^2$ is a convex function. So

$$J(w) = \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle \right]$$

   is a convex function.

2. Since $J(w)$ is convex, it has a subgradient at every point. Give an expression for a subgradient of $J(w)$. You may use any standard results about subgradients, including the result from an earlier homework about subgradients of the pointwise maxima of functions. (Hint: It may be helpful to refer to $\hat{y}_i = \arg\max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i)\rangle\right]$.)

Answer 3.2:

First we choose the function that achieves the maximum at the point:

$$\hat{y}_i = \arg\max_{y \in \mathcal{Y}} \left[\Delta\left(y_i, y\right) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i)\rangle\right].$$

The subgradient of this point is:

$$\Delta J(w) = 2\lambda w + (\Psi(x, \hat{y}) - \Psi(x, \hat{y})).$$

Thus we can conclude the subgradients of the pointwise maxima of functions is :

$$\partial J(w) = 2\lambda w + \frac{1}{n}\sum_{i=1}^{n}(\Psi(x, y) - \Psi(x, y)).$$

3. Give an expression the stochastic subgradient based on the point (x_i, y_i).

   Answer 3.3:

   The expression of the stochastic subgradient based on the point (x_i, y_i) is:

$$2\lambda w + (\Psi(x_i, \hat{y}_i) - \Psi(x_i, y_i)).$$

4. Give an expression for a minibatch subgradient, based on the points $(x_i, y_i), \ldots, (x_{i+m-1}, y_{i+m-1})$.

   Answer 3.4:

   The expression of the minibatch subgradient, based on the points $(x_i, y_i), \ldots, (x_{i+m-1}, y_{i+m-1})$ is:

$$\partial J(w) = 2\lambda w + \frac{1}{m}\sum_{i=1}^{n}(\Psi(x_{i+m-1}, \hat{y}_{i+m-1}) - \Psi(x_{i+m-1}, y_{i+m-1})).$$

4 [OPTIONAL] Another Formulation of Generalized Hinge Loss

In lecture we defined the **margin** of the compatibility score function h on the ith example (x_i, y_i) for class y as

$$m_{i,y}(h) = h(x_i, y_i) - h(x_i, y),$$

and the loss on an individual example (x_i, y_i) to be:

$$\max_y \left([\Delta(y_i, y) - m_{i,y}(h)]_+ \right).$$

Here we investigate whether this is just an instance of the generalized hinge loss $\ell(h, (x, y))$ defined above.

1. Show that $\ell(h, (x_i, y_i)) = \max_{y' \in \mathcal{Y}} [\Delta(y_i, y') - m_{i,y'}(h)]$. (In other words, it looks just like loss above, but without the positive part.)

   Answer 4.1:

   Since $m_{i,y}(h) = h(x_i, y_i) - h(x_i, y) \leq 0$, so we can get $\max_y \left([\Delta(y_i, y) - m_{i,y}(h)]_+ \right) \geq 0$.

   Thus

   $$\ell(h, (x_i, y_i)) = \max_{y' \in \mathcal{Y}} [\Delta(y_i, y') - m_{i,y'}(h)]$$

2. Suppose $\Delta(y, y') \geq 0$ for all $y, y' \in \mathcal{Y}$. Show that for any example (x_i, y_i) and any score function h, the multiclass hinge loss we gave in lecture and the generalized hinge loss presented above are equivalent, in the sense that

$$\max_{y \in \mathcal{Y}} \left([\Delta(y_i, y) - m_{i,y}(h)]_+ \right) = \max_{y \in \mathcal{Y}} \left(\Delta(y_i, y) - m_{i,y}(h) \right).$$

(Hint: This is easy by piecing together other results we have already attained regarding the relationship between ℓ and Δ.)

3. In the context of the generalized hinge loss, $\Delta(y, y')$ is like the "target margin" between the score for true class y and the score for class y'. Suppose that our prediction function f gets the correct class on x_i. That is, $f(x_i) = \arg\max_{y' \in \mathcal{Y}} h(x_i, y') = y_i$. Furthermore, assume that all of our target margins are reached or exceeded. That is

$$m_{i,y}(h) = h(x_i, y_i) - h(x_i, y) \geq \Delta(y_i, y),$$

for all $y \neq y_i$. It seems like in this case, we should have 0 loss. This is almost the case. Show that $\ell\left(h, (x_i, y_i)\right) = 0$ if we assume that $\Delta\left(y, y\right) = 0$ for all $y \in \mathcal{Y}$.

5 [OPTIONAL] Hinge Loss is a Special Case of Generalized Hinge Loss

Let $\mathcal{Y} = \{-1, 1\}$. Let $\Delta(y, \hat{y}) = 1(y \neq \hat{y})$. If $g(x)$ is the score function in our binary classification setting, then define our compatibility function as

$$\begin{aligned} h(x, 1) &= g(x)/2 \\ h(x, -1) &= -g(x)/2. \end{aligned}$$

Show that for this choice of h, the multiclass hinge loss reduces to hinge loss:

$$\ell\left(h, (x, y)\right) = \max_{y' \in \mathcal{Y}} \left[\Delta\left(y, y'\right)\right) + h(x, y') - h(x, y)] = \max\left\{0, 1 - yg(x)\right\}$$

6 Multiclass Classification - Implementation

6.1 One-vs-All (also known as One-vs-Rest)

In this problem we will implement one-vs-all multiclass classification. Our approach will assume we have a binary base classifier that returns a score, and we will predict the class that has the highest score.

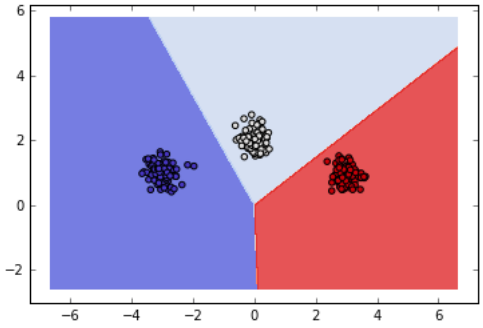
1. Complete the class OneVsAllClassifier in the skeleton code. Following the OneVsAllClassifier code is a cell that extracts the results of the fit and plots the decision region. Include these results in your submission.

   Answer 6.1.1:

```
from sklearn.base import BaseEstimator, ClassifierMixin, clone

class OneVsAllClassifier(BaseEstimator, ClassifierMixin):
    """
    One-vs-all classifier
    We assume that the classes will be the integers 0,..,(n_classes-1).
    We assume that the estimator provided to the class, after fitting, has a "
    decision_function" that
    returns the score for the positive class.
    """
    def __init__(self, estimator, n_classes):
        """
        Constructed with the number of classes and an estimator (e.g. an
        SVM estimator from sklearn)
        @param estimator : binary base classifier used
        @param n_classes : number of classes
        """
        self.n_classes = n_classes
        self.estimators = [clone(estimator) for _ in range(n_classes)]
        self.fitted = False

    def fit(self, X, y=None):
        """
        This should fit one classifier for each class.
        self.estimators[i] should be fit on class i vs rest
        @param X: array-like, shape = [n_samples,n_features], input data
        @param y: array-like, shape = [n_samples,] class labels
        @return returns self
        """

        for i, estimator in enumerate(self.estimators):
            yi = np.zeros(len(y))
            yi[np.where(y==i)[0]] = 1
            #estimator = self.estimators[i]
            estimator.fit(X, yi)
        self.fitted = True
        return self

    def decision_function(self, X):
        """
        Returns the score of each input for each class. Assumes
        that the given estimator also implements the decision_function method (which
     sklearn SVMs do),
```

```
        and that fit has been called.
        @param X : array-like, shape = [n_samples, n_features] input data
        @return array-like, shape = [n_samples, n_classes]
        """
        if not self.fitted:
            raise RuntimeError("You must train classifer before predicting data.")

        if not hasattr(self.estimators[0], "decision_function"):
            raise AttributeError(
                "Base estimator doesn't have a decision_function attribute.")

        n_samples = X.shape[0]
        n_class = self.n_classes
        output = np.zeros((n_samples,n_class))

        for i, estimator in enumerate(self.estimators):
            temp = estimator.decision_function(X)
            output[:,i] = temp

        return output


    def predict(self, X):
        """
        Predict the class with the highest score.
        @param X: array-like, shape = [n_samples,n_features] input data
        @returns array-like, shape = [n_samples,] the predicted classes for each
    input
        """

        arr = self.decision_function(X)
        N = len(arr)
        output = np.zeros(len(X))
        for i in range(N):
            output[i] = np.where(arr[i]==max(arr[i]))[0][0]
        return output
```

```
Coeffs 0
[[-1.0585309  -0.90294204]]
Coeffs 1
[[ 0.33928049 -0.0740662 ]]
Coeffs 2
[[ 0.89164193 -0.8260075 ]]

array([[100,   0,   0],
       [  0, 100,   0],
       [  0,   0, 100]])
```



2. [Optional] Normalize the vectors corresponding to each of the linear SVM classifiers so that they have unit norm. Evaluate the results and plot the decision regions for these normalized vectors.

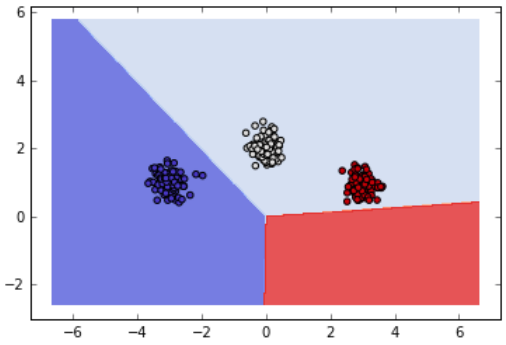
Answer 6.1.2:

```
from sklearn import svm
from sklearn import preprocessing
svm_estimator = svm.LinearSVC(loss='hinge', fit_intercept=False, C=200)
clf_onevsall = OneVsAllClassifier(svm_estimator, n_classes=3)
clf_onevsall.fit(X,y)

for i in range(3) :
    print("Coeffs %d"%i)
     #Will fail if you haven't implemented fit yet
    clf_onevsall.estimators[i].coef_ = preprocessing.normalize(clf_onevsall.
    estimators[i].coef_)
    print(clf_onevsall.estimators[i].coef_)
# create a mesh to plot in
h = .02  # step size in the mesh
x_min, x_max = min(X[:,0])-3,max(X[:,0])+3
y_min, y_max = min(X[:,1])-3,max(X[:,1])+3
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
mesh_input = np.c_[xx.ravel(), yy.ravel()]

Z = clf_onevsall.predict(mesh_input)
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)

from sklearn import metrics
```

```
metrics.confusion_matrix(y, clf_onevsall.predict(X))
```

```
Coeffs 0
[[-0.76078931 -0.64899894]]
Coeffs 1
[[ 0.64052776  0.76793502]]
Coeffs 2
[[ 0.73412276 -0.67901677]]
```

```
array([[100,   0,   0],
       [  0, 100,   0],
       [  0, 100,   0]])
```



3. [Optional] You may notice that every time you run the cell that fits the models and plots the decision regions, you get slightly different results. This is more pronounced when C is larger, e.g. $C = 200$. Investigate and propose an explanation for this. You may use any means necessary, including google searching and asking other people (just like in real life). [It's generally good to investigate things that look odd – you'll almost always learn something, and sometimes you'll uncover a more serious underlying problem.]

   Answer 6.1.2:

   Since if the value of C is large, the weight on training set X will become large to reduce the effect by the large const. So the slight difference will cause the big difference because of the large weight value and hypethesis space.

6.2 Multiclass SVM

In this question, we will implement stochastic subgradient descent for the linear multiclass SVM described in lecture and in this problem set. We will use the class-sensitive feature mapping approach with the "multivector construction", as described in our multiclass classification lecture and in SSBD Section 17.2.1.

1. Complete the skeleton code for multiclass SVM. Following the multiclass SVM implementation, we have included another block of test code. Make sure to include the results from these tests in your assignment, along with your code.

   Answer 6.2.1:

```
def zeroOne(y,a) :
    '''
    Computes the zero-one loss.
    @param y: output class
    @param a: predicted class
    @return 1 if different, 0 if same
    '''
    return int(y != a)

def featureMap(X,y,num_classes) :
    '''
    Computes the class-sensitive features.
    @param X: array-like, shape = [n_samples,n_inFeatures] or [n_inFeatures,], input
     features for input data
    @param y: a target class (in range 0,..,num_classes-1)
    @return array-like, shape = [n_samples,n_outFeatures], the class sensitive
    features for class y
    '''
    #The following line handles X being a 1d-array or a 2d-array
    num_samples, num_inFeatures = (1,X.shape[0]) if len(X.shape) == 1 else (X.shape
    [0],X.shape[1])
    N = num_classes*num_inFeatures
    output = np.zeros((num_samples,N))
    mapped = np.zeros(N)

    if num_samples == 1:
        mapped[y*num_inFeatures:(y+1)*num_inFeatures] = X
        output = mapped
        return output

    for i, value in enumerate(X):
        mapped = np.zeros(N)
        mapped[y[i]*num_inFeatures:(y[i]+1)*num_inFeatures] = value
        output[i,:] = mapped
    return output

def sgd(X, y, num_outFeatures, subgd, eta = 0.1, T = 10000):
    '''
    Runs subgradient descent, and outputs resulting parameter vector.
    @param X: array-like, shape = [n_samples,n_features], input training data
    @param y: array-like, shape = [n_samples,], class labels
    @param num_outFeatures: number of class-sensitive features
    @param subgd: function taking x,y and giving subgradient of objective
    @param eta: learning rate for SGD
    @param T: maximum number of iterations
```

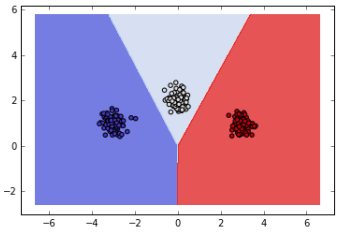
```
    @return: vector of weights
    '''
    num_samples = X.shape[0]
    w_hat = np.zeros(num_outFeatures)
    weight = np.zeros(num_outFeatures)
    idx = list(range(len(X)))

    for t in range (T):
        np.random.shuffle(idx)
        sub_x = X[idx[0]]
        sub_y = y[idx[0]]
        w= eta * subgd(sub_x, sub_y, weight)
        weight = weight -w
        w_hat += weight
    w_hat = w_hat/T
    return w_hat

class MulticlassSVM(BaseEstimator, ClassifierMixin):
    '''
    Implements a Multiclass SVM estimator.
    '''
    def __init__(self, num_outFeatures, lam=1.0, num_classes=3, Delta=zeroOne, Psi=
    featureMap):
        '''
        Creates a MulticlassSVM estimator.
        @param num_outFeatures: number of class-sensitive features produced by Psi
        @param lam: l2 regularization parameter
        @param num_classes: number of classes (assumed numbered 0,..,num_classes-1)
        @param Delta: class-sensitive loss function taking two arguments (i.e.,
    target margin)
        @param Psi: class-sensitive feature map taking two arguments
        '''
        self.num_outFeatures = num_outFeatures
        self.lam = lam
        self.num_classes = num_classes
        self.Delta = Delta
        self.Psi = lambda X,y : Psi(X,y,num_classes)
        self.fitted = False

    def subgradient(self,x,y,w):
        '''
        Computes the subgradient at a given data point x,y
        @param x: sample input
        @param y: sample class
        @param w: parameter vector
        @return returns subgradient vector at given x,y,w
        '''
        yi = y
        h = []

        for yy in range(self.num_classes):
            hsgd = self.Delta(yi,yy) + w.dot(self.Psi(x,yy) - w.dot(self.Psi(x,y)))
            h.append(hsgd)

        #Find the y can make the function obtain the maximum value
        max_value = max(h)
        y_hat = np.where(h == max_value)[0][0]
```

```
        output = 2*self.lam*w.T + self.Psi(x,y_hat) - self.Psi(x,y)
        return output

    def fit(self,X,y,eta=0.1,T=10000):
        '''
        Fits multiclass SVM
        @param X: array-like, shape = [num_samples,num_inFeatures], input data
        @param y: array-like, shape = [num_samples,], input classes
        @param eta: learning rate for SGD
        @param T: maximum number of iterations
        @return returns self
        '''
        self.coef_ = sgd(X,y,self.num_outFeatures,self.subgradient,eta,T)
        self.fitted = True
        return self

    def decision_function(self, X):
        '''
        Returns the score on each input for each class. Assumes
        that fit has been called.
        @param X : array-like, shape = [n_samples, n_inFeatures]
        @return array-like, shape = [n_samples, n_classes] giving scores for each
    sample,class pairing
        '''
        if not self.fitted:
            raise RuntimeError("You must train classifer before predicting data.")

        #Your code goes here and replaces following return statement
        h = np.zeros((X.shape[0], self.num_classes))
        for i,xi in enumerate(X):
            h[i,:] = [self.coef_.dot(self.Psi(xi,yi)) for yi in range(self.
    num_classes)]
        return h

    def predict(self, X):
        '''
        Predict the class with the highest score.
        @param X: array-like, shape = [n_samples, n_inFeatures], input data to
    predict
        @return array-like, shape = [n_samples,], class labels predicted for each
    data point
        '''
        arr = self.decision_function(X)
        N = len(arr)
        output = np.zeros(len(X))
        for i in range(N):
            output[i] = np.where(arr[i]==max(arr[i]))[0][0]
        return output
```

7 [Optional] Audio Classification

In this problem, we will work on the urban sound dataset URBANSOUND8K from the Center for Urban Science and Progress (CUSP) at NYU. We will first extract features from raw audio data using the LibROSA package, and then we will train multiclass classifiers to classify the sounds into 10 sound classes. URBANSOUND8K dataset contains 8732 labeled sound excerpts. For this

```
w:
[-0.29684691 -0.05438605  0.00259882  0.11252727  0.29424809 -0.05814122]

array([[100,   0,   0],
       [  0, 100,   0],
       [  0,   0, 100]])
```



problem, you may use the file UrbanSound8K.csv to randomly sample 2000 examples for training and 2000 examples for validation.

1. In LibROSA, there are many functions for visualizing audio waves and spectra, such as display.waveplot() and display.specshow(). Load a random audio file from each class as a floating point time series with librosa.load(), and plot their waves and linear-frequency power spectrogram. If you are interested, you can also play the audio in the notebook with functions display() and Audio() in IPython.display.

2. Mel-frequency cepstral coefficients (MFCC) are a commonly used feature for sound processing. We will use MFCC and its first and second differences (like discrete derivatives) as our features for classfication. First, use function feature.mfcc() from LibROSA to extract MFCC features from each audio sample. (The first MFCC coefficient is typically discarded in sound analysis, but you do not need to. You can test whether this helps in the optional problem below.) Next, use function feature.delta() to calculate the first and second differences of MFCC. Finally, combine these features and normalize each feature to zero mean and unit variance.

3. Train a linear multiclass SVM on your 2000 example training set. Evaluate your results on the validation set in terms of 0/1 error and generate a confusion table. Compare the results to a one-vs-all classifier using a binary linear SVM as the base classifier. For each model, may use your code from the previous problem, or you may use another implementation (e.g. from sklearn).

4. [More Optional] Compare results to any other multiclass classification methods of your choice.

5. [More Optional] Try different feature sets and see how they affect performance.