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

Due: Tuesday, April 11, 2017, at 10pm (Submit via Gradescope)

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython), though if you need to you may scan handwritten work. You may find the minted package convenient for including source code in your LaTeX document. If you are using LyX, then the listings package tends to work better.

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,

[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.

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

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

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

   ANSWER

   If $y \neq sign(f(x))$, $yf(x) \leq 0$, and $1 - yf(x) \geq 1$ therefore, the inequality holds,

   If $y = sign(f(x))$, $lhs = 0$ and $rhs \geq 0$ therefore, the inequality holds,

[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

(b) Show 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.

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 $\mathcal{F} = \{f(x) = \arg\max_{y \in \mathcal{Y}} h(x, y) \mid h \in \mathcal{H}\}$. 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

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

2. Justify the following two inequalities:

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

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

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

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 $\ell\left(f, (x, y)\right)$.]

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

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

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.

   (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.

5. Conclude that $\ell\left(h_w, (x_i, y_i)\right)$ 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

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

   for some $\lambda > 0$. 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.

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

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

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

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

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

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.)

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

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

   (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 $\ell(h, (x_i, y_i)) = 0$ if we assume that $\Delta(y, y) = 0$ 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:

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

6 Multiclass Classification - Implementation

In this problem we will work on a simple three-class classification example, similar to the one given in lecture. The data is generated and plotted for you in the skeleton code.

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.

```
class OneVsAllClassifier(BaseEstimator, ClassifierMixin):

    def __init__(self, estimator, n_classes):

        self.n_classes = n_classes
        self.estimators = [clone(estimator) for _ in range(n_classes)]
        self.fitted = False

    def fit(self, X, y=None):

        for i in range(self.n_classes):

            tempy = np.copy(y)

            for j in range(len(tempy)):

                if tempy[j] == i:
                    tempy[j] = 1

                else:
                    tempy[j] = -1

                self.estimators[i].fit(X, tempy)

        self.fitted = True
        return self
```

```
    def decision_function(self, X):

        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.")


        s = (X.shape[0], self.n_classes)
        dec = np.zeros(s)

        for i in range(self.n_classes):

            dec[:,i] = self.estimators[i].decision_function(X)

        return(dec)


    def predict(self, X):

        prediction = np.zeros(X.shape[0])
        dec = self.decision_function(X)

        for i in range(X.shape[0]):

            prediction[i] = np.argmax(dec[i])

        return prediction
```

Coeffs 0 [[-1.21554249 -0.83295331]] Coeffs 1 [[0.42090396 -0.30060302]] Coeffs 2 [[0.89162796 -0.82467394]]

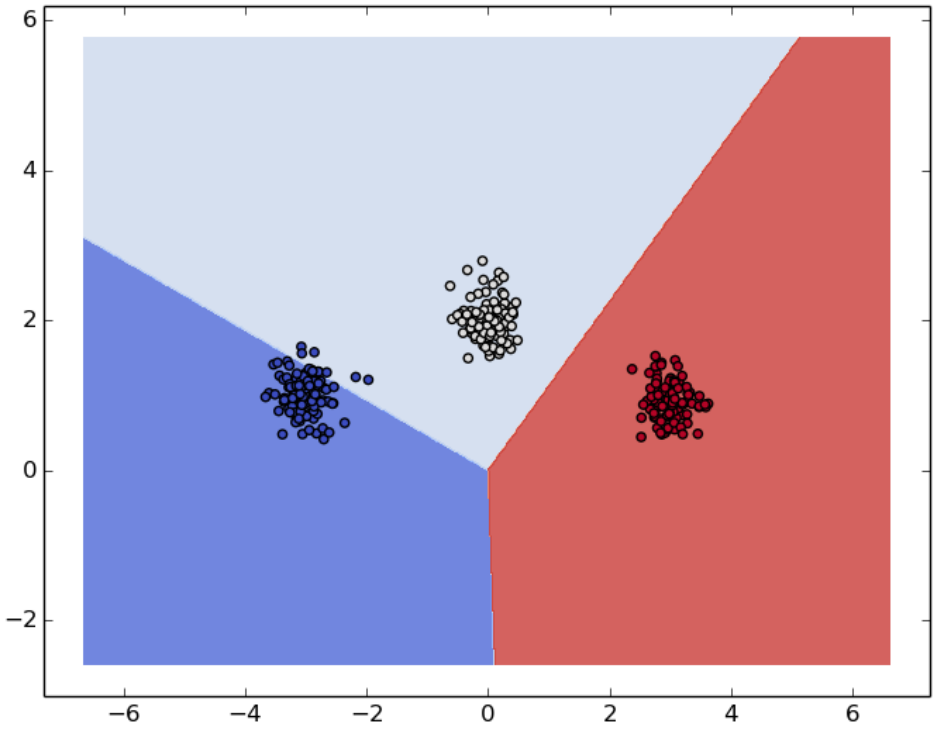

Figure 1: Classification produced after running one vs all

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.

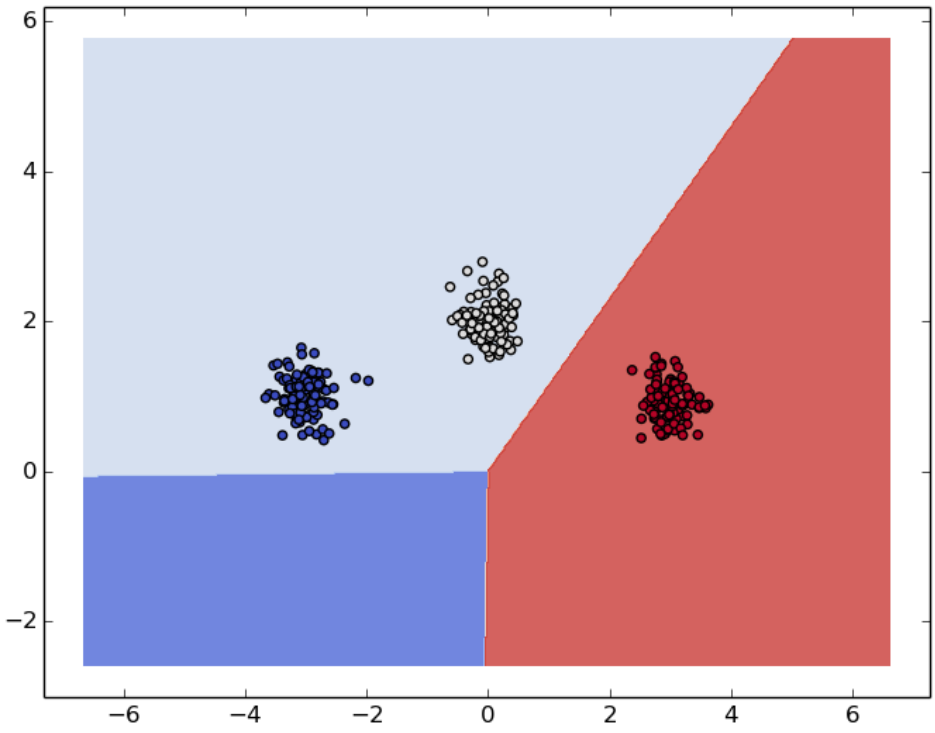

Figure 2: Classification produced after normalizing

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.]

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.

```
def zeroOne(y,a) :

    return int(y != a)

def featureMap(X,y,num_classes) :

    dim = (num_samples, num_outFeatures)
    phi = np.zeros(dim)

    if num_samples == 1:

        new_row = np.zeros(num_outFeatures)
        new_row[y*num_inFeatures:(y+1)*num_inFeatures] = X
        return new_row

    for i in range(X.shape[0]):

        new_row = np.zeros(num_outFeatures)
        new_row[y[i]*num_inFeatures:(y[i]+1)*num_inFeatures] = X[i]
        phi[i,:] = new_row

    return phi



def sgd(X, y, num_outFeatures, subgd, eta = 0.01, T = 10000):

    num_samples = X.shape[0]

    w = np.zeros(num_outFeatures)
    meanw = np.zeros(num_outFeatures)

    ind_arr = list(range(num_samples))

    for i in range(T):
```

```
            # np.random.shuffle(ind_arr)
            j = np.random.randint(num_samples)
            # for j in ind_arr:

            gradient = subgd(X[j],y[j],w)
            w = w - eta * gradient
            meanw += w

        return meanw/T

    class MulticlassSVM(BaseEstimator, ClassifierMixin):

        def __init__(self, num_outFeatures, lam=1.0, num_classes=3, Delta=zeroOne, Psi=

            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):

            y_cap = np.zeros(self.num_classes)

            for p in range(self.num_classes):

                y_cap[p] = self.Delta(y, p) + np.dot(w, (self.Psi(x,p) - self.Psi(x,y))

            res = 2 * self.lam * w + self.Psi(x, np.argmax(y_cap)) - self.Psi(x,y)

            return res


        def fit(self,X,y,eta=0.1,T=10000):

            self.coef_ = sgd(X,y,self.num_outFeatures,self.subgradient,eta,T)
            self.fitted = True
            return self

        def decision_function(self, X):

            if not self.fitted:
                raise RuntimeError("You must train classifer before predicting data.")
```

```
        s = (X.shape[0], self.num_classes)
        dec = np.zeros(s)

        for i in range(X.shape[0]):

            new_row = np.zeros(self.num_classes)

            for j in range(self.num_classes):

                new_row[j] = np.dot(self.coef_, self.Psi(X[i],j))

            dec[i] = new_row

        return(dec)

    def predict(self, X):

        dec = self.decision_function(X)
        prediction = np.zeros(X.shape[0])

        for i in range(X.shape[0]):

            prediction[i] = np.argmax(dec[i])

        return prediction
```

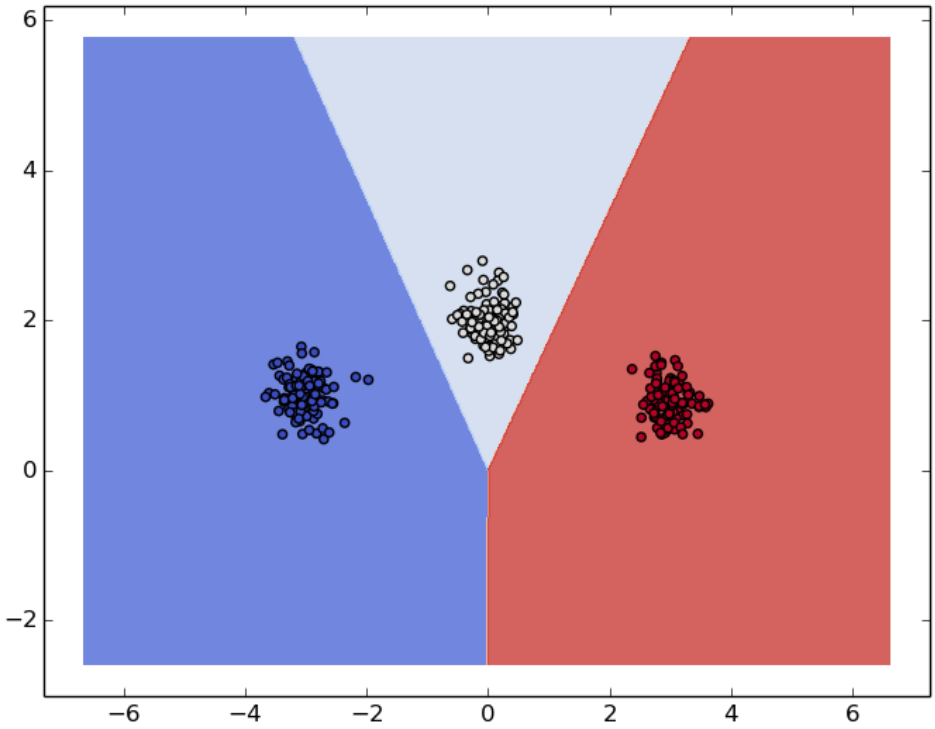

Figure 3: Classification produced using Multiclass SVM

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 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.