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]]

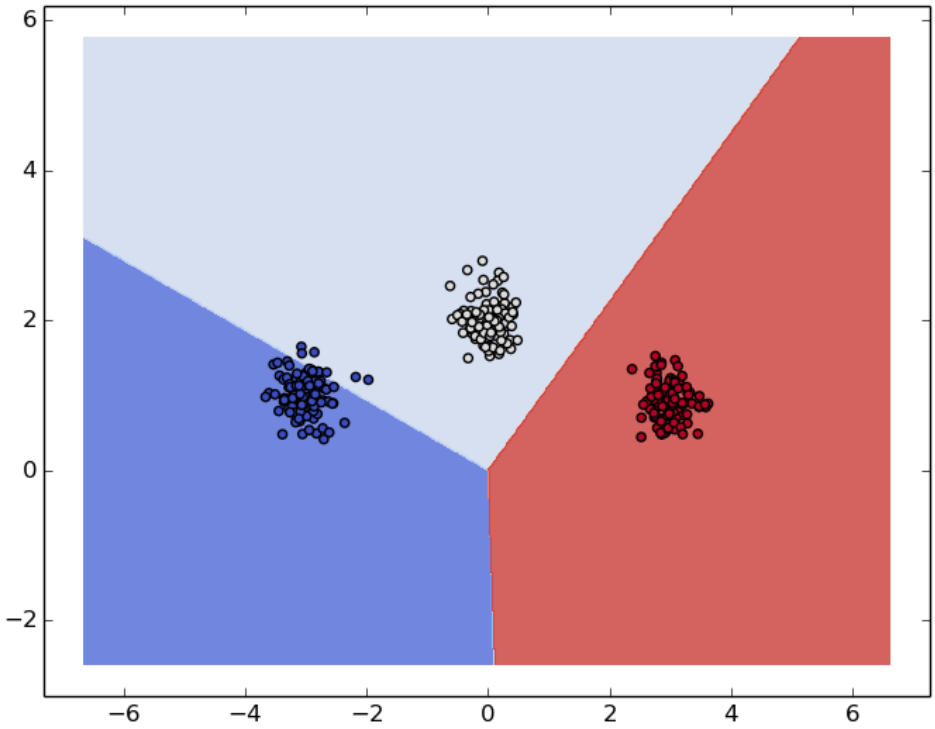

Figure 1: Classification produced after running one vs all

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.

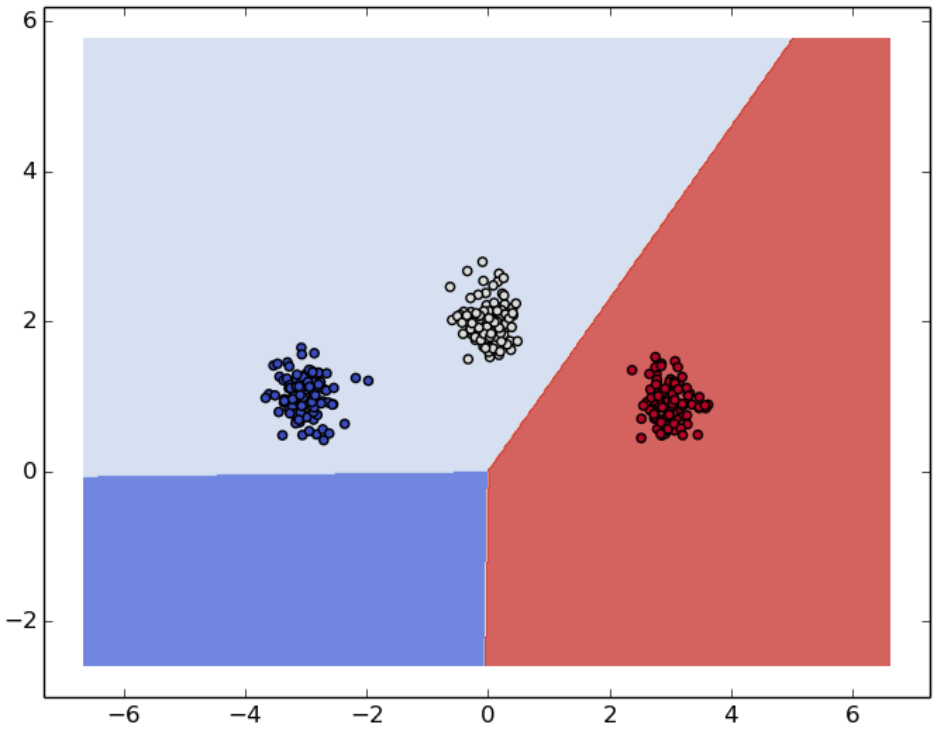

Figure 2: Classification produced after normalizing

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
```

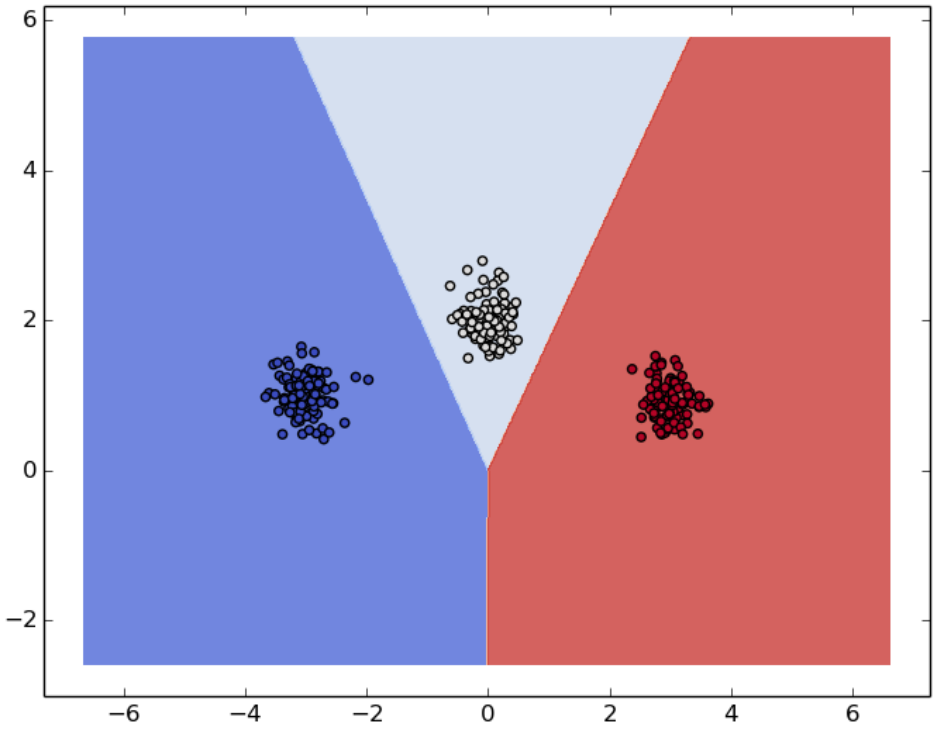

Figure 3: Classification produced using Multiclass SVM

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.