# STA 208: Assignment #2

Due on Monday, April 11, 2016

*Prof. James Sharpnack MW 12:00 - 2:00 P.M.*

**Wenhao Wu**

# Contents

# 1.

The following losses are used as surrogate losses for large margin classification. Demonstrate if they are convex or not, and follow the instructions.

**(a)** exponential loss: $\phi(x) = e^{-x}$

***Answer:*** Since $d^2\phi(x)/dx^2 = e^{-x} > 0$, this function is convex.

**(b)** truncated quadratic loss: $\phi(x) = (\max\{1-x, 0\})^2$

***Answer:*** We can rewrite $\phi(x) = (\max\{1-x, 0\})^2$

$$\phi(x) = \begin{cases} (1-x)^2 & x < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Then the convexity of $\phi(x)$ can be proved by the definition

- $\forall x_1 < 1, x_2 < 1$ or $\forall x_1 \geq 1, x_2 \geq 1$, since both $f(x) = (1-x)^2$ and $g(x) = 0$ are convex functions, we have

$$\phi(tx_1 + (1-t)x_2) \leq t\phi(x_1) + (1-t)\phi(x_2) \tag{2}$$

  $\forall t \in [0, 1]$.

- $\forall x_1 < 1, x_2 \geq 1$, since $\phi(x)$ is monotonically decreasing,

$$\phi(tx_1 + (1-t)x_2) \leq \phi(tx_1) = t\phi(x_1) + (1-t)\phi(x_2) \tag{3}$$

Therefore $\phi(x)$ is convex.

**(c)** hinge loss: $\phi(x) = \max\{1-x, 0\}$

***Answer:*** Since both $f(x) = 1-x$ and $g(x) = 0$ are convex functions, $(\max\{1-x, 0\})$, $\phi(x)$ is also convex.

**(d)** sigmoid loss: $\phi(x) = 1 - \tanh(\kappa x)$, for fixed $\kappa > 0$

***Answer:*** Since

$$\frac{\mathrm{d}\phi}{\mathrm{d}x} = \frac{-4\kappa e^{2\kappa x}}{(1+e^{2\kappa x})} = \kappa\phi(x)(\phi(x) - 2) \tag{4a}$$

$$\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2} = 2\kappa^2\phi(x)(\phi(x) - 2)(\phi(x) - 1) \tag{4b}$$

when $x < 0$, $d^2\phi/dx^2 < 0$, therefore $\phi(x)$ is not convex.

**(e)** Plot these as a function of $x$.

***Answer:***

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib qt

x = np.linspace(-2, 2, num=50)
kappa = 1 # The parameter for the sigmoid loss
```

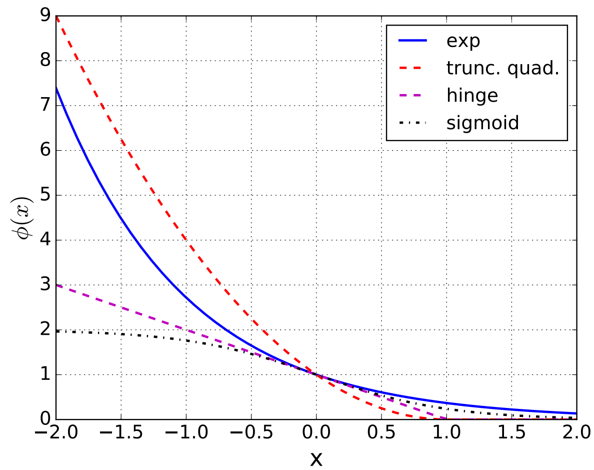


Figure 1: Comparison of different loss functions.

```
axis_font = {'size':'20'}
mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16

plt.plot(x, np.exp(-x), 'b-', linewidth=2, label="exp")
plt.plot(x, (1-x).clip(0) ** 2, 'r--', linewidth=2, label="trunc.␣quad.")
plt.plot(x, (1-x).clip(0), 'm--', linewidth=2, label="hinge")
plt.plot(x, 1 - np.tanh(kappa * x), 'k-.', linewidth=2, label="sigmoid")
plt.legend(prop={'size':16})
plt.grid()
plt.xlabel('x', **axis_font)
plt.ylabel('$\phi(x)$', **axis_font)
```

(This problem is due to notes of Larry Wasserman.)

## 2.

Consider the least-squares problem with $n$ $p$-dimensional covariates, $\{\mathbf{x}_i, y_i\}_{i=1}^n \subset \mathbb{R}^p \times \mathbb{R}$. We would like to fit the following linear model, $\hat{y}(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\beta}$. Also, suppose that there are coefficients $C_+ \subset \{1, \ldots, p\}$ such that for all $j \in C_+$ we require that $\beta_j \geq 0$, $j \in C_+$, and another set $C_- \subset \{1, \ldots, p\}$, such that $\beta_j \leq 0$, $j \in C_-$ (assume that $C_+$ and $C_-$ are non-overlapping). Suppose that $\mathbf{X}^T\mathbf{X}$ is invertible.
Such examples occur in insurance applications: the cost of a given insurance policy is based on a model for the amount of money a customer will cost the company, and each covariate is a variable specific to the customer (such as gender, age, credit history, etc.). It looks bad for the company if the insurance policy is more expensive for a customer that has an older account with the company than a newer account, when everything else is held fixed. Let $x_{i,j} = 1\{\text{customer } i \text{ is has had a policy for more than 2 years}\}$, then $\beta_j \geq 0$ is necessary for this property to hold.

**(a)** Write the constrained optimization for the empirical risk minimization with the constraints.

***Answer:***

$$\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 \tag{5a}$$

$$\text{s.t. } \beta_j \geq 0,\, j \in C_+ \tag{5b}$$

$$\beta_j \leq 0,\, j \in C_- \tag{5c}$$

**(b)** Derive the dual for the optimization as a function of dual parameters.

***Answer:*** The Lagrangian is given by

$$L(\boldsymbol{\beta}, \boldsymbol{\lambda}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \boldsymbol{\beta}^T\boldsymbol{\lambda} \tag{6}$$

where $\lambda_j \geq 0$ for $j \in C_-$, $\lambda_j \leq 0$ for $j \in C_+$ (note for $j \in C_- \cap C_+$ we simply have $\beta_j = 0$ thus $\beta_j$ and the $j$-th column of $\mathbf{X}$ can be removed from the problem) and $\lambda_j = 0$ for $j \notin C_+ \cup C_-$ . Set derivative to 0, we have

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = -\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \boldsymbol{\lambda} = 0 \tag{7}$$

therefore

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{y} - \boldsymbol{\lambda}) \tag{8}$$

substitute back into (7), the dual function is

$$l(\boldsymbol{\lambda}) = \frac{1}{2}\mathbf{y}^T\mathbf{y} - \frac{1}{2}(\mathbf{X}^T\mathbf{y} - \boldsymbol{\lambda})^T(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{y} - \boldsymbol{\lambda}) \tag{9}$$

**(c)** Write the KKT conditions and remark on the implication of the complementary slackness condition.

***Answer:*** The KKT conditions are

$$-\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \boldsymbol{\lambda} = 0; \text{ (stationarity)} \tag{10a}$$

$$\beta_j \geq 0,\, j \in C_+;\; \beta_j \leq 0,\, j \in C_-; \text{ (primal feasibility)} \tag{10b}$$

$$\lambda_j \leq 0,\, j \in C_+;\; \lambda_j \geq 0,\, j \in C_-;\; \lambda_j = 0,\, j \notin C_+ \cap C_-; \text{(dual feasibility)} \tag{10c}$$

$$\lambda_j\beta_j = 0. \text{ (complementary slackness)} \tag{10d}$$

The complementary slackness condition implies that, if $\lambda_j \neq 0$ we must have $\beta_j = 0$, and if $\beta_j \neq 0$ we must have $\lambda_j = 0$.

## 3.

Look at the dataset which can be found here: `https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center`

**(a)** You will predict the final column in the dataset, which is an indicator if the person has made a blood donation. Form three different kernel functions and $n \times n$ kernel matrices of $K_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$ that you think might be appropriate. You may use 2 different built in kernels, but must define one yourself.

***Answer:*** We will use two-built in kernels from scikit-learn, namely the linear kernel and the radial basis function kernel. We will also use the Epanechnikov kernel function

$$K(x, x') = D\left(\frac{|x - x'|}{\lambda}\right) \tag{11}$$

continued on next page... 

Table 1: Comparison of different classifiers.

| Classifier | Optimal parameters | Training score | Test score |
|---|---|---|---|
| SVC+Linear | $C = 10^4$ | 0.7754 | 0.7326 |
| SVC+RBF | $\gamma = 10^{-2}$, $C = 10^2$ | 0.7840 | 0.7433 |
| SVC+Epanechnikov | $l = 10^2$, $C = 10^3$ | 0.7743 | 0.7326 |
| KNN+Linear | $k = 2$ | 0.7786 | 0.7380 |
| KNN+RBF | $k = 2$, $\gamma = 10^{-4}$ | 0.7797 | 0.7380 |
| KNN+Epanechnikov | $k = 2$, $l = 10$ | 0.7797 | 0.7380 |

where

$$D(t) = \begin{cases} \frac{3}{4}(1-t^2) & |t| < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

**(b)** Apply kernel SVMs and $k$-nearest neighbors (where the distance is $d(i,j) = k(\mathbf{x}_i, \mathbf{x}_i) + k(\mathbf{x}_j, \mathbf{x}_j) - 2k(\mathbf{x}_i, \mathbf{x}_j)$) with each of the 3 kernels.

***Answer:*** See next section.

**(c)** Tune any parameters based on what you have learned about validation, and compare these methods with test errors (there are 6 different methods to compare, kNN and SVM with each kernel).

***Answer:*** We split the dataset into a 25% test set and a 75% training+validation set. The parameters of the rbf kernel and the Epanechnikov kernel are tuned based on a Stratified ShuffleSplit cross validation, where the test_size is set to 1/3 of the training+validation dataset (25% of the original dataset) with 5 iterations.
The range of tuning parameters are set as follows. For the support vector classifier, we take $\log C =$ range$(-4, 4)$. For the k-nearest neighbor classifier we take $k =$ range$(1, 15)$. For the rbf kernel, we take $\log \gamma =$ range$(-4, 4)$. For the Epanechnikov kernel, we take bandwidth $\log l =$ range$(-3, 3)$.
The tuning and testing results of the 6 tuned classifiers shown in Table. 1. These results suggest that all the classifiers perform almost equally badly, as a trivial classifier always output 0 would achieve almost the same score over the given dataset.

```
import pandas as pd
import numpy as np
import timeit
import sys

from sklearn import preprocessing, neighbors
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import train_test_split,
    ↪ StratifiedShuffleSplit
from sklearn.grid_search import GridSearchCV
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline

df = pd.read_csv('transfusion.data') # Import the datafile
```

```python
X = preprocessing.scale(df.values[:, :4].astype("float64")) # The predictors, standardized
y = df.values[:, 4] # The responses
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25) # Set aside 25% of the data for test
print("{0} is 0, {1} is 1".format(1 - sum(y) / len(y), sum(y) / len(y)))

cv = StratifiedShuffleSplit(y_train, n_iter=5, test_size=1/3, random_state=42) # Generate the cross validation labels

C_range = np.logspace(-4, 4, 9) # Range of Parameter C for all 3 svcs
k_range = np.arange(1, 15) # Range of parameter n_neighbors for all 3 knncs

gamma_range = np.logspace(-4, 4, 9) #  Range of parameter gamma for rbf kernel
l_range = np.logspace(-3, 3, 7) # Range of parameter l for Epanechnikov kernel

classifiers = dict() # The set of classifiers
params_grid = dict() # The set of cross validation parameters for each classifier

# The svm classifier with linear kernel
classifiers["svc_linear"] = SVC(kernel='linear')
params_grid["svc_linear"] = dict(C=C_range)

# The svm classifier with rbf kernel
classifiers["svc_rbf"] = SVC(kernel='rbf')
params_grid["svc_rbf"] = dict(gamma=gamma_range, C=C_range)

# The svm classifier with user defined Epanechnikov kernel
def epa(X, Y, l=1.0):
    """ Epanechnikov kernel with bandwidth parameter
    return a n_row_X-by-n-row_Y matrix
    """
    n_sample_X = len(X)
    n_sample_Y = len(Y)
    t = np.empty([n_sample_X, n_sample_Y], dtype="float64")
    for i in range(n_sample_X):
        for j in range(n_sample_Y):
            t[i, j] = np.linalg.norm(X[i] - Y[j]) / l

    return 3 / 4  * (1 - t ** 2) * (abs(t) < 1)

class EpaKernel(BaseEstimator, TransformerMixin):
    def __init__(self, l=1.0):
        super(EpaKernel, self).__init__()
        self.l = l
```

```
    def transform(self, X):
        return epa(X, self.X_train_, l=self.l)

    def fit(self, X, y=None, **fit_params):
        self.X_train_ = X
        return self

classifiers["svc_epa"] = Pipeline([('epa', EpaKernel()),('svm', SVC()),])
params_grid["svc_epa"] = dict([('epa__l', l_range), ('svm__kernel', ['
    ↪ precomputed']), ('svm__C', C_range),])

# The knn classifier with linear kernel
classifiers["knnc_linear"] = neighbors.KNeighborsClassifier()
params_grid["knnc_linear"] = dict(n_neighbors=k_range)

# The knn classifier with rbf kernel
def kernel_rbf(x, y, gamma):
    return np.exp(-gamma * np.linalg.norm(x-y)  ** 2)

def dist_rbf(x, y, gamma):
    return kernel_rbf(x, x, gamma) + kernel_rbf(y, y, gamma) - 2 *
        ↪ kernel_rbf(x, y, gamma)

classifiers["knnc_rbf"] = neighbors.KNeighborsClassifier(metric=dist_rbf)
params_grid["knnc_rbf"] = dict(metric_params=[{'gamma': gamma} for gamma
    ↪ in gamma_range], n_neighbors=k_range)

# The knn classifier with user defined Epanechnikov kernel
def kernel_epa(x, y, l):
    t = np.linalg.norm(x - y) / l
    return 3 / 4  * (1 - t ** 2) if abs(t) < 1 else 0.0

def dist_epa(x, y, l):
    return kernel_epa(x, x, l) + kernel_epa(y, y, l) - 2 * kernel_epa(x, y
        ↪ , l)

classifiers["knnc_epa"]  = neighbors.KNeighborsClassifier(metric=dist_epa)
params_grid["knnc_epa"] = dict(metric_params=[{'l': l} for l in l_range],
    ↪ n_neighbors=k_range)

for classifier in classifiers.keys():
    start_time = timeit.default_timer()

    print("Tuning␣{0}...".format(classifier))

    grid = GridSearchCV(classifiers[classifier], param_grid=params_grid[
        ↪ classifier], cv=cv, verbose=0, n_jobs=4)
    grid.fit(X_train, y_train)
```

continued on next page...

```
    print(" - The best parameters are {0} with a score of {1}".format(grid
        ↪ .best_params_, grid.best_score_))
    print(" - The score on the test set is {0}".format(grid.score(X_test,
        ↪ y_test)))
    print(" - Elapsed time is {0}".format(timeit.default_timer() -
        ↪ start_time))
    sys.stdout.flush()
```