ANLY 561 HW

Name:Yuqi Wang
NetID:yw545

Problem 1

Part (a)

In [5]:
```
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer

def chain_rule(Dg, Df, var_shape):
    # Computes the Jacobian D (g o f)
    dim = len(var_shape)
    Dg_axes = list(range(Dg.ndim-dim, Dg.ndim))
    Df_axes = list(range(dim))
    return np.tensordot(Dg, Df, axes=(Dg_axes, Df_axes))

# Compute the Jacobian blocks of X @ W + b

def DX_affine(X, W, b):
    # (d_{x_{i, j}} (X @ W))_{a, b} = e_a^T e_ie_j^T W e_b, so a,i slices equal W.T
    D = np.zeros((X.shape[0], W.shape[1], X.shape[0], X.shape[1]))
    for k in range(X.shape[0]):
        D[k,:,k,:]=W.T
    return D, X.shape

def DW_affine(X, W, b):
    # (d_{w_{i, j}} (X @ W))_{a, b} = e_a^T X e_ie_j^T e_b, so b, j slices equal x
    D = np.zeros((X.shape[0], W.shape[1], W.shape[0], W.shape[1]))
    for k in range(W.shape[1]):
        D[:,k,:,k]=X
    return D, W.shape

def Db_affine(X, W, b):
    # (d_{b_i} (1 @ b))_{a, b} = e_a^T 1 e_i^T e_b, so b, i slices are all ones
    D = np.zeros((X.shape[0], W.shape[1], b.shape[1]))
    for k in range(b.shape[1]):
        D[:,k,k]=1
    return D, b.shape

def logit(z):
    # This is vectorized
    return 1/(1+np.exp(-z))

def Dlogit(Z):
    # The Jacobian of the matrix logit
    D = np.zeros((Z.shape[0], Z.shape[1], Z.shape[0], Z.shape[1]))
    A = logit(Z) * logit(-Z)
    for i in range(Z.shape[0]):
        for j in range(Z.shape[1]):
            D[i,j,i,j] = A[i,j]
    return D, Z.shape
```

```python
def softmax(z):
    v = np.exp(z)
    return v / np.sum(v)

def matrix_softmax(Z):
    return np.apply_along_axis(softmax, 1, Z)

def Dmatrix_softmax(Z):
    D = np.zeros((Z.shape[0], Z.shape[1], Z.shape[0], Z.shape[1]))
    for k in range(Z.shape[0]):
        v = np.exp(Z[k,:])
        v = v / np.sum(v)
        D[k,:,k,:] = np.diag(v) - np.outer(v,v)
        #print(D[k,:,k,:])
    return D, Z.shape

def cross_entropy(P, Q):
    return -np.sum(P * np.log(Q))/P.shape[0]

def DQcross_entropy(P, Q):
    return - P * (1/Q)/P.shape[0], Q.shape

def nn_loss_closure(X, Y):
    # vars[0]=W_1, vars[1]=b_1, vars[2]=W_2, vars[3]=b_2
    # cross_entropy(Y, matrix_softmax(affine(logit(affine(X; W_1, b_1))); W_2, b_2))
    def f(var):
        return cross_entropy(Y, matrix_softmax((logit((logit((X @ var[0]) + var[1]) @  var[2]) + var[3]) @  var[4]) + var[5]))
    return f

def nn_loss_gradient_closure(X, Y):
    def df(var):
        # Activation of first layer
        Z1 = (X @ var[0]) + var[1]
        X2 = logit(Z1)

        # Activation of second layer
        Z2 = (X2 @ var[2]) + var[3]
        X3 = logit(Z2)

        # Activation of third layer
        Z3 = (X3 @ var[4]) + var[5]
        Q = matrix_softmax(Z3)

        # Backpropagation tells us we can immediately contract DQ DZ3
        D_Q, Qshape = DQcross_entropy(Y, Q)
        D_Z3, Z3shape = Dmatrix_softmax(Z3)
```

```python
        back_prop3 = chain_rule(D_Q, D_Z3, Qshape)

        # Jacobians for phi_3

        D_X3, X3shape = DX_affine(X3, var[4], var[5])
        D_W3, W3shape = DW_affine(X3, var[4], var[5])
        D_b3, b3shape = Db_affine(X3, var[4], var[5])


        D_X2, X2shape = DX_affine(X2, var[2], var[3])

        # Jacobian for psi_1\2
        D_Z1, Z1shape = Dlogit(Z1)
        D_Z2, Z2shape = Dlogit(Z2)


        back_prop2 = chain_rule(chain_rule(back_prop3, D_X3, X3shape), D_Z2, Z2shape)
        back_prop1 = chain_rule(chain_rule(back_prop2, D_X2, X2shape), D_Z1, Z1shape)

        # Jacobians for phi_1
        D_W1, W1shape = DW_affine(X, var[0], var[1])
        D_b1, b1shape = Db_affine(X, var[0], var[1])

        # Jacobians for phi_2
        D_W2, W2shape = DW_affine(X2, var[2], var[3])
        D_b2, b2shape = Db_affine(X2, var[2], var[3])

        # Compute all the gradients
        W1grad = chain_rule(back_prop1, D_W1, W1shape)
        b1grad = chain_rule(back_prop1, D_b1, b1shape)
        W2grad = chain_rule(back_prop2, D_W2, W2shape)
        b2grad = chain_rule(back_prop2, D_b2, b2shape)
        W3grad = chain_rule(back_prop3, D_W3, W3shape)
        b3grad = chain_rule(back_prop3, D_b3, b3shape)

        return [W1grad, b1grad, W2grad, b2grad, W3grad, b3grad]
    return df

def update_blocks(x,y,t):
    # An auxiliary function for backtracking with blocks of variables
    num_blocks = len(x)
    z = [None]*num_blocks
    for i in range(num_blocks):
        z[i] = x[i] + t*y[i]
    return z

def block_backtracking(x0, f, dx, df0, alpha=0.1, beta=0.5, verbose=False):
```

```python
    num_blocks = len(x0)

    delta = 0
    for i in range(num_blocks):
        delta = delta + np.sum(dx[i] * df0[i])
    delta = alpha * delta

    f0 = f(x0)

    t = 1
    x = update_blocks(x0, dx, t)
    fx = f(x)
    while (not np.isfinite(fx)) or f0+t*delta<fx:
        t = beta*t
        x = update_blocks(x0, dx, t)
        fx = f(x)

    if verbose:
        print((t, delta))
        l=-1e-5
        u=1e-5
        s = np.linspace(l, u, 64)
        fs = np.zeros(s.size)
        crit = f0 + s*delta
        tan = f0 + s*delta/alpha
        for i in range(s.size):
            fs[i] = f(update_blocks(x0, dx, s[i]))
        plt.plot(s, fs)
        plt.plot(s, crit, '--')
        plt.plot(s, tan, '.')
        plt.scatter([0], [f0])
        plt.show()

    return x, fx

def negate_blocks(x):
    # Helper function for negating the gradient of block variables
    num_blocks = len(x)
    z = [None]*num_blocks
    for i in range(num_blocks):
        z[i] = -x[i]
    return z

def block_norm(x):
    num_blocks=len(x)
    z = 0
    for i in range(num_blocks):
```

```
        z = z + np.sum(x[i]**2)
    return np.sqrt(z)

def random_matrix(shape, sigma=0.1):
    # Helper for random initialization
    return np.reshape(sigma*rd.randn(shape[0]*shape[1]), shape)

### Begin gradient descent example

### Random seed
rd.seed(1234)

data = load_breast_cancer() # Loads the Wisconsin Breast Cancer dataset (569 examples in 30 dimensions)

# Parameters for the data
dim_data = 30
num_labels = 2
num_examples = 569

# Parameters for training
num_train = 400

X = data['data'] # Data in rows
targets = data.target # 0-1 labels
labels = np.zeros((num_examples, num_labels))
for i in range(num_examples):
    labels[i,targets[i]]=1 # Conversion to one-hot representations

# Prepare hyperparameters of the network
hidden_nodes = 20

# Initialize variables
W1_init = random_matrix((dim_data, hidden_nodes))
b1_init = np.zeros((1, hidden_nodes))

W12_init = random_matrix((hidden_nodes, hidden_nodes))
b12_init = np.zeros((1, hidden_nodes))

W2_init = random_matrix((hidden_nodes, num_labels))
b2_init = np.zeros((1, num_labels))

x = [W1_init, b1_init, W12_init, b12_init, W2_init, b2_init]
f = nn_loss_closure(X[:num_train,:], labels[:num_train,:])
df = nn_loss_gradient_closure(X[:num_train,:], labels[:num_train,:])
dx = lambda v: negate_blocks(df(v))

for i in range(100):
```

```python
    ngrad = dx(x)
    x, fval = block_backtracking(x, f, ngrad, df(x), alpha=0.1, verbose=False)

    train_data = matrix_softmax(logit(logit(X[:num_train,:]@x[0] + x[1])@x[2] +x[3]) @ x[4] + x[5])
    train_labels = np.argmax(train_data, axis=1)
    per_correct = 100*(1 - np.count_nonzero(train_labels - targets[:num_train])/num_train)

    if i % 2 == 0:
        print("Step: %d, Avg Cross Entropy: %f, Gradient Norm: %f, Training Accuracy: %.1f percent" % (i,fval,block_norm(ngrad), per_correc

test_data = matrix_softmax(logit(logit(X[num_train:,:]@x[0] + x[1])@x[2] +x[3]) @ x[4] + x[5])
test_labels = np.argmax(test_data, axis=1)
per_correct = 100*(1 - np.count_nonzero(test_labels - targets[num_train:])/(num_examples-num_train))

print('Final test accuracy: %.1f percent' % per_correct)
```

```
C:\Users\45336\Anaconda3\lib\site-packages\ipykernel_launcher.py:38: RuntimeWarning: overflow encountered in exp

Step: 0, Avg Cross Entropy: 0.684698, Gradient Norm: 0.573653, Training Accuracy: 56.8 percent
Step: 2, Avg Cross Entropy: 0.684045, Gradient Norm: 0.038587, Training Accuracy: 56.8 percent
Step: 4, Avg Cross Entropy: 0.683980, Gradient Norm: 0.013244, Training Accuracy: 56.8 percent
Step: 6, Avg Cross Entropy: 0.683967, Gradient Norm: 0.004895, Training Accuracy: 56.8 percent
Step: 8, Avg Cross Entropy: 0.683967, Gradient Norm: 0.000959, Training Accuracy: 56.8 percent
Step: 10, Avg Cross Entropy: 0.683966, Gradient Norm: 0.001043, Training Accuracy: 56.8 percent
Step: 12, Avg Cross Entropy: 0.683965, Gradient Norm: 0.000915, Training Accuracy: 56.8 percent
Step: 14, Avg Cross Entropy: 0.683965, Gradient Norm: 0.001049, Training Accuracy: 56.8 percent
Step: 16, Avg Cross Entropy: 0.683964, Gradient Norm: 0.000924, Training Accuracy: 56.8 percent
Step: 18, Avg Cross Entropy: 0.683964, Gradient Norm: 0.001086, Training Accuracy: 56.8 percent
Step: 20, Avg Cross Entropy: 0.683963, Gradient Norm: 0.000943, Training Accuracy: 56.8 percent
Step: 22, Avg Cross Entropy: 0.683963, Gradient Norm: 0.002028, Training Accuracy: 56.8 percent
Step: 24, Avg Cross Entropy: 0.683962, Gradient Norm: 0.001797, Training Accuracy: 56.8 percent
Step: 26, Avg Cross Entropy: 0.683962, Gradient Norm: 0.001586, Training Accuracy: 56.8 percent
Step: 28, Avg Cross Entropy: 0.683961, Gradient Norm: 0.001398, Training Accuracy: 56.8 percent
Step: 30, Avg Cross Entropy: 0.683961, Gradient Norm: 0.001235, Training Accuracy: 56.8 percent
Step: 32, Avg Cross Entropy: 0.683960, Gradient Norm: 0.001011, Training Accuracy: 56.8 percent
Step: 34, Avg Cross Entropy: 0.683960, Gradient Norm: 0.000896, Training Accuracy: 56.8 percent
Step: 36, Avg Cross Entropy: 0.683959, Gradient Norm: 0.001099, Training Accuracy: 56.8 percent
Step: 38, Avg Cross Entropy: 0.683959, Gradient Norm: 0.000935, Training Accuracy: 56.8 percent
Step: 40, Avg Cross Entropy: 0.683958, Gradient Norm: 0.000846, Training Accuracy: 56.8 percent
Step: 42, Avg Cross Entropy: 0.683958, Gradient Norm: 0.001008, Training Accuracy: 56.8 percent
Step: 44, Avg Cross Entropy: 0.683957, Gradient Norm: 0.000876, Training Accuracy: 56.8 percent
Step: 46, Avg Cross Entropy: 0.683957, Gradient Norm: 0.001062, Training Accuracy: 56.8 percent
Step: 48, Avg Cross Entropy: 0.683956, Gradient Norm: 0.000893, Training Accuracy: 56.8 percent
Step: 50, Avg Cross Entropy: 0.683956, Gradient Norm: 0.001086, Training Accuracy: 56.8 percent
Step: 52, Avg Cross Entropy: 0.683955, Gradient Norm: 0.000893, Training Accuracy: 56.8 percent
Step: 54, Avg Cross Entropy: 0.683955, Gradient Norm: 0.001743, Training Accuracy: 56.8 percent
```

```
Step: 56, Avg Cross Entropy: 0.683954, Gradient Norm: 0.001269, Training Accuracy: 56.8 percent
Step: 58, Avg Cross Entropy: 0.683954, Gradient Norm: 0.001875, Training Accuracy: 56.8 percent
Step: 60, Avg Cross Entropy: 0.683953, Gradient Norm: 0.001297, Training Accuracy: 56.8 percent
Step: 62, Avg Cross Entropy: 0.683953, Gradient Norm: 0.001846, Training Accuracy: 56.8 percent
Step: 64, Avg Cross Entropy: 0.683952, Gradient Norm: 0.001231, Training Accuracy: 56.8 percent
Step: 66, Avg Cross Entropy: 0.683952, Gradient Norm: 0.000910, Training Accuracy: 56.8 percent
Step: 68, Avg Cross Entropy: 0.683951, Gradient Norm: 0.001729, Training Accuracy: 56.8 percent
Step: 70, Avg Cross Entropy: 0.683951, Gradient Norm: 0.001118, Training Accuracy: 56.8 percent
Step: 72, Avg Cross Entropy: 0.683950, Gradient Norm: 0.000848, Training Accuracy: 56.8 percent
Step: 74, Avg Cross Entropy: 0.683950, Gradient Norm: 0.000946, Training Accuracy: 56.8 percent
Step: 76, Avg Cross Entropy: 0.683949, Gradient Norm: 0.001795, Training Accuracy: 56.8 percent
Step: 78, Avg Cross Entropy: 0.683949, Gradient Norm: 0.002188, Training Accuracy: 56.8 percent
Step: 80, Avg Cross Entropy: 0.683948, Gradient Norm: 0.001184, Training Accuracy: 56.8 percent
Step: 82, Avg Cross Entropy: 0.683948, Gradient Norm: 0.000836, Training Accuracy: 56.8 percent
Step: 84, Avg Cross Entropy: 0.683947, Gradient Norm: 0.000895, Training Accuracy: 56.8 percent
Step: 86, Avg Cross Entropy: 0.683947, Gradient Norm: 0.001522, Training Accuracy: 56.8 percent
Step: 88, Avg Cross Entropy: 0.683946, Gradient Norm: 0.001614, Training Accuracy: 56.8 percent
Step: 90, Avg Cross Entropy: 0.683946, Gradient Norm: 0.001647, Training Accuracy: 56.8 percent
Step: 92, Avg Cross Entropy: 0.683945, Gradient Norm: 0.001619, Training Accuracy: 56.8 percent
Step: 94, Avg Cross Entropy: 0.683944, Gradient Norm: 0.001536, Training Accuracy: 56.8 percent
Step: 96, Avg Cross Entropy: 0.683944, Gradient Norm: 0.001414, Training Accuracy: 56.8 percent
Step: 98, Avg Cross Entropy: 0.683943, Gradient Norm: 0.001273, Training Accuracy: 56.8 percent
Final test accuracy: 76.9 percent
```

After about 100 steps, the final test accuracy is around 76.9%.

Part (b)

Three ways to make this implementation more efficient:

1. The Jacobian matrix is very sparse, so we may find a package about sparse matrix and then use the package to store the Jacobian in Python instead of storing as a full matrix. For example, we can find a package which allows us to only store data positions where the data entry is nonzero.
2. Besides, we can write an outer function for the affine function in order to reduce the number of indices which chain rule requires summation over.
3. In addition, we can try other methods. In stead of Gradient Descent, we can use Stochastic Gradient Descent in order to try to jump out of the local minimum.
4. Finally, we may reduce the size of training data to make this implementation more efficient.

Problem 2

Part (a)

We have

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{pmatrix}$$

We set

$$Y = \begin{pmatrix} y_{1,1} & y_{1,2} \\ y_{2,1} & y_{2,2} \end{pmatrix} = X \star \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = c_{(i,j)}(X, \mathcal{T})_{\mathbf{k}\setminus\{i\}\oplus\mathbf{l}\setminus\{j\}}$$

where $\mathcal{T}$ is a 2 by 2 by 3 by 3 tensor. As a form of constraction, we have

$$y_{1,1} = \sum_{i=1}^{3}\sum_{j=1}^{3} x_{i,j}\mathcal{T}_{1,2,i,j} = x_{1,1} + x_{2,2}$$

So

$$\mathcal{T}_{1,1,\cdot,\cdot} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Similarly, we can get

$$\mathcal{T}_{1,2,\cdot,\cdot} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\mathcal{T}_{2,1,\cdot,\cdot} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\mathcal{T}_{2,2,\cdot,\cdot} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Thus,

$$\mathcal{T} = \left(\left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}\right), \left(\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\right)\right)$$

And contraction should occur between X and $\mathcal{T}$ along $i = \{1, 2\}, j = \{3, 4\}$.

Part (b)

We have

$$X = \left(\begin{pmatrix} x_{1,1,1} & x_{1,1,2} & x_{1,1,3} \\ x_{1,2,1} & x_{1,2,2} & x_{1,2,3} \\ x_{1,3,1} & x_{1,3,2} & x_{1,3,3} \end{pmatrix}, \begin{pmatrix} x_{2,1,1} & x_{2,1,2} & x_{2,1,3} \\ x_{2,2,1} & x_{2,2,2} & x_{2,2,3} \\ x_{2,3,1} & x_{2,3,2} & x_{2,3,3} \end{pmatrix} \right)$$

We set

$$Y = \begin{pmatrix} y_{1,1} & y_{1,2} \\ y_{2,1} & y_{2,2} \end{pmatrix} = X \star \mathcal{H} = X \star \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \right) = c_{(i,j)}(X, \mathcal{T})_{\mathbf{k}\setminus\{i\}\oplus\mathbf{l}\setminus\{j\}}$$

where $\mathcal{T}$ is a 2 by 2 by 2 by 3 by 3 tensor. As a form of constraction, we have

$$y_{1,1} = \sum_{i=1}^{2} \sum_{j=1}^{3} \sum_{k=1}^{3} x_{i,j,k} \mathcal{T}_{1,1,i,j,k} = x_{1,1,1} + x_{1,2,2} + x_{2,1,1} + x_{2,1,2} + x_{2,1,3} + x_{2,2,1} + x_{2,2,2}$$

So

$$\mathcal{T}_{1,1,\cdot,\cdot,\cdot} = \left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right)$$

Similarly, we can get

$$\mathcal{T}_{1,2,\cdot,\cdot,\cdot} = \left(\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \right)$$

$$\mathcal{T}_{2,1,\cdot,\cdot,\cdot} = \left(\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \right)$$

$$\mathcal{T}_{2,1,\cdot,\cdot,\cdot} = \left(\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \right)$$

Thus,

$$\mathcal{T} = \left(\left(\left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right), \left(\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \right) \right), \left(\left(\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \right), \left(\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \right) \right) \right)$$

And contraction should occur between X and $\mathcal{T}$ along $i = \{1, 2, 3\}, j = \{3, 4, 5\}$.

Problem 3

Weather Based Merchandise Inventory Optimization

Outline:

Title: Using machine learning to predict demand for weather-sensitive products at Walmart Stores

Thesis: This paper is going to discuss how major weather events affect the sales of potentially weather-sensitive products at different Walmart stores. And in order to optimize prediction accuracy of the sales volume, several supervised and unsupervised machine learning methods were applied in this study, including regression, different models to a historical datasets from Walmart to find the inner pattern for products sales giving certain weather condition. A portion of original data sets will be randomly selected for each store as the test data set before building our model to evaluate the results. Based on this inner pattern, we hope to help the market to optimize its sales for weather-sensitive products by offering strategies such as product-bundling.

I. Introduction

A. Background introduction and overview

B. The previous researches of prediction for product sales based on weather conditions.

II. Exploratory Data Analysis

A. Describe the details and selections of attributes in our data used in this project.

B. Analyze the data - do some basic statistical analyses.

C. Clean the raw datasets - remove all zero entries.

D. Transformed or manipulated the datasets for further analysis - merge three datasets based on Date.

III. Approaches used to analyze the datasets

A. SVM

1. Introduction to SVM
2. How to use SVM on these datasets
3. Advantage and disadvantages

B. Decision tree

1. Introduction to Decision tree
2. How to use Decision tree on these datasets
3. Advantage and disadvantages

C. Neural Network

1. Introduction to Neural Network
2. How to use Neural Network on these datasets
3. Advantage and disadvantages

D. Regression

1. Introduction to Regression
2. How to use Regression on these datasets
3. Advantage and disadvantages

IV. Prediction analysis

A. Explain how to use the Root Mean Squared Logarithmic Error (RMSLE) to test prediction accuracy

B. Showing results for different methods

1. SVM
2. Decision tree
3. Neural Network
4. Regression

V. Discussion

Discuss the results.

VI. References

VII. Appendix

Visualizations