ECE 50024: Homework 2

Manish Kumar Krishne Gowda, 0033682812 (Spring 2023)

**Exercise 1: Loading Data via Python**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt**  **import** **cvxpy** **as** **cp**  **import** **csv**  male\_rows = []  female\_rows = []  # Reading csv file for male data  **with** open("data/male\_train\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  male\_rows.append(list(np.float\_(row)))  male\_rows = np.array(male\_rows)  male\_rows[:,**1**] = np.divide(male\_rows[:,**1**],**10**)  male\_rows[:,**2**] = np.divide(male\_rows[:,**2**],**1000**)  **print**("male\_bmi : " + str(male\_rows[**0**:**10**,**1**]))  **print**("male\_stature\_m : " + str(male\_rows[**0**:**10**,**2**]))  csv\_file.close()  # Reading csv file for female data  **with** open("data/female\_train\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  female\_rows.append(list(np.float\_(row)))  female\_rows = np.array(female\_rows)  female\_rows[:,**1**] = np.divide(female\_rows[:,**1**],**10**)  female\_rows[:,**2**] = np.divide(female\_rows[:,**2**],**1000**)  **print**("female\_bmi : " + str(female\_rows[**0**:**10**,**1**]))  **print**("female\_stature\_m : " + str(female\_rows[**0**:**10**,**2**]))  csv\_file.close() |

**OUTPUT:**

**male\_bmi : [3. 2.56 2.42 2.74 2.59 2.53 2.27 2.54 3.41 3.34]**

**male\_stature\_m : [1.679 1.586 1.773 1.816 1.809 1.662 1.829 1.686 1.761 1.797]**

**female\_bmi : [2.82 2.22 2.71 2.81 2.55 2.3 3.56 3.11 2.46 4.3 ]**

**female\_stature\_m : [1.563 1.716 1.484 1.651 1.548 1.665 1.564 1.676 1.69 1.704]**

**===================================================================================**

**Exercise 2: Build a Linear Classifier via Optimization**

**(a)**

∇θEtrain(θ) = ∇θ ∥y − Xθ∥2 = −2XT (y − Xθ)

Equating this to zero, we obtain XT (y − Xθ) = 0

* θ(^) = (XTX)-1XT y

θ(^) is a unique solution if XTX is invertible, which will happen only if the columns of X are linearly independent. This means X ∈ RN×d has full rank d.

XTX is not invertible then Ridge regression and Lasso regression methods can be used.

**(b) & (c)**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90 | **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt**  **import** **cvxpy** **as** **cp**  **import** **csv**  male\_rows = []  female\_rows = []  male\_test\_rows = []  female\_test\_rows = []  # Reading csv files for male data  **with** open("data/male\_train\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  male\_rows.append(list(np.float\_(row)))  male\_rows = np.array(male\_rows)  male\_rows[:,**1**] = np.divide(male\_rows[:,**1**],**10**)  male\_rows[:,**2**] = np.divide(male\_rows[:,**2**],**1000**)  csv\_file.close()  **with** open("data/male\_test\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  male\_test\_rows.append(list(np.float\_(row)))  male\_test\_rows = np.array(male\_test\_rows)  male\_test\_rows[:,**1**] = np.divide(male\_test\_rows[:,**1**],**10**)  male\_test\_rows[:,**2**] = np.divide(male\_test\_rows[:,**2**],**1000**)  csv\_file.close()  # Reading csv files for female data  **with** open("data/female\_train\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  female\_rows.append(list(np.float\_(row)))  female\_rows = np.array(female\_rows)  female\_rows[:,**1**] = np.divide(female\_rows[:,**1**],**10**)  female\_rows[:,**2**] = np.divide(female\_rows[:,**2**],**1000**)  csv\_file.close()  **with** open("data/female\_test\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  female\_test\_rows.append(list(np.float\_(row)))  female\_test\_rows = np.array(female\_test\_rows)  female\_test\_rows[:,**1**] = np.divide(female\_test\_rows[:,**1**],**10**)  female\_test\_rows[:,**2**] = np.divide(female\_test\_rows[:,**2**],**1000**)  csv\_file.close()  #least squares  N = male\_rows.shape[**0**] + female\_rows.shape[**0**]  d = **3**  x = np.vstack((male\_rows[:,**1**:**3**],female\_rows[:,**1**:**3**]))  X = np.column\_stack((x, np.ones(N))) #consider the basis function as 1 + x1 + x2  y = np.vstack((np.ones((male\_rows.shape[**0**],**1**)),-**1**\*np.ones((female\_rows.shape[**0**],**1**))))  XtX = np.dot(X.T, X)  Xty = np.dot(X.T, y)  theta\_cap = np.dot(np.linalg.pinv(XtX), Xty )  **print**(f"theta\_cap by analytic method : {theta\_cap}")  #male test  N\_test\_male = male\_test\_rows.shape[**0**]  x\_male = male\_test\_rows[:,**1**:**3**]  X\_male\_test = np.column\_stack((x\_male, np.ones(N\_test\_male))) #consider the basis function as 1 + x1 + x2  y\_male\_test = X\_male\_test**@theta\_cap**  y\_male\_test = np.sign(y\_male\_test)  correct\_prediction = y\_male\_test[np.where(y\_male\_test==**1**)]  **print**(f"predicted {correct\_prediction.shape[0]} correctly out of {y\_male\_test.shape[0]-1} male samples")  #female test  N\_test\_female = female\_test\_rows.shape[**0**]  x\_female = female\_test\_rows[:,**1**:**3**]  X\_female\_test = np.column\_stack((x\_female, np.ones(N\_test\_female))) #consider the basis function as 1 + x1 + x2  y\_female\_test = X\_female\_test**@theta\_cap**  y\_female\_test = np.sign(y\_female\_test)  correct\_prediction = y\_female\_test[np.where(y\_female\_test==-**1**)]  **print**(f"predicted {correct\_prediction.shape[0]} correctly out of {y\_female\_test.shape[0]-1} female samples")  # ===== CVX method ==========  theta = cp.Variable(d)  y\_cvx = cp.reshape(y, (y.shape[**0**],))  cost = cp.Minimize(cp.sum\_squares(X**@theta**-y\_cvx))  prob = cp.Problem(cost)  prob.solve()  theta\_cap\_cvx = theta.value  **print**(f"theta\_cap by cvx method : {theta\_cap\_cvx}") |

**OUTPUT**

**theta\_cap by analytic method : [[ -0.12339677]**

**[ 6.67486843]**

**[-10.7017505 ]]**

**predicted 411 correctly out of 500 male samples**

**predicted 430 correctly out of 500 female samples**

**theta\_cap by cvx method : [ -0.12339677 6.67486843 -10.7017505 ]**

**d**

**A piece of paper with writing

Description automatically generated with low confidence**

**e,f,g &h**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107 | **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt**  **import** **cvxpy** **as** **cp**  **import** **csv**  male\_rows = []  female\_rows = []  male\_test\_rows = []  female\_test\_rows = []  # Reading csv files for male data  **with** open("data/male\_train\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  male\_rows.append(list(np.float\_(row)))  male\_rows = np.array(male\_rows)  male\_rows[:,**1**] = np.divide(male\_rows[:,**1**],**10**)  male\_rows[:,**2**] = np.divide(male\_rows[:,**2**],**1000**)  csv\_file.close()  **with** open("data/male\_test\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  male\_test\_rows.append(list(np.float\_(row)))  male\_test\_rows = np.array(male\_test\_rows)  male\_test\_rows[:,**1**] = np.divide(male\_test\_rows[:,**1**],**10**)  male\_test\_rows[:,**2**] = np.divide(male\_test\_rows[:,**2**],**1000**)  csv\_file.close()  # Reading csv files for female data  **with** open("data/female\_train\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  female\_rows.append(list(np.float\_(row)))  female\_rows = np.array(female\_rows)  female\_rows[:,**1**] = np.divide(female\_rows[:,**1**],**10**)  female\_rows[:,**2**] = np.divide(female\_rows[:,**2**],**1000**)  csv\_file.close()  **with** open("data/female\_test\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  female\_test\_rows.append(list(np.float\_(row)))  female\_test\_rows = np.array(female\_test\_rows)  female\_test\_rows[:,**1**] = np.divide(female\_test\_rows[:,**1**],**10**)  female\_test\_rows[:,**2**] = np.divide(female\_test\_rows[:,**2**],**1000**)  csv\_file.close()  **def** **delta\_f**(theta\_var):  XtX = np.dot(X.T, X)  Xty = np.dot(X.T, y)  **return** (-**2**\*Xty + **2**\*XtX**@theta\_var**)  **def** **iter**(theta\_var):  XtX = np.dot(X.T, X)  del\_f = delta\_f(theta\_var)  numerator = del\_f.T**@del\_f**  denominator = **2**\*del\_f.T**@XtX@del\_f**  **return** numerator/denominator  N = male\_rows.shape[**0**] + female\_rows.shape[**0**]  d = **3**  x = np.vstack((male\_rows[:,**1**:**3**],female\_rows[:,**1**:**3**]))  X = np.column\_stack((x, np.ones(N))) #consider the basis function as 1 + x1 + x2  y = np.vstack((np.ones((male\_rows.shape[**0**],**1**)),-**1**\*np.ones((female\_rows.shape[**0**],**1**))))  theta\_not = np.zeros((d,**1**))  N\_test= male\_test\_rows.shape[**0**] + female\_test\_rows.shape[**0**]  x\_test = np.vstack((male\_test\_rows[:,**1**:**3**],female\_test\_rows[:,**1**:**3**]))  X\_test = np.column\_stack((x\_test, np.ones(N\_test))) #consider the basis function as 1 + x1 + x2  y\_test = np.vstack((np.ones((male\_test\_rows.shape[**0**],**1**)),-**1**\*np.ones((female\_test\_rows.shape[**0**],**1**))))  theta = theta\_not  sq\_err = []  x\_axis = np.arange(**0**,**50000**)  **for** k **in** x\_axis:  theta = theta - iter(theta)\*delta\_f(theta)  sq\_err.append(np.sum((y\_test - (X\_test**@theta**))\*\***2**))  **print**(theta)  plt.semilogx(x\_axis, sq\_err, linewidth=**8**)  plt.grid(True)  plt.show()  #momentum method  sq\_err\_momentum = []  thetak = theta\_not  thetak\_minus1 = theta\_not  beta = **0.9**  **for** k **in** x\_axis:  curr\_iter = iter(thetak)  thetak\_plus1 = thetak - beta\*curr\_iter\*delta\_f(thetak\_minus1) - (**1**-beta)\*curr\_iter\*delta\_f(thetak)  thetak\_minus1 = thetak  thetak = thetak\_plus1  sq\_err\_momentum.append(np.sum((y\_test - (X\_test**@thetak\_plus1**))\*\***2**))  **print**(sq\_err\_momentum)  **print**(thetak\_plus1)  plt.semilogx(x\_axis, sq\_err\_momentum, linewidth=**8**)  plt.grid(True)  plt.show()  **Chart, line chart  Description automatically generated**  alphak\_with\_grad\_descent  **Chart  Description automatically generated**  alphak\_with\_momentum |

**Note :**

1. **In momentum method error was found to oscillate with peak around 500th iteration and the error stabilizes at 523.6**
2. **θ(^) in all the methods was found to be**

**[[ -0.12339686]**

**[ 6.67486611]  
 [-10.70174642]]**

**Exercise 3: Visualization and Testing**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **cvxpy** **as** **cp**

**import** **csv**

male\_rows = []

female\_rows = []

male\_test\_rows = []

female\_test\_rows = []

# Reading csv files for male data

**with** open("data/male\_train\_data.csv", "r") **as** csv\_file:

reader = csv.reader(csv\_file, delimiter=',')

fields = next(reader)

**for** row **in** reader:

male\_rows.append(list(np.float\_(row)))

male\_rows = np.array(male\_rows)

male\_rows[:,**1**] = np.divide(male\_rows[:,**1**],**10**)

male\_rows[:,**2**] = np.divide(male\_rows[:,**2**],**1000**)

csv\_file.close()

**with** open("data/male\_test\_data.csv", "r") **as** csv\_file:

reader = csv.reader(csv\_file, delimiter=',')

fields = next(reader)

**for** row **in** reader:

male\_test\_rows.append(list(np.float\_(row)))

male\_test\_rows = np.array(male\_test\_rows)

male\_test\_rows[:,**1**] = np.divide(male\_test\_rows[:,**1**],**10**)

male\_test\_rows[:,**2**] = np.divide(male\_test\_rows[:,**2**],**1000**)

csv\_file.close()

# Reading csv files for female data

**with** open("data/female\_train\_data.csv", "r") **as** csv\_file:

reader = csv.reader(csv\_file, delimiter=',')

fields = next(reader)

**for** row **in** reader:

female\_rows.append(list(np.float\_(row)))

female\_rows = np.array(female\_rows)

female\_rows[:,**1**] = np.divide(female\_rows[:,**1**],**10**)

female\_rows[:,**2**] = np.divide(female\_rows[:,**2**],**1000**)

csv\_file.close()

**with** open("data/female\_test\_data.csv", "r") **as** csv\_file:

reader = csv.reader(csv\_file, delimiter=',')

fields = next(reader)

**for** row **in** reader:

female\_test\_rows.append(list(np.float\_(row)))

female\_test\_rows = np.array(female\_test\_rows)

female\_test\_rows[:,**1**] = np.divide(female\_test\_rows[:,**1**],**10**)

female\_test\_rows[:,**2**] = np.divide(female\_test\_rows[:,**2**],**1000**)

csv\_file.close()

# a) visualize the classifier.

N = male\_rows.shape[**0**] + female\_rows.shape[**0**]

d = **3**

x = np.vstack((male\_rows[:,**1**:**3**],female\_rows[:,**1**:**3**]))

X = np.column\_stack((x, np.ones(N))) #consider the basis function as 1 + x1 + x2

y = np.vstack((np.ones((male\_rows.shape[**0**],**1**)),-**1**\*np.ones((female\_rows.shape[**0**],**1**))))

XtX = np.dot(X.T, X)

Xty = np.dot(X.T, y)

theta\_cap = np.dot(np.linalg.pinv(XtX), Xty )

**print**(f"theta\_cap by analytic method : {theta\_cap}")

plt.scatter(male\_rows[:,**1**], male\_rows[:,**2**], edgecolor ="blue", marker ="o")

plt.scatter(female\_rows[:,**1**], female\_rows[:,**2**], c ="red", marker =".")

line\_x = np.linspace(**0**, **9**, **1000**)

line\_y = -theta\_cap[**2**] / theta\_cap[**1**] - theta\_cap[**0**] / theta\_cap[**1**] \* line\_x

plt.scatter(line\_x,line\_y, c ="black", linewidths=**0.1**)

plt.show()

# b) classification accuracy

N\_test\_male = male\_test\_rows.shape[**0**]

x\_male = male\_test\_rows[:,**1**:**3**]

X\_male\_test = np.column\_stack((x\_male, np.ones(N\_test\_male))) #consider the basis function as 1 + x1 + x2

y\_male\_test = X\_male\_test**@theta\_cap**

y\_male\_test = np.sign(y\_male\_test)

wrong\_prediction\_male = y\_male\_test[np.where(y\_male\_test==-**1**)]

correct\_prediction\_male = y\_male\_test[np.where(y\_male\_test==**1**)]

wrong\_prediction\_male\_pcnt = **100**\*wrong\_prediction\_male.shape[**0**]/(y\_male\_test.shape[**0**]-**1**)

**print**(f"Type 2 error = {wrong\_prediction\_male\_pcnt}%")

#female test

N\_test\_female = female\_test\_rows.shape[**0**]

x\_female = female\_test\_rows[:,**1**:**3**]

X\_female\_test = np.column\_stack((x\_female, np.ones(N\_test\_female))) #consider the basis function as 1 + x1 + x2

y\_female\_test = X\_female\_test**@theta\_cap**

y\_female\_test = np.sign(y\_female\_test)

wrong\_prediction\_female = y\_female\_test[np.where(y\_female\_test==**1**)]

correct\_prediction\_female = y\_female\_test[np.where(y\_female\_test==-**1**)]

wrong\_prediction\_female\_pcnt = **100**\*wrong\_prediction\_female.shape[**0**]/(y\_female\_test.shape[**0**]-**1**)

**print**(f"Type 1 error = {wrong\_prediction\_female\_pcnt}%")

TruePositive = correct\_prediction\_male.shape[**0**]

FalsePositive = wrong\_prediction\_female.shape[**0**]

FalseNegative = wrong\_prediction\_male.shape[**0**]

Precision = TruePositive/(TruePositive+FalsePositive)

Recall = TruePositive/(TruePositive+FalseNegative)

**print**(f"Precision = {Precision}")

**print**(f"Recall={Recall}")

**Chart, scatter chart

Description automatically generated**

Classifier Visualiser

**Output**

Type 2 error = 18.0%

Type 1 error = 14.2%

Precision = 0.8526970954356846

Recall=0.8203592814371258

**Exercise 4: Regularization**

a)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt**  **import** **cvxpy** **as** **cp**  **import** **csv**  male\_rows = []  female\_rows = []  male\_test\_rows = []  female\_test\_rows = []  # Reading csv files for male data  **with** open("data/male\_train\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  male\_rows.append(list(np.float\_(row)))  male\_rows = np.array(male\_rows)  male\_rows[:,**1**] = np.divide(male\_rows[:,**1**],**10**)  male\_rows[:,**2**] = np.divide(male\_rows[:,**2**],**1000**)  csv\_file.close()  **with** open("data/male\_test\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  male\_test\_rows.append(list(np.float\_(row)))  male\_test\_rows = np.array(male\_test\_rows)  male\_test\_rows[:,**1**] = np.divide(male\_test\_rows[:,**1**],**10**)  male\_test\_rows[:,**2**] = np.divide(male\_test\_rows[:,**2**],**1000**)  csv\_file.close()  # Reading csv files for female data  **with** open("data/female\_train\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  female\_rows.append(list(np.float\_(row)))  female\_rows = np.array(female\_rows)  female\_rows[:,**1**] = np.divide(female\_rows[:,**1**],**10**)  female\_rows[:,**2**] = np.divide(female\_rows[:,**2**],**1000**)  csv\_file.close()  **with** open("data/female\_test\_data.csv", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=',')  fields = next(reader)  **for** row **in** reader:  female\_test\_rows.append(list(np.float\_(row)))  female\_test\_rows = np.array(female\_test\_rows)  female\_test\_rows[:,**1**] = np.divide(female\_test\_rows[:,**1**],**10**)  female\_test\_rows[:,**2**] = np.divide(female\_test\_rows[:,**2**],**1000**)  csv\_file.close()  N = male\_rows.shape[**0**] + female\_rows.shape[**0**]  d = **3**  x = np.vstack((male\_rows[:,**1**:**3**],female\_rows[:,**1**:**3**]))  X = np.column\_stack((x, np.ones(N))) #consider the basis function as 1 + x1 + x2  y = np.vstack((np.ones((male\_rows.shape[**0**],**1**)),-**1**\*np.ones((female\_rows.shape[**0**],**1**))))  lambd = np.arange(**0.1**, **10**, **0.1**)  theta\_lambd\_norm = np.zeros(len(lambd))  mse = np.zeros(len(lambd))  **for** idx, lam **in** enumerate(lambd):  theta\_lambd = cp.Variable((d,**1**))  objective\_lambd = cp.Minimize(cp.sum\_squares(X**@theta\_lambd** - y) + lam\*cp.sum\_squares(theta\_lambd))  prob = cp.Problem(objective\_lambd)  prob.solve()  theta\_solution = theta\_lambd.value  theta\_lambd\_norm[idx] = (np.linalg.norm(theta\_solution))\*\***2**  mse[idx] = (np.linalg.norm(np.matmul(X, theta\_solution)-y))\*\***2**  plt.plot(theta\_lambd\_norm, mse, linewidth=**8**)  plt.grid(True)  plt.show()  plt.plot(lambd, mse, linewidth=**8**)  plt.grid(True)  plt.show()  plt.plot(lambd, theta\_lambd\_norm, linewidth=**8**)  plt.grid(True)  plt.show()  Chart, line chart  Description automatically generated  4a-i |
|  |  |

Chart, line chart

Description automatically generated

4a-ii

Chart, line chart

Description automatically generated

4a-iii

4b)

Text, letter

Description automatically generatedText, letter

Description automatically generatedText, letter

Description automatically generatedText, letter

Description automatically generated

**Exercise 5: Project Check Point 2**

**Assigned Paper: Learning to Reweight Examples for Robust Deep Learning**

**What is problem that the proposed method addresses?**   
🡪 Deep neural networks overfit to training set biases and label noises. Regularizers and example reweighting algorithms are popular solutions to these problems. But these algorithms require careful tuning of additional hyperparameters,

**Why is the problem important?**Traditionally, validation is performed at the end of training, which can be prohibitively expensive if the example weights are treated as some hyperparameters to optimize. To circumvent this, they perform validation at every training iteration to dynamically determine the example weights of the current batch. This approach significantly increases the robustness to training set biases.

**What are the innovations of this paper compared to previous methods?**   
The paper proposes a learning algorithm that learns to assign weights to training examples based on their gradient directions. They perform gradient descent on the mini-batch example weights which are initialized to zero. Thereby they minimize the loss on clean unbiased validation set.

**Are there any existing implementations of the method? Which one will you start playing with?** The model in the paper is derived from a meta-learning objective towards an online approximation that can fit into any regular supervised training. Instead of minimizing the expected loss for the training set each input example is weighted equally and this reweighted input is used to model the dataset. There are multiple examples of automatic differentiation

Techniques which will be the correct starting point as it is needed to compute the gradient of the validation loss. The paper itself for example references a couple of other papers which implements it using popular deep learning frameworks such as TensorFlow

**Identify at least one key concept in the assigned paper that you are not familiar, and you need to learn to reimplement the paper.**   
As of now I have limited knowledge of the concept of deep neural networks and its implementation in software. This paper further narrows down the deep learning problem to batch learning which needs to be practically implemented. The paper itself was read in haste with a myopic vision of completing HW2 rather than to create a framework to work on the final project. Hence a thorough reading to understand the algorithm mentioned and its translation to a practical dataset needs to be realized.