CIS 520, Machine Learning, Fall 2019
Homework 6
Due: Monday, November 4th, 11:59pm
Submit to Gradescope

Matthew Scharf

November 4, 2019

1 EM Algorithm with Red and Blue Coins

1. $p(x, z; \theta) = (\pi p_r^x (1-p_r)^{1-x})^z ((1-\pi) p_b^x (1-p_b)^{1-x})^{1-z}$

2. $\ln \mathrm{Ł}_c(\theta) = \sum_{i=1}^m (z_i[ln(\pi) + x_i ln(p_r) + (1-x_i) ln(1-p_r)] + (1-z_i)[ln(1-\pi) + x_i ln(p_b) + (1-x_i) ln(1-p_b)])$

3.

$$
\begin{aligned}
0 &= \frac{\partial lnL_c(\theta)}{\partial \pi} = \sum_{i=1}^m (\frac{z_i}{\hat{\pi}} - \frac{1-z_i}{1-\hat{\pi}}) \\
\hat{\pi} &= \frac{\sum_{i=1}^m z_i}{m} \\
0 &= \frac{\partial lnL_c(\theta)}{\partial p_r} = \sum_{i=1}^m z_i (\frac{x_i}{\hat{p_r}} - \frac{1-x_i}{1-\hat{p_r}}) \\
\hat{p_r} &= \frac{\sum_{i=1}^m (z_i x_i)}{\sum_{i=1}^m z_i} \\
0 &= \frac{\partial lnL_c(\theta)}{\partial p_b} = \sum_{i=1}^m (1-z_i)(\frac{x_i}{\hat{p_b}} - \frac{1-x_i}{1-\hat{p_b}}) \\
\hat{p_b} &= \frac{\sum_{i=1}^m ((1-z_i) x_i)}{\sum_{i=1}^m (1-z_i)}
\end{aligned}
$$

4. $\mathbf{P}(Z_i = 1 \mid X_i = x_i; \theta^t) = \frac{p(X_i = x_i | Z_i = 1, \theta) p(Z_i = 1 | \theta}{p(X_i = x_i | \theta)} = \frac{\pi p_r^{x_i} (1-p_r)^{1-x_i}}{\pi p_r^{x_i} (1-p_r)^{1-x_i} + (1-\pi) p_b^{x_i} (1-p_b)^{1-x_i}}$

5. This is equivalent to our answer to (3) except we are replacing z_i with γ_i^t. So,

$$
\begin{aligned}
\hat{\pi}^{t+1} &= \frac{\sum_{i=1}^m \gamma_i^t}{m} \\
\hat{p}_r^{t+1} &= \frac{\sum_{i=1}^m (\gamma_i^t x_i)}{\sum_{i=1}^m \gamma_i^t} \\
\hat{p}_b^{t+1} &= \frac{\sum_{i=1}^m ((1-\gamma_i^t) x_i)}{\sum_{i=1}^m (1-\gamma_i^t)}
\end{aligned}
$$

2 Performance Measures for Face Detection in Images

I will evaluate the research groups using the following table structure:

TP	FP
FN	TN

1. (a) A

280	100
20	19,600

   i. TPR = .933

   ii. TNR = .995

   iii. GM = .964

   iv. Precision = .737

   v. F1 = .824

   (b) B

270	60
30	12,140

   i. TPR = .9

   ii. TNR = .995

   iii. GM = .946

   iv. Precision = .818

   v. F1 = .857

   (c) Based on GM, I would choose A, but based on F1, I would choose B.

2. F1 is a better performance measure because it disregards the high rate of easily identifiable negatives and focuses on precision and recall.

3. (a) (FPR, TPR) = (.33,.93)
   Distance = .337

   (b) (FPR, TPR) = (.2,.9)
   Distance = .224

   So, method (B) is better based on this metric.

4. (a) Yes, because $recall = TPR = .95 > .933, \quad .9$

   (b) No, because $specificity = TNR = .99 < .995$

3 Missing Data Imputation

3.1 Zero Imputation

- Add the accuracy and the Frobenius norm in this report.

Frobenius Norm	15925.766288536064
Accuracy	56.50557620817844height

Table 1: Accuracy and Frobenius norm for Zero Imputation

- Add the code of the completed zeroImpute method.

```
def zeroImpute(X_miss):
  '''
  Returns :
    X_imputed which has zeroes instead of missing values and same shape as X_miss.
  '''
  X_imputed = X_miss.copy()
  np.nan_to_num(X_imputed,copy=False)

  assert X_imputed.shape == X_miss.shape

  return X_imputed
```

Listing 1: zeroImpute method

3.2 Mean Imputation

- Add the accuracy and the Frobenius norm in this report.

Frobenius Norm	8851.631626598086
Accuracy	90.70631970260223height

Table 2: Accuracy and Frobenius norm for Mean Imputation

- Add the code of the completed meanImpute method.

```
def meanImpute(X_miss):
  '''
  Returns :
    X_imputed which has mean of the corresponding column instead of the missing values and same shape as X_miss.
  '''
  X_imputed = X_miss.copy()

  X_imputed = np.apply_along_axis(lambda x: np.nan_to_num(x, nan=np.nanmean(x)), 0, X_imputed)

  assert X_imputed.shape == X_miss.shape

  return X_imputed
```

Listing 2: meanImpute method

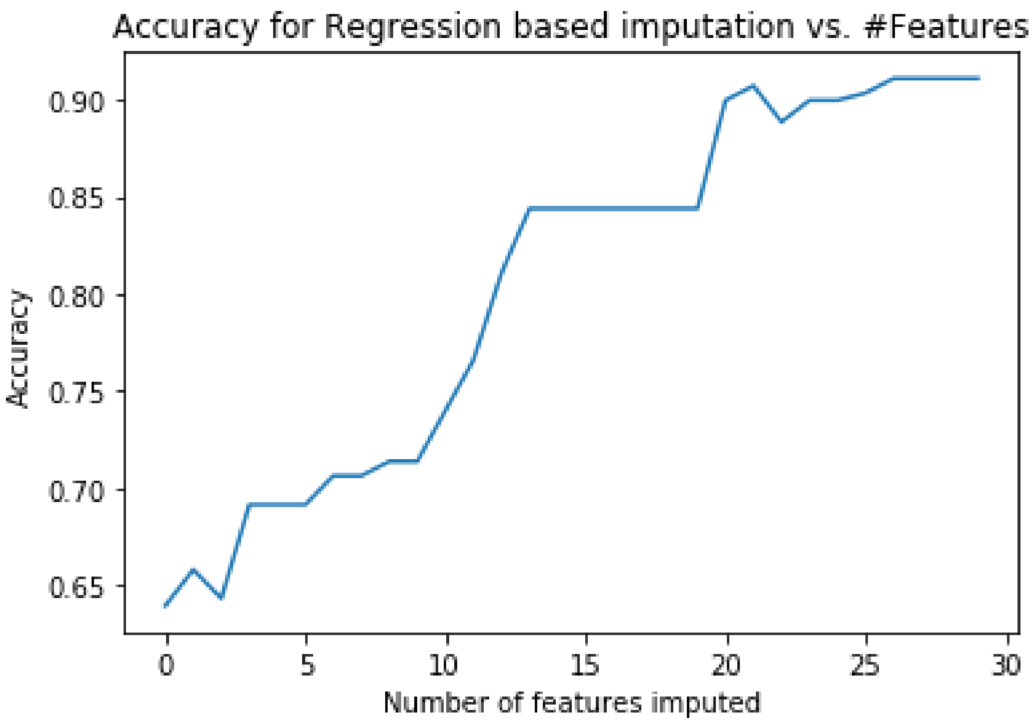
3.3 Regression Imputation

- Complete the following table.

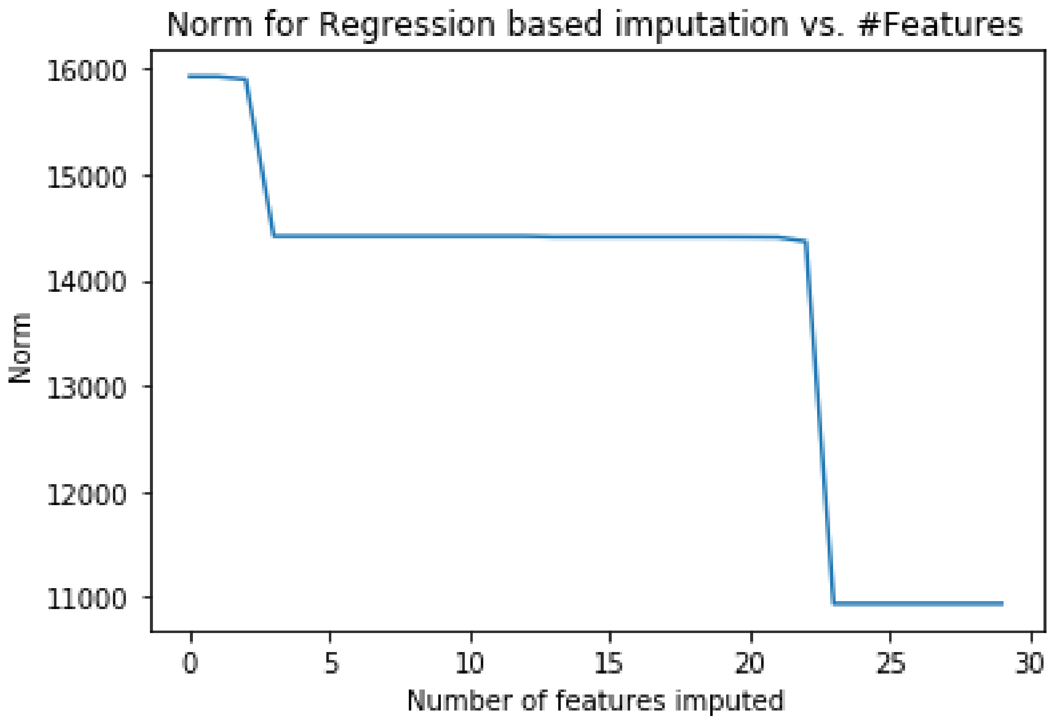
Epoch	Frobenius Norm	Accuracy
After Base Imputation	15925.766288536064	0.5650557620817844
2	10938.927994534588	0.9107806691449815
3	9430.426745903798	0.9256505576208178
4	8572.822146771927	0.9219330855018587
5	7740.276157478176	0.929368029739777
5	7090.749161839094	0.9144981412639405

Table 3: Accuracy and Frobenius norm for Zero Imputation

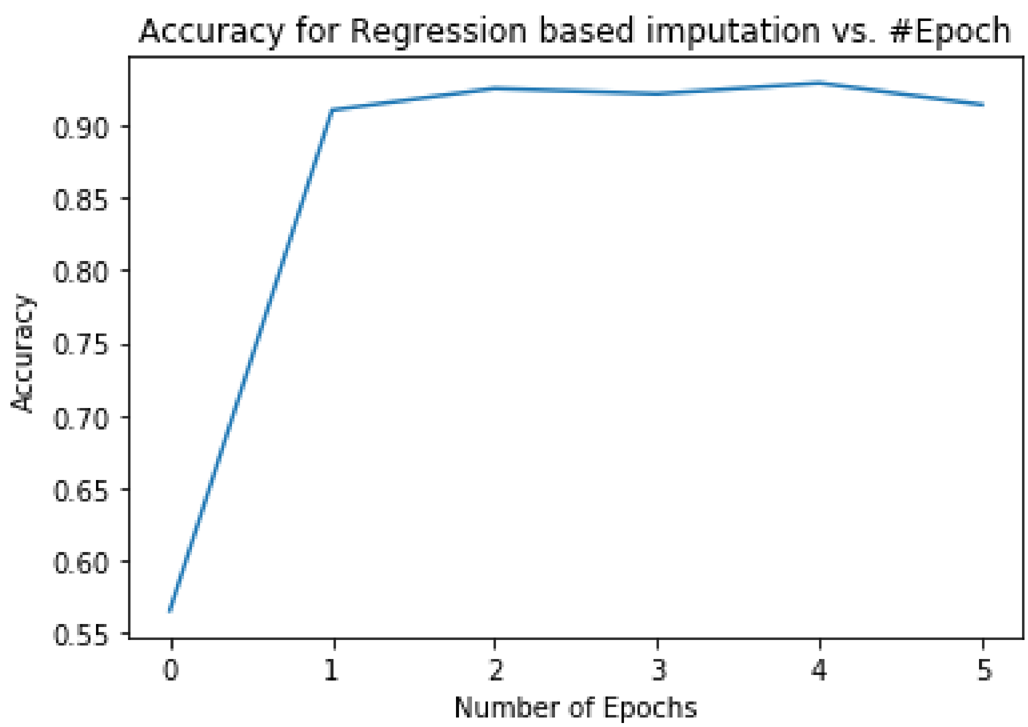
- Plot for Accuracy for Regression based imputation vs. Number of Features imputed



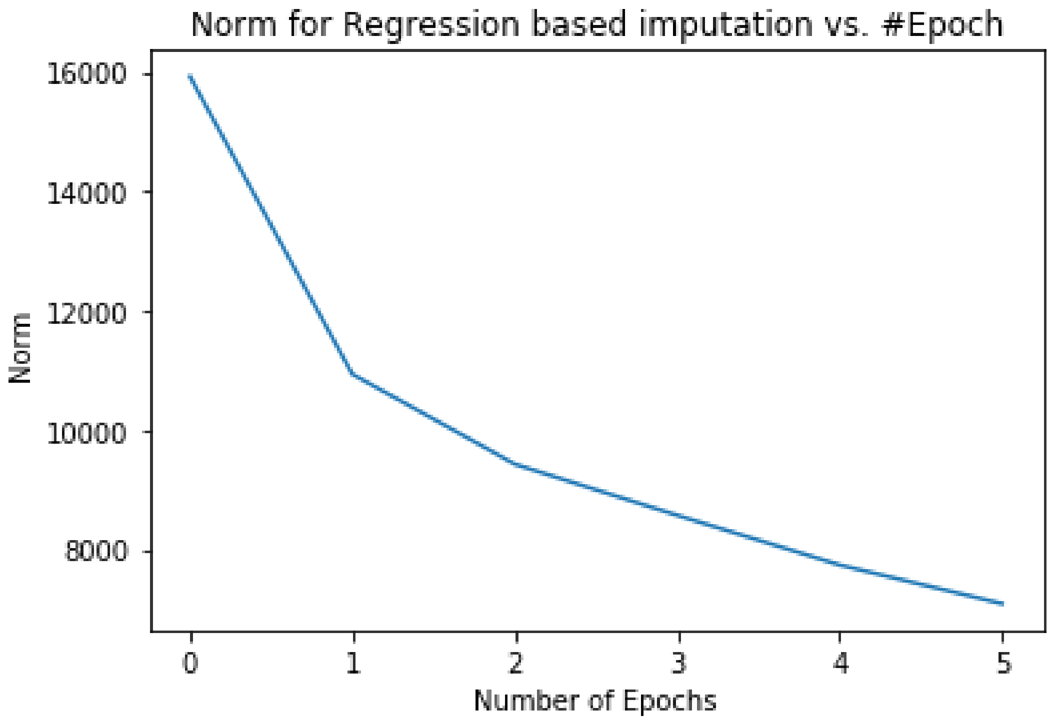
- Plot for Norm for Regression based imputation vs. Number of Features imputed



- Plot for Accuracy for Regression based imputation vs Number of Epochs



- Plot for Norm for Regression based imputation vs. Number of Epochs



- Add the code of the completed regressedImpute method.

```python
def regressedImpute(X_baseImputed, X_miss, X_test, y_test, computePerFeatureStatistics = False):
  '''
  Returns :
    X_imputed which has mean of the linearly regressed value instead of the missing values and same shape as X_miss.
  if computePerFeatureStatistics is True, also:
    list of Frobenius norms of difference between reconstructions and original data (without missing values) calculated after each imputing each column.
    list of accuracies on test set of Logistic Regression classifier trained on imputed data after each imputing each column.
  '''
  X_imputed = X_baseImputed.copy()
  frobenius_norms =[]
  accuracies =[]
  # TODO 3.3.1
  # We do a linear regression based imputation here, for each column, train a classifier to predict its value based on values of other features and
  # replace the NaN with the predicted values.
  # IMPORTANT : You should not use regressed values from an earlier column to predict a later column, make sure to train the regression model on base imputed
  #             and not modify base imputed during the run.
  #             You can use X_miss to find which values were originally NaNs.
  for feature_index in range(X_baseImputed.shape[1]):
      nan_indices = np.isnan(X_miss[:,feature_index])
      feature_train_x = np.delete(X_baseImputed, feature_index, axis=1)
      feature_train_y = X_baseImputed[:,feature_index]
      feature_model = LinearRegression()
      feature_model.fit(feature_train_x, feature_train_y)
      feature_prediction = feature_model.predict(feature_train_x)
      X_imputed[nan_indices ,feature_index] = feature_prediction[nan_indices]

      if computePerFeatureStatistics == True:
          model = LogisticRegression()
          model.fit(X_imputed,y_miss)
          accuracies.append(model.score(X_test,y_test))
```

```
        frobenius_norms.append(LA.norm(X_train - X_imputed))

  if computePerFeatureStatistics == True:
    return X_imputed, frobenius_norms, accuracies
  else:
    return X_imputed
```

Listing 3: regressedImpute method

3.4 Follow Up Questions

1. Regression imputation because it has the high accuracy and lowest Frobenius Norm from the original X.

2. You generally could provided that y is correlated with X. However, typically you separate y because it is something that you would like to predict with X. So, if you impute X with y, then you will gain information not available to you out of sample-this will cause over-fitting issues.

3. Accuracy goes up and the norm goes down. This is because the more we impute, the more we know about X, which will make our norm shrink and our predictive power of y increase.

4. Accuracy goes up and the norm goes down. This is because each iteration uses the past approximation of X to generate the next approximation of X. As such, with the each iteration, the approximation improves. As a result, the norm shrinks and our predictive power of y increase.

4 K-Means Analysis

1. Fill in the table below by reporting the resulting cluster labels and resulting centroids from running the iteration function on the given parameters.

X	Initial Centroids	Resulting Cluster Labels	Resulting Centroids
[[1], [2], [10], [12]]	[1, 2]	[0, 1, 1, 1]	[[1.], [8.]]
[[1], [2], [10], [12]]	[1, 8]	[0, 0, 1, 1]	[[1.5], [11.]]
[[1], [2], [10], [12]]	[2, 2]	[0, 0, 0, 0]	[[6.25], [2.]]
[[0,5,0],[0,5,0],[0,4,3],[0,3,4]]	[[2.5,0,0],[-2.5,0,0]]	[0, 0, 0, 0]	[[0. , 4.25, 1.75], [-2.5 , 0. , 0.]]

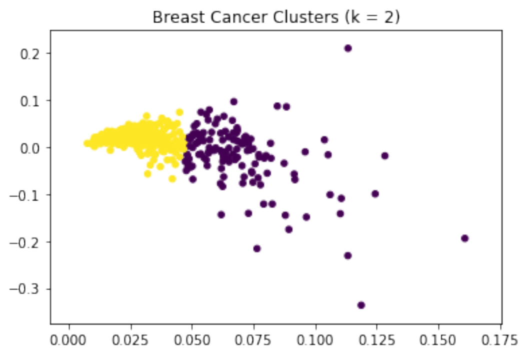
2. If an iteration of the k-means algorithm returns less than K classes, what might that indicate about the data?

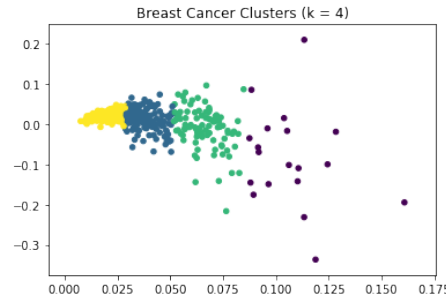
   It might indicate that the data does not need K classes for classification.

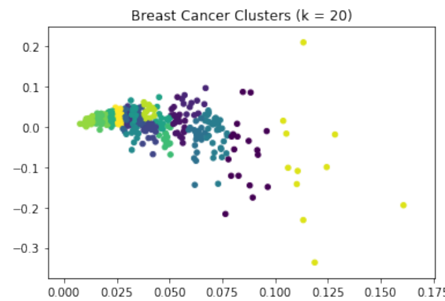
4.1 Part 2: Putting the algorithm together

Now write the whole K-means function using your iteration function, as is described in the notebook. The function takes in the dataset with the number of classes and returns the final centroid values, the final list of labels telling which class each datapoint belongs to, and the number of K-means iterations.

1. Put your 3 sanity-check graphs here.

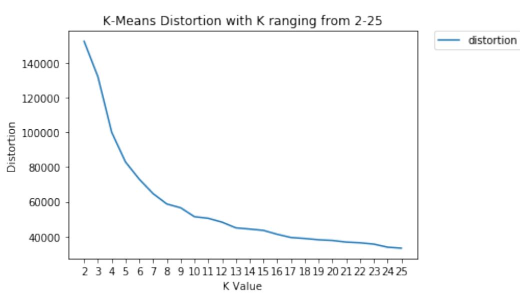


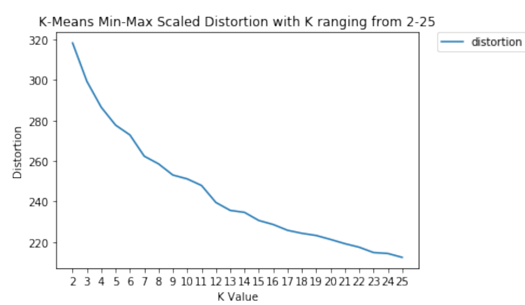


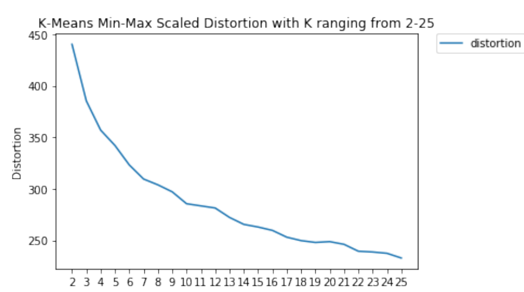


2. It took 9, 23, and 20 iterations respectively for k=2,4,20. I expected it to be positively correlated with k and it is when k is small. However, iterations to converge seems to decrease as k becomes large. This is likely because when k gets big enough, each class has very few points and so convergence speeds up.

1. Using the breast cancer dataset, record your labeled plots for distortion, min-max scaled distortion, and log scaled distortion here:



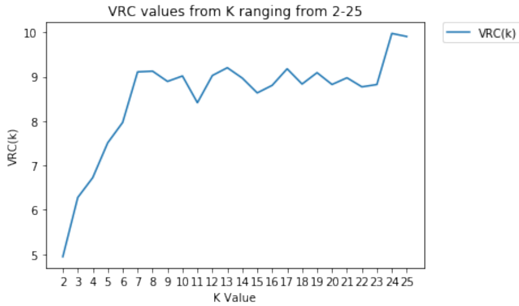




2. This is an unsupervised learning problem so there is no validation error-we are not trying to predict anything. For linear regression we could because it is a supervised learning problem.

4.2 Part 4: VRC

1. Using the breast cancer dataset, record your labeled plot for VRC(k) over different values of k from 2-15



2. Distortion and min-max distortion seem to suggest that k=5 is optimal but log distortion and VRC seem to suggest that k=7 is optimal. So, the chosen value of k varies slightly over the 4 methods, but not too much. In this case VRC seemed to give the clearest value of k to choose.