Data Mining

Assignment 4

Image Classification

Team Members:

Yassmin Barakat 86

Youssef Zook 90

Objectives:

1. understand the basic Image Classification pipeline and the data-driven approach (train/predict stages)
2. understand the train/val/test splits and the use of validation data for hyperparameter tuning.
3. develop proficiency in writing efficient vectorized code with numpy
4. implement and apply a k-Nearest Neighbor (kNN) classifier, (SVM) classifier, Softmax classifier and a Two layer neural network classifier
5. understand the differences and tradeoffs between these classifiers
6. get a basic understanding of performance improvements from using higher-level representations than raw pixels (e.g. color histograms, Histogram of Gradient (HOG) features)

Classifiers:

- ## KNN
    - ### Pseudocodes:

K_nearest_neighbour.py

TODO:

1. **def** compute_distances_two_loops(self, X):
2. dists[i][j] = np.sqrt(np.sum(np.square(X[i] - self.X_train[j])))

TODO:

1. **def** compute_distances_one_loop(self, X):
2. dists[i,:] = np.sqrt(np.sum(np.square(X[i] - self.X_train), axis = 1))

TODO:

1. **def** compute_distances_no_loops(self, X):
2. X_mul = np.dot(X, self.X_train.T)
3. X_test_sum = np.sum(np.square(X), axis = 1)
4. X_train_sum = np.sum(np.square(self.X_train), axis=1)
5. dists = np.sqrt(X_test_sum[:,np.newaxis] + X_train_sum[np.newaxis,:] - 2*X_mul)

TODO:

1. **def** predict_labels(self, dists, k=1):
2. sorted_neighbors_idx = np.argsort(dists[i])[0:k]
3. closest_y = self.y_train[sorted_neighbors_idx]
4. y_pred[i] = np.argmax(np.bincount(closest_y))

Knn.ipynb

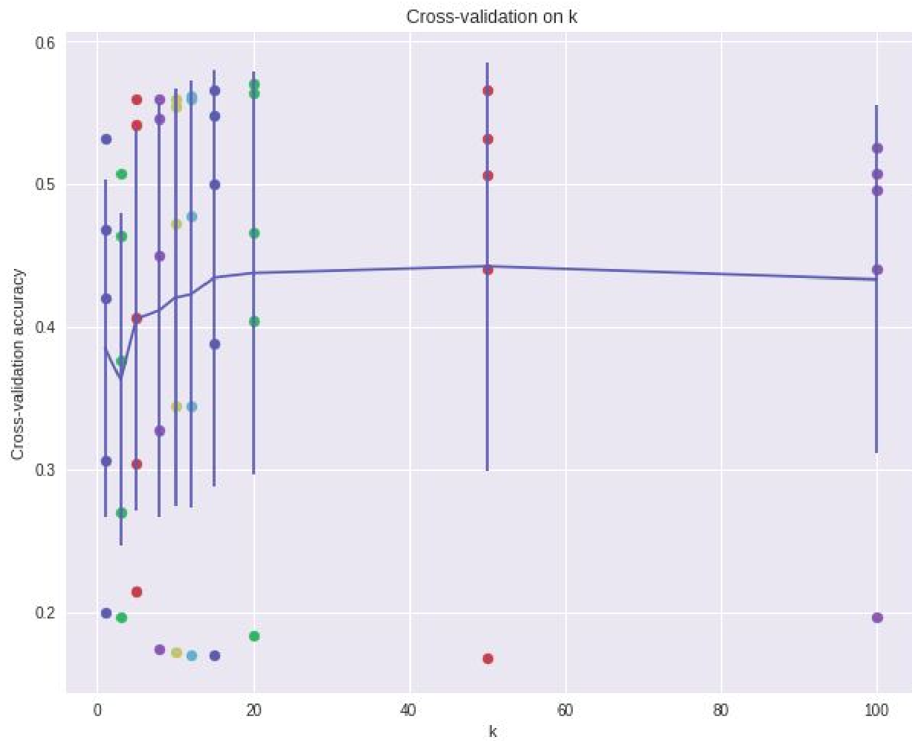
TODO:

1. for i in range(num_folds):
2. X_train_k = np.concatenate(np.delete(X_train_folds,(i), axis = 0))
3. y_train_k = np.concatenate(np.delete(y_train_folds,(i), axis = 0))
4. X_validation = X_train_folds[i]
5. y_validation = y_train_folds[i]
6. clf = KNearestNeighbor()
7. clf.train(X_train_k, y_train_k)
8. dis = clf.compute_distances_no_loops(X_validation)
9. for k in k_choices:
10. y_validation_pred = classifier.predict_labels(dis, k=k)
11. num_correct = np.sum(y_validation_pred == y_validation)
12. accuracy = float(num_correct) / num_test
13. k_to_accuracies[k].append(accuracy)

- ## Best accuracy based on cross validation:

Based on the cross-validation results, the best k is 20.

```
accuracy: 0.570000
```



- ## How data is presented:

Data is splitted into training and test sets, then each one is reshaped to (32*32*3) row pixels dimensions.

```
Training data shape:  (5000,3072)
Training labels shape:  (5000,)
Test data shape:  (500, 3072)
Test labels shape:  (500,)
```

- ## Model accuracy and results:

Based on the cross-validation results, the best k is 20.
Test set accuracy: 0.272000
When testing for k = 10, got 0.282000 accuracy

- SVM
  - Pseudocodes:

linear_svm.py

TODO:

```
def svm_loss_naive(W, X, y, reg):
for i in range(num_train):
 scores = X[i].dot(W)
 correct_class_score = scores[y[i]]
 for j in range(num_classes):
 if j == y[i]:
 continue
 margin = scores[j] - correct_class_score + 1 # note delta = 1
  if margin > 0:
  loss += margin
  dW[:,j] += X[i]
   dW[:,y[i]] -= X[i]
```

TODO:

```
def svm_loss_vectorized(W, X, y, reg):
delta = 1
scores = np.dot(X,W)
margin = np.maximum(0, scores - scores[np.arange(num_train), y][:, None] + delta)
margin[np.arange(num_train), y] = 0
loss = np.sum(margin)
loss /= num_train
loss += reg * np.sum(W * W)
temp_for_sum = np.zeros(margin.shape)
temp_for_sum[np.where(margin > 0)] = 1
temp_for_sum[np.arange(num_train), y] = -1 * np.sum(temp_for_sum, axis = 1)
dW = np.dot(X.T, temp_for_sum)
dW /= num_train
dW += reg * W
```

linear_classifier.py

TODO:

```
def train(self, X, y, learning_rate=1e-3, reg=1e-5, num_iters=100, batch_size=200, verbose=False):
 indices = np.random.choice(num_train, batch_size, replace=True)
X_batch = X[indices]
y_batch = y[indices]
loss, grad = self.loss(X_batch, y_batch, reg)
loss_history.append(loss)
self.W = self.W - (learning_rate * grad)
```

TODO:

```
def predict(self, X):
multi_class_pred = np.dot(X, self.W)
y_pred = np.argmax(multi_class_pred, axis = 1)
```

svm.ipynb

TODO:

```
for l_r in learning_rates:
for reg in regularization_strengths:
svm = LinearSVM()
svm.train(X_train, y_train, learning_rate=l_r, reg=reg,num_iters=1500, verbose=False)
y_train_pred = svm.predict(X_train)
y_val_pred = svm.predict(X_val)
accuracy_train = np.mean(y_train == y_train_pred)
accuracy_val = np.mean(y_val == y_val_pred)
results[(l_r, reg)] = (accuracy_train,accuracy_val)
End for
End for
if accuracy_val > best_val:
best_val = accuracy_val
best_svm = svm
End if
```

- ## Best accuracy based on cross validation:

```
best validation accuracy achieved during cross-validation:
0.385000
lr => 1.000000e-07 , reg => 5.000000e+04
train accuracy: 0.372571 , val accuracy: 0.385000
```

- ## How data is presented:

Data is splitted into training, validation, development and test sets, then each one is reshaped to (32*32*3) row pixels dimensions and then appended the bias dimension of ones.

```
Training data shape:  (49000, 3073)
Validation data shape:  (1000, 3073)
Test data shape:  (1000, 3073)
dev data shape:  (500, 3073)
```

- ## Model accuracy and results:

```
linear SVM on raw pixels final test set accuracy: 0.371000
```

- Two layer neural network
  - Pseudocodes:
    - **neural_net.py**

TODO: `Perform the forward pass`

h = X*W1+b1 #forward first layer , (N,H)
h_act = maximum(h, 0) # ReLU activation layer
y_hat = h_act* W2+b2 # layer 2 forward, (N,C)
scores = y_hat

TODO: `compute the loss`

# compute softmax probabilies
sco_exp = exp(scores - np.max(scores)) # (e^scores) normalized, (N,C)
den = sum(sco_exp, axis=1) # denominator, sigma(e^scores), (N,)
softmax_proba = sco_exp/den # = each row / sigma(row), (N,C)
# compute cross_entropy
log_liklihood =log(softmax_proba[range(N),y]) # -ln(Xi*Yi) ,
# -np.log(array that is have only the class #yi to get it's log)
entropy = -sum(log_liklihood) / N # -sigma(ln(Xi*Yi))/N
# compute regularized loss
loss = entropy + 0.5 * reg * (sum(W1^2) + sum(W2^2))

TODO: `Compute the gradients`

dE/d(y_hat) = [e^y_hat/sigma(e^y_hat)) - 1]*y
g[w2] = dE/d(y_hat) * d(y_hat)/d(w2)
g[b2] = dE/d(y_hat) * d(y_hat)/d(b2)
g[w1] = dE/d(y_hat) * d(y_hat)/d(hidden) * d(hidden)/d(w1) and neglect values < 0 as relu don't use them
g[b1] = dE/d(y_hat) * d(y_hat)/d(hidden) * d(hidden)/d(b1) and neglect values < 0 as relu don't use them
Where:
dh/dw1 = X
dh/db1 = 1
dy_hat/dh = W2
dy_hat/dw2 = h
dy_hat/db2 = 1

TODO: `Create a random minibatch`

random_idxs = np.random.choice(num_train, batch_size)
X_batch = X[random_idxs]
y_batch = y[random_idxs]

TODO: `Use the gradients in the grads dictionary to update the parameters`

self.params["W1"] -= learning_rate * grads["W1"]
self.params["b1"] -= learning_rate * grads["b1"]
self.params["W2"] -= learning_rate * grads["W2"]
self.params["b2"] -= learning_rate * grads["b2"]

TODO: `def predict(self, X):`

```
out1 = np.matmul(X, self.params["W1"]) + self.params["b1"]
   out1_relu = np.maximum(out1,0)
   out2 = np.matmul(out1_relu, self.params["W2"]) + self.params["b2"]
   y_pred = np.argmax(out2, axis=1) # get max label
```

- **two_layer_net.ipynb**

TODO: tune parameters - Grid Search

Loop on different values for [hidden size, learning rate, learning decay, epchs number. Regularization strenght]:

Create model with these params

Train the model

If the model is the best so far, store it in the best_net

Print best model validation

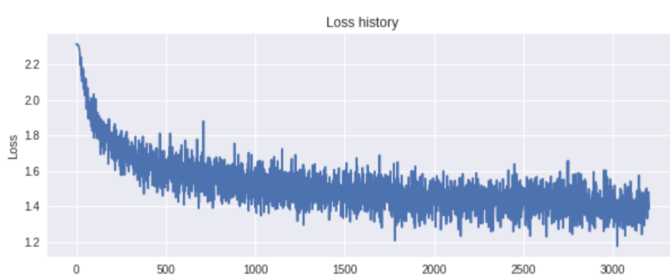
- Best accuracy based on cross validation:
  - 0.539000
- How data is presented:

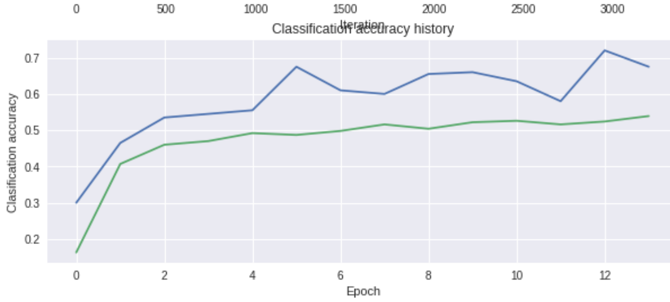
Train data shape: (49000, 3072)
Train labels shape: (49000,)
Validation data shape: (1000, 3072)
Validation labels shape: (1000,)
Test data shape: (1000, 3072)
Test labels shape: (1000,)

- Model accuracy and results:

Test accuracy: 0.531

Plot of the loss function and train / validation accuracies





- # Softmax
  - ## Pseudocodes:
    - ### Softmax.py

TODO: `Compute the softmax loss and its gradient using explicit loops.`

```
 # y_hat = X * W >> (N,D) * (D,C)
for n in range(N):          # for each sample
  for c in range(C):        # for each class
    for d in range(D):      # for each dimension
      y_hat[n, c] += X[n, d] * W[d, c]
  y_hat[n, :] = np.exp(y_hat[n, :])   # e^y_hat
  y_hat[n, :] /= np.sum(y_hat[n, :])  # e^y_hat / Sigma >> Softmax probabilites

# loss = -simga[ln(softmax probabilites)*y]/N + 0.5*reg*simga(W^2)
loss -= np.sum(np.log(y_hat[np.arange(N), y])) / N +  0.5 * reg * np.sum(W**2)

# dE/dW = dE/dy_hat * dy_hat/dW = (y_hat-1) * X >> # dw =  X.T * (y_hat-1) , (D,C)
y_hat[np.arange(N), y] -= 1   # (N, C)
for n in range(N):
  for d in range(D):
    for c in range(C):
      dW[d, c] += X[n, d] * y_hat[n, c]
# add reg term
dW = dW/N + reg*W
```

TODO: `Compute the softmax loss and its gradient using no explicit loops`

```
 # forward
score = np.dot(X, W)   # (N, C)
out = np.exp(score)
out /= np.sum(out, axis=1, keepdims=True)   # (N, C)
loss -= np.sum(np.log(out[np.arange(N), y]))
loss = loss/N + 0.5 * reg * np.sum(W**2)
# backward
dout = np.copy(out)   # (N, C)
dout[np.arange(N), y] -= 1
dW = np.dot(X.T, dout)  # (D, C)
dW = dW/N + reg * W
```

- ### softmax.ipynb

TODO: `Tune parameters, grid search`

Loop on different values for [learning rate, Regularization strength]:
	Create model with these params
	Train the model
	If the model is the best so far, store it in the best_net
	Print best model validation

- Best accuracy based on cross validation:

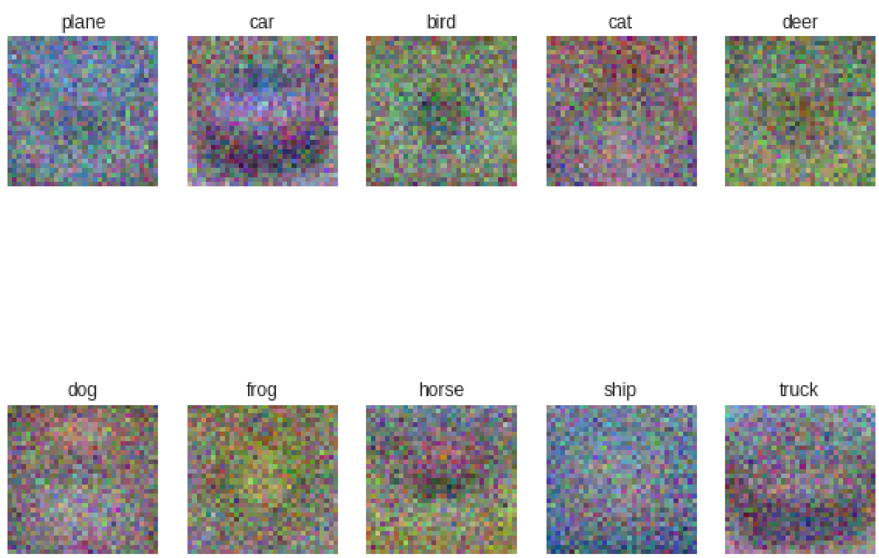
  0.376

- How data is presented:

  Train data shape: (49000, 3073)
  Train labels shape: (49000,)
  Validation data shape: (1000, 3073)
  Validation labels shape: (1000,)
  Test data shape: (1000, 3073)
  Test labels shape: (1000,)
  dev data shape: (500, 3073)
  dev labels shape: (500,)

- Model accuracy and results:

  test set accuracy: 0.349



Features:

For each image we will compute a Histogram of Oriented Gradients (HOG) as well as a color histogram using the hue channel in HSV color space. We form our final feature vector for each image by concatenating the HOG and color histogram feature vectors.

Roughly speaking, HOG should capture the texture of the image while ignoring color information, and the color histogram represents the color of the input image while ignoring texture. As a result, we expect that using both together ought to work better than using either alone. Verifying this assumption would be a good thing to try for your interests.

The hog_feature and color_histogram_hsv functions both operate on a single image and return a feature vector for that image. The extract_features function takes a set of images and a list of feature functions and evaluates each feature function on each image, storing the results in a matrix where each column is the concatenation of all feature vectors for a single image.

- ## Pseudocodes:

features.ipynb

TODO: svm

```
for l_r in learning_rates:
  for reg in regularization_strengths:
  svm = LinearSVM()
svm.train(X_train_feats,y_train,learning_rate=l_r,reg=reg,num_iters=1500,
verbose=False)
  y_train_pred = svm.predict(X_train_feats)
  y_val_pred = svm.predict(X_val_feats)
  accuracy_train = np.mean(y_train == y_train_pred)
  accuracy_val = np.mean(y_val == y_val_pred)
  results[(l_r, reg)] = (accuracy_train,accuracy_val)
  End for
  End for
  if accuracy_val > best_val:
  best_val = accuracy_val
  best_svm = svm
  End if
```

TODO: two layer neural network

```
for l_r in learning_rates:
  for reg in regularization_strengths:
  net = TwoLayerNet(input_dim, hidden_dim, num_classes)
net.train(X_train_feats,y_train,X_val_feats,y_val,learning_rate=l_r,
reg=reg,num_iters=2000, verbose=False)
  y_train_pred = svm.predict(X_train_feats)
  y_val_pred = svm.predict(X_val_feats)
  accuracy_train = np.mean(y_train == y_train_pred)
```

10. accuracy_val = np.mean(y_val == y_val_pred)
11. results[(l_r, reg)] = (accuracy_train,accuracy_val)
12. End for
13. End for
14. if accuracy_val > best_val:
15. best_val = accuracy_val
16. best_svm = svm
17. End if