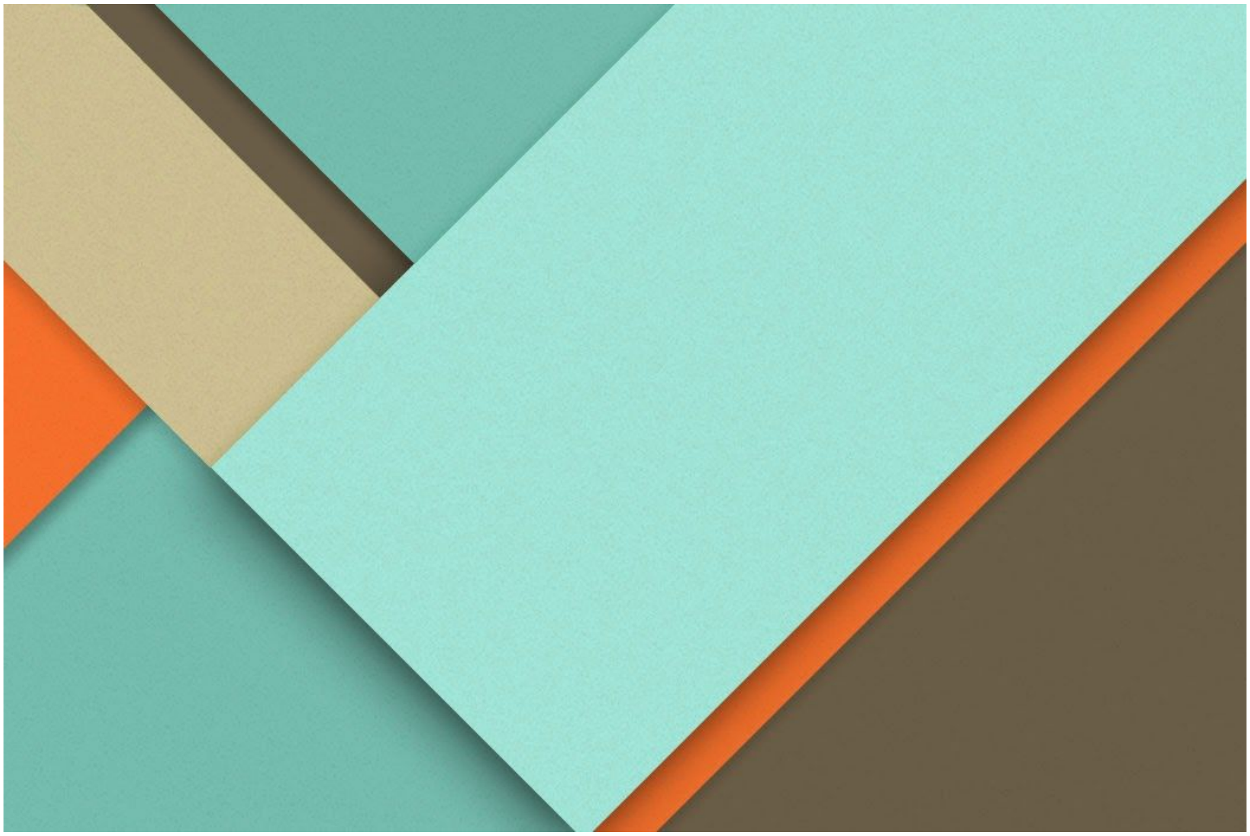


Assignment_3

Toqa alaa ahmed (14)

Nada salama mohamed ali (58)

Part I: Data Classification

Read Data

```
Results:

        - predicting field 2, diagnosis: B = benign, M = malignant
        - sets are linearly separable using all 30 input features
        - best predictive accuracy obtained using one separating plane
                in the 3-D space of Worst Area, Worst Smoothness and
                Mean Texture.  Estimated accuracy 97.5% using repeated
                10-fold crossvalidations.  Classifier has correctly
                diagnosed 176 consecutive new patients as of November
                1995.


5. Number of instances: 569

6. Number of attributes: 32 (ID, diagnosis, 30 real-valued input features)

7. Attribute information

1) ID number
2) Diagnosis (M = malignant, B = benign)


All feature values are recoded with four significant digits.

8. Missing attribute values: none

9. Class distribution: 357 benign, 212 malignant
```

```python
attr_names = ['Id', 'diagnosis']
for i in range (1,31):
    attr_names.append("attr "+ str(i))

data = pd.read_csv(data_path, sep=',',names=attr_names)
print(data.shape)
data.head(30)
```

(569, 32)

	Id	diagnosis	attr 1	attr 2	attr 3	attr 4	attr 5	attr 6	attr 7	attr 8	attr 9	attr 10	attr 11	attr 12
0	842302	M	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	1.0950	0.9053
1	842517	M	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	0.5435	0.7339
2	84300903	M	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	0.7456	0.7869
3	84348301	M	11.420	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	0.4956	1.1560
4	84358402	M	20.290	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	0.7572	0.7813
5	843786	M	12.450	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	0.2087	0.07613	0.3345	0.8902
6	844359	M	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	0.1794	0.05742	0.4467	0.7732
7	84458202	M	13.710	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	0.2196	0.07451	0.5835	1.3770
8	844981	M	13.000	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	0.2350	0.07389	0.3063	1.0020

Preprocessing

1- create feature and label

```
replace(['B','M'], [0,1])

Data.drop(columns=['Id', 'diagnosis'])
```

2- split to train and test

70% for training & 30% for test

3- Scaling

by using Standard Scaler It standardize features by removing the mean and scaling to unit variance The standard score of a sample x is calculated as: z = (x - u) / s

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print(X_train.shape)
print(X_test.shape)

```

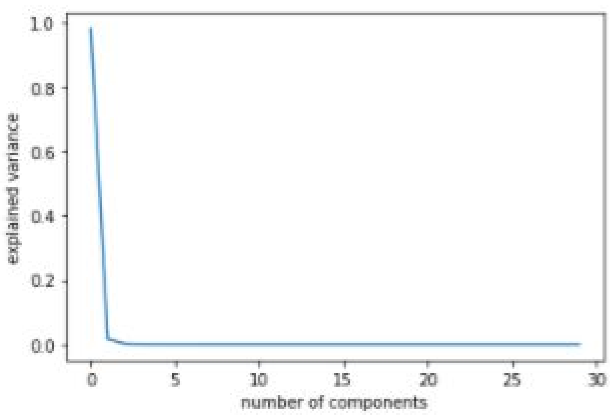
```
(398, 30)
(171, 30)
```

4- GridSearchCV

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

params = {
    'C': [1.0, 2.0, 2.5, 5.0, 10, 100, 1000]
}

model_GSCV = GridSearchCV(LogisticRegression(),scoring='accuracy', param_grid = params, cv = 10)

model_GSCV.fit(X_train, y_train)
print("best_parameters of model",model_GSCV.best_params_ )
y_pred = model_GSCV.predict(X_test)

# Model Accuracy
print("Train Accuracy: %.3f"% model_GSCV.score(X_train, y_train))
print("Test Accuracy:",metrics.accuracy_score(y_test, y_pred))
print(metrics.confusion_matrix(y_test, y_pred))
print(metrics.classification_report(y_test, y_pred,target_names=['M','B']))
```

5- feature selection/ feature projection (PCA)

```
plt.xlabel('number of components')
plt.ylabel('explained variance')
plt.show()
```



```
pca = PCA(0.9)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
X_train = pd.DataFrame(data = X_train_pca)
X_test = pd.DataFrame(data = X_test_pca)

print(X_train.shape)
print(X_test.shape)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

```
(398, 7)
(171, 7)
```

```
print (pca.explained_variance_ratio_ ) #percentage of variance explained by each of the selected components
```

```
[0.43689315 0.19415163 0.09661545 0.06716611 0.0549883  0.04012257
 0.02183068]
```

Training

1- train before applying feature selection/ feature projection (PCA)

```
best_parameters of model {'C': 5.0}
Train Accuracy: 0.952
Test Accuracy: 0.9532163742690059
[[101   7]
 [  1  62]]
              precision    recall  f1-score   support

           M       0.99      0.94      0.96       108
           B       0.90      0.98      0.94        63

    accuracy                           0.95       171
   macro avg       0.94      0.96      0.95       171
weighted avg       0.96      0.95      0.95       171
```

7- train after applying feature selection/ feature projection (PCA)

Without GridSearchCV

```
Accuracy: 0.9649122807017544
[[107   1]
 [  5  58]]
              precision    recall  f1-score   support

           M       0.96      0.99      0.97       108
           B       0.98      0.92      0.95        63

    accuracy                           0.96       171
   macro avg       0.97      0.96      0.96       171
weighted avg       0.97      0.96      0.96       171
```

```
#if {'C': 1.0}

train score: 0.977
Accuracy: 0.9649122807017544
[[107   1]
[  5  58]]
          precision    recall  f1-score   support

       M       0.96      0.99      0.97       108
       B       0.98      0.92      0.95        63

accuracy                           0.96       171
macro avg       0.97      0.96      0.96       171
weighted avg    0.97      0.96      0.96       171
```

With GridSearchCV

```
best_parameters of model {'C': 2.5}
Train Accuracy: 0.980
Test Accuracy: 0.9590643274853801
[[106   2]
 [  5  58]]
              precision    recall  f1-score   support

           M       0.95      0.98      0.97       108
           B       0.97      0.92      0.94        63

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171
```

Part II: Image Classification

Two-layer neural network classifier

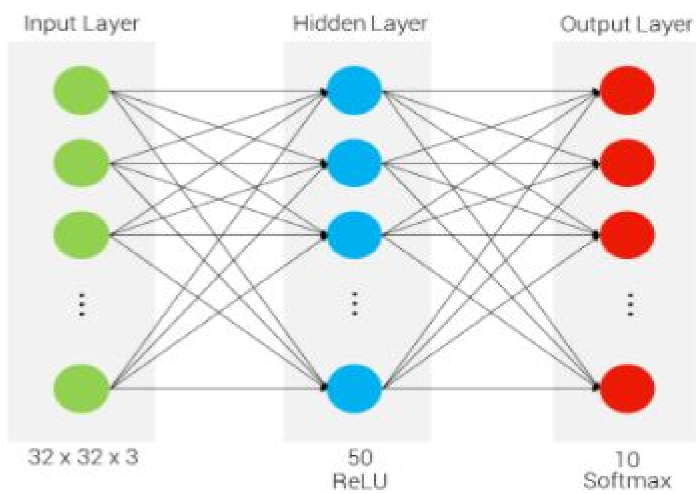


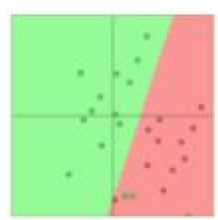
Figure 1: *Artificial Neural Network Architecture*

Softmax classification and regularization

- Softmax classification

$$p(y_j = 1|x) = \frac{\exp(W_j \cdot x)}{\sum_{c=1}^{C} \exp(W_c \cdot x)}$$

x: *word vector*
W: *weight*
C: *class*



- Cross-entropy loss function

$$-\sum_{j=1}^{C} y_j \log\left(p(y_j = 1|x)\right) = -\sum_{j=1}^{C} y_j \log\left(\frac{\exp(W_j \cdot x)}{\sum_{c=1}^{C} \exp(W_c \cdot x)}\right)$$

- Simplified version

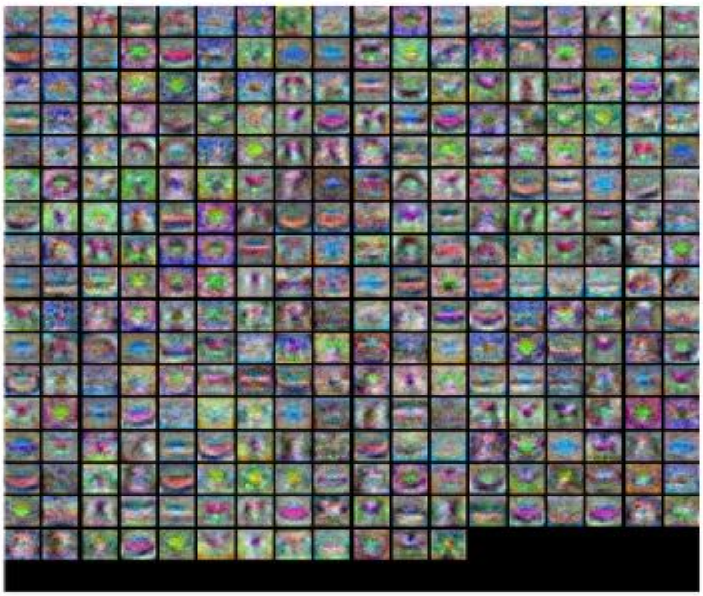
$$-\log\left(\frac{\exp(W_k \cdot x)}{\sum_{c=1}^{C} \exp(W_c \cdot x)}\right)$$

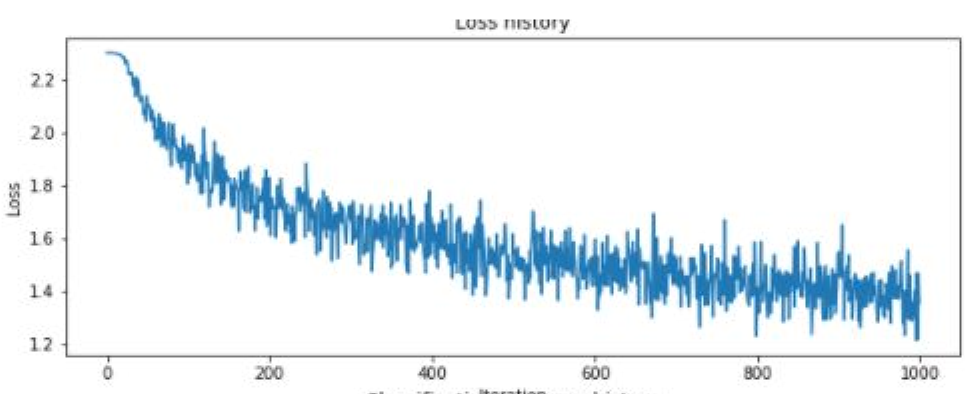
k: *index of the correct class*

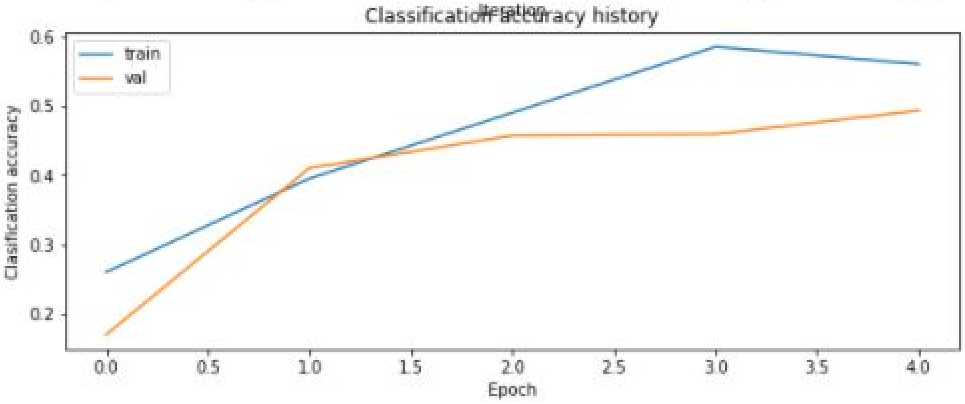
- Loss to a dataset of N points

$$-\sum_{i=1}^{N} \log\left(\frac{\exp(W_{k(i)} \cdot x^{(i)})}{\sum_{c=1}^{C} \exp(W_c \cdot x^{(i)})}\right)$$

```
iteration 0 / 1000: loss 2.302594
iteration 100 / 1000: loss 1.942689
iteration 200 / 1000: loss 1.828271
iteration 300 / 1000: loss 1.665591
iteration 400 / 1000: loss 1.632863
iteration 500 / 1000: loss 1.558518
iteration 600 / 1000: loss 1.590588
iteration 700 / 1000: loss 1.521043
iteration 800 / 1000: loss 1.412103
iteration 900 / 1000: loss 1.548200
h_300_lr_0.001_r_0.0  Validation accuracy:  0.5
better accuracy:0.500000
```







Inline Question

Now that you have trained a Neural Network classifier, you may find that your testing accuracy is much lower than the training accuracy. In what ways can we decrease this gap? Select all that apply.

1. Train on a larger dataset.
2. Add more hidden units.
3. Increase the regularization strength.
4. None of the above.

Your answer:

1 and 3

Your explanation:

1: if we get more data the train and test data set will increase so we can generalize our data and get better result

3: to avoid noisy features which make overfitting. where these noisy be useful at training but they are actually redundant

Softmax Classifier

Data Representation

Inputs have dimension D

There are C classes

The mini batches are of sizeof N examples.

W: A numpy array of shape (D, C) containing weights. --> initialized randomly

X: A numpy array of shape (N, D) containing a minibatch of data. --> random choice

$$S_y = \frac{e^{f_y}}{\sum_{i=1}^{C} e^{f_i}}$$

$$L = -log(S_y) + \frac{1}{2}\lambda \sum W^2$$

$$\frac{\partial L}{\partial W_i} = \begin{cases} -\frac{1}{S_y} S_y(1 - S_i)X + \lambda W_i & \text{if } i = y \\ -\frac{1}{S_y}(-S_i S_y)X + \lambda W_i & \text{if } i \neq y \end{cases} = \begin{cases} (S_i - 1)X + \lambda W_i & \text{if } i = y \\ S_i X + \lambda W_i & \text{if } i \neq y \end{cases}$$

Gradient checking result

```
numerical: 0.494256 analytic: 0.494256, relative error: 2.452256e-08
numerical: 0.671965 analytic: 0.671965, relative error: 8.584771e-08
numerical: -1.860149 analytic: -1.860149, relative error: 1.077599e-08
numerical: 0.910240 analytic: 0.910240, relative error: 4.060607e-08
numerical: 1.316211 analytic: 1.316211, relative error: 2.368138e-08
numerical: 1.178860 analytic: 1.178860, relative error: 5.385546e-08
numerical: 1.007153 analytic: 1.007153, relative error: 2.923655e-08
numerical: 1.076064 analytic: 1.076064, relative error: 1.094703e-08
numerical: 1.285390 analytic: 1.285390, relative error: 1.268162e-08
numerical: 0.525046 analytic: 0.525046, relative error: 5.140283e-08
numerical: -3.720903 analytic: -3.720903, relative error: 5.991534e-09
numerical: 0.888902 analytic: 0.888902, relative error: 4.451598e-09
numerical: 0.327333 analytic: 0.327333, relative error: 1.192196e-07
numerical: -0.673430 analytic: -0.673430, relative error: 1.234302e-07
numerical: 1.372302 analytic: 1.372302, relative error: 4.447484e-08
numerical: -1.415692 analytic: -1.415692, relative error: 5.324776e-09
numerical: -1.701768 analytic: -1.701768, relative error: 7.368061e-09
numerical: 0.643266 analytic: 0.643266, relative error: 2.835033e-08
numerical: 0.066093 analytic: 0.066093, relative error: 7.075825e-07
numerical: 3.777125 analytic: 3.777125, relative error: 7.164153e-10
```

Relative error is very low so we can conclude that grad is computed correctly.

Comparison between naive and vectorized loss

```
naive loss: 2.336267e+00 computed in 27.606042s
vectorized loss: 2.336267e+00 computed in 0.008121s
Loss difference: 0.000000
Gradient difference: 0.000000
```

As shown from the result, they both have the same value but vectorized loss is computed much faster.

Validation and Test accuracy

```
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.337612 val accuracy: 0.348000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.328612 val accuracy: 0.339000
lr 5.000000e-07 reg 2.500000e+04 train accuracy: 0.345204 val accuracy: 0.366000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.320061 val accuracy: 0.340000
best validation accuracy achieved during cross-validation: 0.366000
```

```
softmax on raw pixels final test set accuracy: 0.325000
```

Inline Questions

Inline Question 1:

Why do we expect our loss to be close to -log(0.1)? Explain briefly.**

Your answer: because W is initilaized randomly so the probability of select any of the classes are equal. As we have 10 classes, p =0.1 and then loss = -log(p) = -log(0.1)

Inline Question - *True or False*

It's possible to add a new datapoint to a training set that would leave the SVM loss unchanged, but this is not the case with the Softmax classifier loss.

Your answer: True for softmax

Your explanation: The Softmax classifier takes all datapoints scores into account in the calculation of the loss

Features

Extract The HOG and color histogram features from the data set and use them to train the two-layer net classifier.

`X_train shape =(49000, 154)`

`Where number of images = 49000 and number of features = 154`

`hidden_dim = 500`

`num_classes = 10`

Training results

```
lr 1.000000e-02 reg 5.000000e-06 train accuracy: 0.865898 val accuracy: 0.595000
lr 1.000000e-02 reg 5.000000e-05 train accuracy: 0.861061 val accuracy: 0.606000
lr 1.000000e-02 reg 5.000000e-04 train accuracy: 0.854673 val accuracy: 0.606000
lr 5.000000e-02 reg 5.000000e-06 train accuracy: 0.844347 val accuracy: 0.593000
lr 5.000000e-02 reg 5.000000e-05 train accuracy: 0.815878 val accuracy: 0.603000
lr 5.000000e-02 reg 5.000000e-04 train accuracy: 0.784571 val accuracy: 0.609000
lr 1.000000e-01 reg 5.000000e-06 train accuracy: 0.755837 val accuracy: 0.606000
lr 1.000000e-01 reg 5.000000e-05 train accuracy: 0.680224 val accuracy: 0.606000
lr 1.000000e-01 reg 5.000000e-04 train accuracy: 0.584673 val accuracy: 0.560000
best validation accuracy achieved during cross-validation: 0.609000
```

```
# Run your best neural net classifier on the test set. You should be able
# to get more than 55% accuracy.

test_acc = (best_net.predict(X_test_feats) == y_test).mean()
print(test_acc)
```

```
0.59
```