Assignment 2
Saumya Rawat
201401110

Report : Naive Bayes Classifier

Solution 1: Naïve Bayes Classifier for UCI Census-Income (KDD) Data Set

Pre Processing of the data :

- The data contains 37 nominal features,used as categorical features for classification.
- There are 7 continous features, out of which age, wage per hour, capital gains were binned according to the histograms plotted.
- The missing data for categorical features was replaced with the mode of the feature as mean cannot be used and it was seen that one of the resulting values was dominant in the observations for the features.

```
if __name__ == "__main__":
    train_data = read_data("dataset/census-income.data")
    test_data = read_data("dataset/census-income.data")

    #Applying per column:
    print(train_data.apply(num_missing, axis=0)) #axis=0 defines that function is to be applied on each column

    #Binning ages
    #plt.hist(list(train_data['age']), facecolor='green', alpha=0.75, bins=9)
    bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
    group_names = [1,2,3,4,5,6,7,8,9]
    categories = pd.cut(train_data['age'], bins, labels=group_names)
    train_data['age'] = pd.cut(train_data['age'], bins, labels=group_names)

    categories = pd.cut(test_data['age'], bins, labels=group_names)
    test_data['age'] = pd.cut(test_data['age'], bins, labels=group_names)

    #Binning wage per hour
    train_data.hist(column="wage_per_hour",bins=30)
    group_names = [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]
    train_data['wage_per_hour'] = pd.qcut(train_data['wage_per_hour'], 30, labels=group_names)

    #Binning capital_gains
    train_data.hist(column="capital_gains",bins=30)
    group_names = [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]
    train_data['capital_gains'] = pd.qcut(train_data['capital_gains'], 30, labels=group_names)

    #Nominal labels to numeric values
    cat_columns = train_data.select_dtypes(['object']).columns
    train_data[cat_columns] = train_data[cat_columns].apply(lambda x: x.astype('category'))
    train_data[cat_columns] = train_data[cat_columns].apply(lambda x: x.cat.codes)

    test_data[cat_columns] = test_data[cat_columns].apply(lambda x: x.astype('category'))
    test_data[cat_columns] = test_data[cat_columns].apply(lambda x: x.cat.codes)

    #Coding LoanStatus as Y=1, N=0:
    train_data[cat_columns] = coding(train_data[cat_columns], {'N':0,'Y':1})

    train_data.to_csv("dataset/train_data.csv", sep=',')
    test_data.to_csv("dataset/test_data.csv", sep=',')
```

•

<u>Naive Bayes Classifier:</u>

The Naive Bayes Classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. It is easy to build and useful for large datasets.

Likelihood Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)\,P(c)}{P(x)}$$

Posterior Probability Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Pipeline of the classifier :

```
if __name__ == "__main__":
    accuracy_mat = []
    for t in range(10):
        model = Model("dataset/train_data.csv")
        model.load_database()
        splitRatio = 0.7
        model.k_fold_split(splitRatio)
        print('Size of train=',len(model.trainingSet),' and test=',len(model.testSet))
        model.summarize_by_class()
        predictions = model.test_classifier()
        accuracy = model.get_accuracy(predictions)
        print('Accuracy[',t,']: ',accuracy)
        accuracy_mat.append(accuracy)
    mean_accuracy = np.mean(accuracy_mat)
    stddev_accuracy = np.std(accuracy_mat,ddof = 1)
    print('For 10 runs, Mean Accuracy : ',mean_accuracy,' and Standard Deviation: ',stddev_accuracy)
```

Test Runs:

- The data was sampled as 70% train and 30% test for 10 runs and the following accuracy and standard deviation was reported :

```
Size of train= 139666  and test= 59857
Accuracy[ 0 ]:  70.62164826169037
Size of train= 139666  and test= 59857
Accuracy[ 1 ]:  70.78370115441803
Size of train= 139666  and test= 59857
Accuracy[ 2 ]:  70.71019262575805
Size of train= 139666  and test= 59857
Accuracy[ 3 ]:  70.60995372303991
Size of train= 139666  and test= 59857
Accuracy[ 4 ]:  70.5297626008654
Size of train= 139666  and test= 59857
Accuracy[ 5 ]:  70.79205439631122
Size of train= 139666  and test= 59857
Accuracy[ 6 ]:  70.93907145363116
Size of train= 139666  and test= 59857
Accuracy[ 7 ]:  70.62666020682626
Size of train= 139666  and test= 59857
Accuracy[ 8 ]:  70.63668409709808
Size of train= 139666  and test= 59857
Accuracy[ 9 ]:  70.66174382277762
For 10 runs, Mean Accuracy :  70.6911472342  and Standard Deviation:  0.118337790033
```

Solution 2: Derivations

(1)

we want to find $P(Y|X)$ where $Y \to$ label, $X \to$ data

but since $P(X|Y)$ is easier to model, we write

$$\underbrace{P(Y|X)}_{\text{posterior}} = \frac{\overbrace{P(X|Y)}^{\text{likelihood}}\ \overbrace{P(Y)}^{\text{prior}}}{P(X)}$$

$P(X)$ is unknown but has a known parametric form,

$$P(x|\theta)$$

↳ unknown parameter.

we might have expert domain knowledge of

prior $\to P(\theta)$ & posterior $\to P(\theta|D)$ (D: set of samples).

$\because$ x, D selected independently $\Rightarrow P(x|\theta, D) = P(x|\theta)$

$$\Rightarrow P(x|D) = \int p(x|\theta)\, p(\theta|D)\, d\theta.$$

for gaussian distribution:

$P(x|D), P(\theta|D) = ?$ where $p(x|\mu) \sim N(\mu, \Sigma)$

Univariate case:

$p(x|\mu) \sim N(\mu, \sigma^2)$ — (2) (μ: unknown, σ^2: known), assuming $p(\mu) \sim N(\mu_0, \sigma_0^2)$ — (3)

best guess for μ uncertainty

then Bayes formula $\Rightarrow$

$$P(\mu|D) = \frac{P(D|\mu)P(\mu)}{\int P(D|\mu)\, P(\mu)\, d\mu} = \alpha P(D|\mu) P(\mu) \quad (4)$$

hence we have, (2), (3), (4) $\Rightarrow$

$$P(\mu|D) = \alpha \prod_k \left[\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma}}\right] \left[\frac{1}{\sqrt{2\pi}\sigma_0} e^{-\frac{1}{2}\frac{(\mu-\mu_0)^2}{\sigma_0}}\right] \quad (5)$$

②

$$\Rightarrow P(\mu|D) = \alpha\, e^{-\frac{1}{2}\left[\sum\left(\frac{\mu-x_k}{\sigma}\right)^2 + \left(\frac{\mu-\mu_0}{\sigma_0}\right)^2\right]}$$

$$= \alpha\, e^{-\frac{1}{2}\left[\left(\frac{n}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\mu^2 - 2\left(\frac{1}{\sigma^2}\sum x_k + \frac{\mu_0}{\sigma_0}\right)\mu\right]}$$

$$\therefore \frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}, \quad \frac{\mu_n}{\sigma_n^2} = \frac{n}{\sigma^2}\bar{x}_n + \frac{\mu_0}{\sigma_0^2}$$ (from std form) ⑦

$\bar{x}_n \rightarrow$ sample mean, $\bar{x}_n = \frac{1}{n}\sum_{k=1}^{n} x_k$

$$⑦ \Rightarrow \mu_n = \frac{n\sigma_0^2}{n\sigma_0^2+\sigma^2}\bar{x}_n + \frac{\sigma^2}{n\sigma_0^2+\sigma^2}\mu_0, \quad \sigma_n^2 = \frac{\sigma_0^2\sigma^2}{n\sigma_0^2+\sigma^2}$$ ⑧

$$\therefore p(x|D) = \frac{1}{2\pi\sigma\sigma_n} e^{\frac{(x-\mu_n)^2}{\sigma^2+\sigma_n^2}} \int e^{-\frac{1}{2}\frac{\sigma^2+\sigma_n^2}{\sigma^2\sigma_n^2}\left(\mu - \frac{\sigma_n^2 x + \sigma^2\mu_n}{\sigma^2+\sigma_n^2}\right)^2} d\mu$$

we can say, $p(x|D) \sim N(\mu_n, \sigma^2+\sigma_n^2)$

comparing to standard form:

$$p(\mu|D) = \alpha\, e^{(\mu-\mu_n)^t \Sigma_n^{-1}(\mu-\mu_n)}$$

like to ⑦ $\Rightarrow$ $\Sigma_n^{-1} = n\Sigma^{-1} + \Sigma_0^{-1}$ & $\Sigma_n^{-1}\mu_n = n\Sigma^{-1}\hat{\mu}_n + \Sigma_0^{-1}\mu_0$.

$\hookrightarrow = \frac{1}{n}\sum_k x_k$

like to ⑧ $\Rightarrow$ $\mu_n = \Sigma_0\left(\Sigma_0 + \frac{1}{n}\Sigma\right)^{-1}\hat{\mu}_n + \frac{1}{n}\Sigma\left(\Sigma_0+\frac{1}{n}\Sigma\right)^{-1}\mu_0$. &

$\Sigma_n = \Sigma_0\left(\Sigma_0 + \frac{1}{n}\Sigma\right)^{-1}\frac{1}{n}\Sigma$

finally we have $P(x|D) \sim N(\mu_n, \Sigma+\Sigma_n)$

For the multivariate case:

$p(x|\mu) \sim N(\mu, \Sigma)$ & $p(\mu) \sim N(\mu_0, \Sigma_0)$

using Bayes formula,

$$p(\mu|D) = \alpha \prod_{k=1}^{n} p(x_k|\mu)\, p(\mu)$$

$\parallel$ le to 5 ~~ili~~ ~~EEP~~

$$p(\mu|D) = \alpha'' e^{-\frac{1}{2}\left[\mu^t(n\Sigma^{-1} + \Sigma_0^{-1})\mu - 2\mu^t\left(\Sigma^{-1}\sum x_k + \Sigma_0^{-1}\mu_0\right)\right]}$$

Solution 3: PCA, LDA and Gaussian Naive Bayes Classifier

<u>PCA :</u>

Principal component analysis is often used to reduce the dimensionality and bring out strong patterns in a dataset. The steps in PCA are :

- Centre the dataset, Compute the d-dimensional mean vector (i.e., the means for every dimension of the whole dataset)
- Compute the the covariance matrix of the whole data set
- Compute eigenvectors (e1,e2,...,ed) and corresponding eigenvalues (λ1,λ2,...,λd)
- Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a d×k dimensional matrix W
- Use this d×k eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the mathematical equation: $y=W^T \times x$ (where x is a d×1-dimensional vector representing one sample, and y is the transformed k×1-dimensional sample in the new subspace.)
- Since in the particular dataset, d >> n, Kernel PCA was used.

Now if A is an $m \times n$ matrix, with $n >> m$, then $A^T A$ is a very large $n \times n$ matrix. So instead of computing the eigenvectors of $A^T A$, we might like to compute the eigenvectors of the much smaller $m \times m$ matrix AA^T -- assuming we can figure out a relationship between the two. So how are the eigenvectors of $A^T A$ related to the eigenvectors of AA^T?

Let v be an eigenvector of AA^T with eigenvalue λ. Then

- $AA^T v = \lambda v$
- $A^T(AA^T v) = A^T(\lambda v)$
- $(A^T A)(A^T v) = \lambda(A^T v)$

```python
lines = open('dataset/dorothea_train.data', 'r').readlines()
dataList = [line.rstrip('\n') for line in lines]
k = len(max(dataList,key=len))
#data = [[]]

#x=for i in range(len(dataList)):
    #dataList[0].extend([data0]*(k-len(dataList[i])))
cols = np.arange(k)

train_data = pd.read_csv(
    filepath_or_buffer='dataset/dorothea_train.data',
    header=None,
    sep=" ",
    names=cols,
    engine = 'python')

train_data=train_data.fillna(0)
X = lil_matrix((train_data.shape[0],100001))
for i in range(train_data.shape[0]):
   X[i,train_data[i]] = 1

k=500

#Kernel_PCA since d>>n
K=X.dot(X.T)

#Centering Kernel since data has to be standardizied
kern_cent = KernelCenterer()
S = kern_cent.fit_transform(K.toarray())

#val,vec=linalg.eigs(S,k,which='LM')

eig_vals, eig_vecs = np.linalg.eig(S)
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]
# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)
vec = np.array([ eig_pairs[i][1] for i in range(k)])
vec = vec.T # to make eigen vector matrix nxk

# d×k-dimensional eigenvector matrix W.
W=X.T.dot(vec)
Y=X.dot(W)

global pca_X
pca_X = deepcopy(Y)
```

LDA :

- Computing the d-dimensional mean vectors
- Computing the Scatter Matrices
  - Within-class scatter matrix *SW*
  - Between-class scatter matrix *SB*
- Solving the generalized eigenvalue problem for the matrix $S^{-1}W*SB$
- Selecting linear discriminants for the new feature subspace
  - Choosing *k* eigenvectors with the largest eigenvalues
- Transforming the samples onto the new subspace

```
#dimension
n = pca_X.shape[0]
d = pca_X.shape[1]

#Calculate n dimensional means
overall_mean = np.mean(pca_X, axis=0)
mean_vectors = []
for class_no in range(2):
    mean_vectors.append(np.mean(pca_X[class_indices[class_no]], axis=0))


#Calculate within class scatter matrix
S_W = np.zeros((d,d))
for class_no,mean_vector in zip(range(2), mean_vectors):
    scatter_matrix = np.zeros((d,d))
    for row in pca_X[class_indices[class_no]]:
        row, mean_vector = row.reshape(d,1), mean_vector.reshape(d,1)
        scatter_matrix += (row-mean_vector).dot((row-mean_vector).T)
    S_W += scatter_matrix


#Calculate between class scatter matrix
S_B = np.zeros((d,d))
for i,mean_vector in enumerate(mean_vectors):
    n = pca_X[class_indices[i],:].shape[0]
    mean_vector = mean_vector.reshape(d,1)
    overall_mean = overall_mean.reshape(d,1) # make column vector
    S_B += n * (mean_vector - overall_mean).dot((mean_vector - overall_mean).T)

eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))

# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)

# Construct KxD eigenvector matrix W
W = eig_pairs[0][1]

global lda_X
lda_X = np.real(pca_X.dot(W))
```

Gaussian Naive Bayes Classifier :

The Classifier model is defined as :

```
class Model:
    def __init__(self, X, labels):
        self.dataset = X
        self.class_labels = labels
        self.test_class_labels = list(labels)
        self.trainingSet= []
        self.testSet = []
        self.summaries = {}
```

Here summaries is a dictionary object that holds mean and standard deviations for every class.

First the data is divided as test and train and also seperated on the basis of class labels :

```
def k_fold_split(self,splitRatio):
    trainSize = int(len(self.dataset) * splitRatio)
    trainSet = []
    copy = list(self.dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy))
        trainSet.append(copy.pop(index))
        self.test_class_labels.pop(index)
    self.trainingSet = trainSet
    self.testSet = copy

def seperate_by_class(self):
    separated = {}
    for i in range(len(self.dataset)):
        vector = self.dataset[i]
        label = self.class_labels[i]
        if (label not in separated):
            separated[label] = []
        separated[label].append(vector)
    return separated
```

The classifier is then trained, calculation of summaries for each class.

```
def train_classifier(self,data_flag):
    separated = self.seperate_by_class()
    summaries = {}
    for classValue, instances in separated.items():
        if data_flag is 0:
            summaries[classValue] = self.summarize_pca(instances)
        elif data_flag is 1:
            summaries[classValue] = self.summarize_lda(instances)
    self.summaries = summaries
```

Testing phase : For predicting values for the test set, probabilities are calculated using :-

```
def calc_probability(self,x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calc_class_probabilities(self, inputVector):
    probabilities = {}
    for classValue, classSummaries in self.summaries.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            if type(inputVector) == np.float64 :
                x = inputVector
            else:
                x = inputVector[i]
            probabilities[classValue] *= self.calc_probability(x, mean, stdev)
    return probabilities
```

```
def test_classifier(self):
    predictions = []
    for i in range(len(self.testSet)):
        result = self.predict(self.testSet[i])
        predictions.append(result)
    return predictions

def get_accuracy(self,predictions):
    correct = 0
    for i in range(len(self.testSet)):
        if self.test_class_labels[i] == predictions[i]:
            correct += 1
    return (correct/float(len(self.testSet))) * 100.0
```

Results obtained are :

Split of train and test data = 70 % (train)

Test Run 1 :

K = 100

```
Size of train= 560  and test= 240
Accuracy after PCA(K==100):  49.583333333333336
Size of train= 560  and test= 240
Accuracy after LDA:  79.58333333333333
```

Test Run 2:

K = 100

```
In [1]: runfile('/home/levicorpus/Documents/UG3/SMAI/A2/Assign2/Prob3/prob2.py',
wdir='/home/levicorpus/Documents/UG3/SMAI/A2/Assign2/Prob3')
Size of train= 560  and test= 240
Accuracy after PCA(K==100):  55.41666666666667
Size of train= 560  and test= 240
Accuracy after LDA:  77.91666666666667
```

Test Run 1:

K = 500

```
Size of train= 560  and test= 240
Accuracy after PCA(K==500):  88.33333333333333
Size of train= 560  and test= 240
Accuracy after LDA:  95.41666666666667
```

Test Run 2:

K = 500

```
In [7]: runfile('/home/levicorpus/Documents/UG3/SMAI/A2/Assign2/Prob3/prob3.py',
wdir='/home/levicorpus/Documents/UG3/SMAI/A2/Assign2/Prob3')
Reloaded modules: pca, lda
Exception ignored in: <_io.FileIO name='dataset/dorothea_train.data' mode='rb' closefd=True>
ResourceWarning: unclosed file <_io.TextIOWrapper name='dataset/dorothea_train.data' mode='r'
encoding='UTF-8'>
Size of train= 560  and test= 240
Accuracy after PCA(K==500):  88.75
Size of train= 560  and test= 240
Accuracy after LDA:  96.25
```

Observations :

The accuracy for PCA is low, since it doesn’t take into consideration the classification of the data. However using LDA significantly increases accuracy since it considers maximises inter class distance and minimizes intra class distance.
Also as K increases, accuracy is increasing, since the number of dimensions retained for LDA is higher and hence more information is captured for classification.

On changing the split ratio for train vs test data, the accuracy does not significantly vary from previous results

```
In [8]: runfile('/home/levicorpus/Documents/UG3/SMAI/A2/Assign2/Prob3/prob3.py',
wdir='/home/levicorpus/Documents/UG3/SMAI/A2/Assign2/Prob3')
Reloaded modules: pca, lda
Exception ignored in: <_io.FileIO name='dataset/dorothea_train.data' mode='rb' closefd=True>
ResourceWarning: unclosed file <_io.TextIOWrapper name='dataset/dorothea_train.data' mode='r'
encoding='UTF-8'>
Size of train= 400  and test= 400
Accuracy after PCA(K==500):  90.25
Size of train= 400  and test= 400
Accuracy after LDA:  96.75

In [9]: runfile('/home/levicorpus/Documents/UG3/SMAI/A2/Assign2/Prob3/prob3.py',
wdir='/home/levicorpus/Documents/UG3/SMAI/A2/Assign2/Prob3')
Reloaded modules: pca, lda
Exception ignored in: <_io.FileIO name='dataset/dorothea_train.data' mode='rb' closefd=True>
ResourceWarning: unclosed file <_io.TextIOWrapper name='dataset/dorothea_train.data' mode='r'
encoding='UTF-8'>
Size of train= 400  and test= 400
Accuracy after PCA(K==500):  90.25
Size of train= 400  and test= 400
Accuracy after LDA:  96.5
```