ISYE 6740 Homework 5

Katherine Barthelson

100 points + 3 bonus points

1. Comparing classifiers. (65 points)

In lectures, we learn different classifiers. This question is compare them on two datasets. Python users, please feel free to use Scikit-learn, which is a commonly-used and powerful Python library with various machine learning tools. But you can also use other similar libraries in other languages of your choice to perform the tasks.

1. Part One (Divorce classification/prediction). (30 points)

This dataset is about participants who completed the personal information form and a divorce predictors scale. The data is a modified version of the publicly available at https://archive.ics.uci (https://archive.ics.uci). edu/ml/datasets/Divorce+Predictors+data+set (by injecting noise so you will not get the exactly same results as on UCI website). The dataset marriage.csv is contained in the homework folder. There are 170 participants and 54 attributes (or predictor variables) that are all real-valued. The last column of the CSV file is label y (1 means "divorce", 0 means "no divorce"). Each column is for one feature (predictor variable), and each row is a sample (participant). A detailed explanation for each feature (predictor variable) can be found at the website link above. Our goal is to build a classifier using training data, such that given a test sample, we can classify (or essentially predict) whether its label is 0 ("no divorce") or 1 ("divorce"). We are going to compare the following classifiers (Naive Bayes, Logistic Regression, and KNN). Use the first 80% data for training and the remaining 20% for testing. If you use scikit-learn you can use train test split to split the dataset. Remark: Please note that, here, for Naive Bayes, this means that we have to estimate the variance for each individual feature from training data. When estimating the variance, if the variance is zero to close to zero (meaning that there is very little variability in the feature), you can set the variance to be a small number, e.g., = 10−3 . We do not want to have include zero or nearly variance in Naive Bayes. This tip holds for both Part One and Part Two of this question.

In [8]:
```
# Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, precision_
score, recall_score, roc_auc_score, f1_score
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Load data and split into test, train data sets
marriage = pd.read_csv('data/marriage.csv', header=None)
display(marriage.head())
# Predictors
x = marriage.iloc[:,:-1]
# last column = label
y = marriage.iloc[:,-1]
# 80% train, 20% test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

	0	1	2	3	4	5	6	7	8
0	1.560903	3.681587	3.450467	3.211998	-1.203045	0.597706	-0.970093	-0.750970	-0.511495
1	4.153272	5.173858	4.100690	2.580173	3.305788	-1.505512	-0.029398	5.702657	2.230281
2	2.226241	1.575322	2.389117	2.725405	-0.304562	2.832803	1.787779	0.565755	1.328212
3	3.553458	2.859042	2.928414	1.833241	1.271119	4.165213	2.078597	4.506175	2.521628
4	0.506547	1.419223	1.716153	1.319274	2.853840	0.047412	-0.016515	0.620795	1.202992

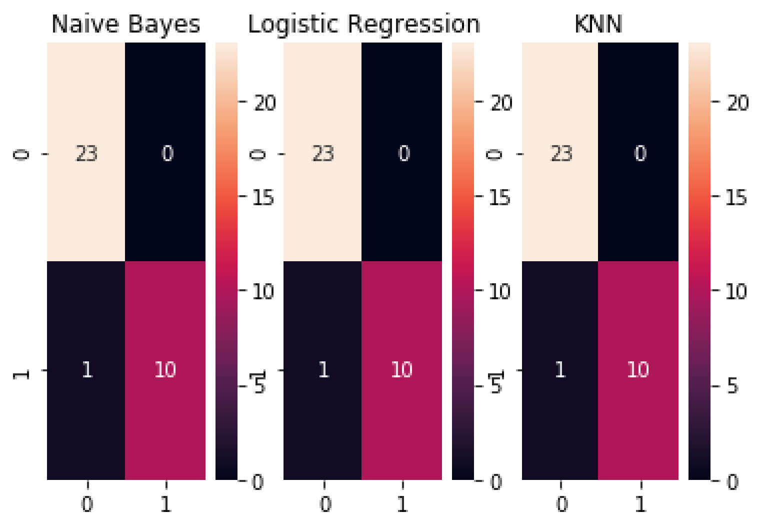
5 rows × 55 columns

(a) (15 points) Report testing accuracy for each of the three classifiers. Comment on their performance: which performs the best and make a guess why they perform the best in this setting.

In [9]:
```
# Naive Bayes - following sklearn documentation
# estimate the variance - if the variance is zero to close to zero set t
he variance to = 10-3
gnb = GaussianNB()
gnb_pred = gnb.fit(x_train, y_train).predict(x_test)
gnb_accuracy = accuracy_score(y_test, gnb_pred)
gnb_conf = confusion_matrix(y_test, gnb_pred)
print("Naive Bayes Accuracy %f" % gnb_accuracy)
print("Number of mislabeled points out of a total %d points : %d" % (x_t
est.shape[0], (y_test != gnb_pred).sum()))

# Logistic Regression
log = LogisticRegression()
log_pred = log.fit(x_train,y_train).predict(x_test)
log_accuracy = accuracy_score(y_test, log_pred)
log_conf = confusion_matrix(y_test, log_pred)
print("Logistic Regression Accuracy %f" % log_accuracy)
print("Number of mislabeled points out of a total %d points : %d" % (x_t
est.shape[0], (y_test != log_pred).sum()))

# KNN
knn = KNeighborsClassifier()
knn_pred = knn.fit(x_train,y_train).predict(x_test)
knn_accuracy = accuracy_score(y_test, knn_pred)
knn_conf = confusion_matrix(y_test, knn_pred)
print("KNN Accuracy %f" % knn_accuracy)
print("Number of mislabeled points out of a total %d points : %d" % (x_t
est.shape[0], (y_test != knn_pred).sum()))

# Plot confusion matrix
figs, axes = plt.subplots(nrows=1, ncols=3)
ax0, ax1, ax2 = axes.flatten()
ax0.set_title('Naive Bayes')
ax1.set_title('Logistic Regression')
ax2.set_title('KNN')
sns.heatmap(gnb_conf, annot=True,ax=ax0)
sns.heatmap(log_conf, annot=True,ax=ax1)
sns.heatmap(knn_conf, annot=True,ax=ax2)
```

```
Naive Bayes Accuracy 0.970588
Number of mislabeled points out of a total 34 points : 1
Logistic Regression Accuracy 0.970588
Number of mislabeled points out of a total 34 points : 1
KNN Accuracy 0.970588
Number of mislabeled points out of a total 34 points : 1
```

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8fa68ee190>
```

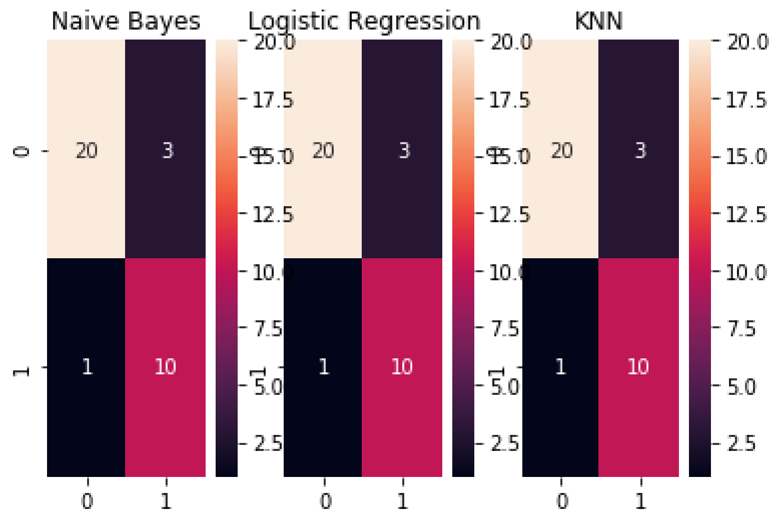


(b) (15 points) Now perform PCA to project the data into two-dimensional space. Build the classifiers (Naive Bayes, Logistic Regression, and KNN) using the two-dimensional PCA results. Plot the data points and decision boundary of each classifier in the two-dimensional space. Comment on the difference between the decision boundary for the three classifiers. Please clearly represent the data points with different labels using different colors.

In [28]:
```
# Convert the dataset into 2D -- using data split previously
x_train2 = x_train.iloc[:,:2]
x_test2 = x_test.iloc[:,:2]

# Build the classifiers on the reduced dataset
# Naive Bayes - following sklearn documentation
# estimate the variance - if the variance is zero to close to zero set t
he variance to = 10-3
gnb2 = GaussianNB()
gnb2_pred = gnb2.fit(x_train2, y_train).predict(x_test2)
gnb2_accuracy = accuracy_score(y_test, gnb2_pred)
gnb2_conf = confusion_matrix(y_test, gnb2_pred)
print("Naive Bayes Accuracy %f" % gnb2_accuracy)
print("Number of mislabeled points out of a total %d points : %d" % (x_t
est2.shape[0], (y_test != gnb2_pred).sum()))

# Logistic Regression
log2 = LogisticRegression()
log2_pred = log2.fit(x_train2,y_train).predict(x_test2)
log2_accuracy = accuracy_score(y_test, log2_pred)
log2_conf = confusion_matrix(y_test, log2_pred)
print("Logistic Regression Accuracy %f" % log2_accuracy)
print("Number of mislabeled points out of a total %d points : %d" % (x_t
est2.shape[0], (y_test != log2_pred).sum()))

# KNN
knn2 = KNeighborsClassifier()
knn2_pred = knn2.fit(x_train2,y_train).predict(x_test2)
knn2_accuracy = accuracy_score(y_test, knn2_pred)
knn2_conf = confusion_matrix(y_test, knn2_pred)
print("KNN Accuracy %f" % knn2_accuracy)
print("Number of mislabeled points out of a total %d points : %d" % (x_t
est2.shape[0], (y_test != knn2_pred).sum()))

# Plot confusion matrix
figs, axes = plt.subplots(nrows=1, ncols=3)
ax0, ax1, ax2 = axes.flatten()
ax0.set_title('Naive Bayes')
ax1.set_title('Logistic Regression')
ax2.set_title('KNN')
sns.heatmap(gnb2_conf, annot=True,ax=ax0)
sns.heatmap(log2_conf, annot=True,ax=ax1)
sns.heatmap(knn2_conf, annot=True,ax=ax2)
```

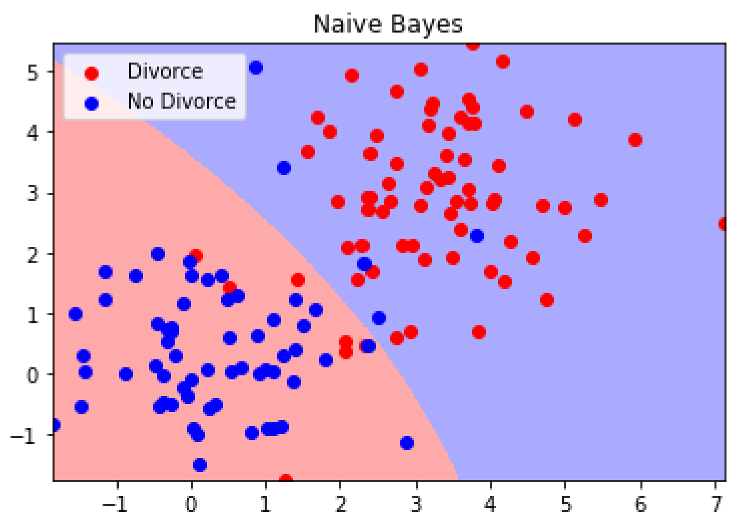
```
Naive Bayes Accuracy 0.882353
Number of mislabeled points out of a total 34 points : 4
Logistic Regression Accuracy 0.882353
Number of mislabeled points out of a total 34 points : 4
KNN Accuracy 0.882353
Number of mislabeled points out of a total 34 points : 4
```

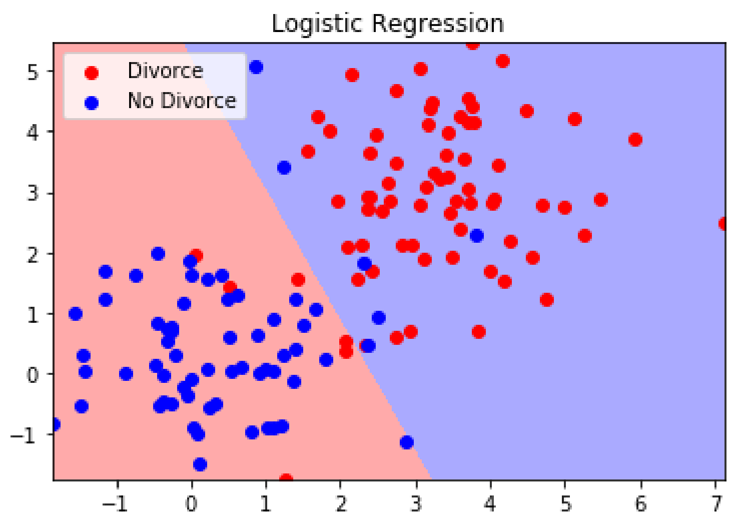
Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8eed7b7850>
```

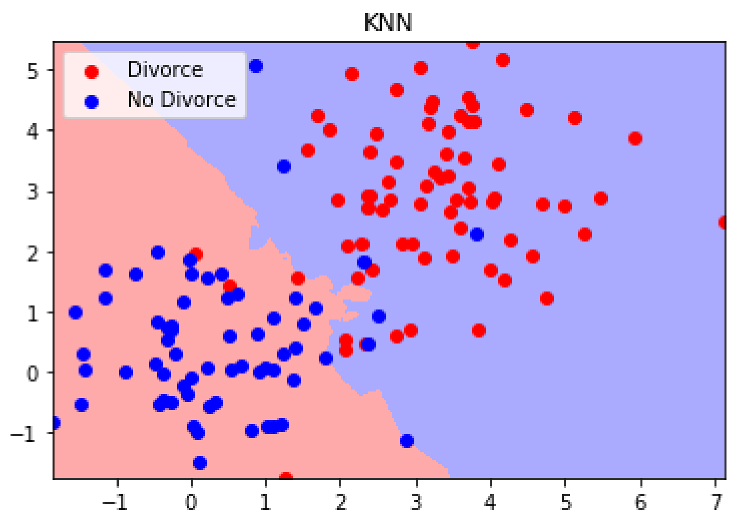


In [11]:
```python
# Define decsion boundary plots - from demo code
def plot_decision_boundary(model, title, x_train, x_test, y_train):
    h = 0.01
    cmap_light = ListedColormap(['#FFAAAA',  '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000',  '#0000FF'])

    x_min = x_train.iloc[:,0].min()
    y_min = x_train.iloc[:,1].min()
    x_max = x_train.iloc[:,0].max()
    y_max = x_train.iloc[:,1].max()

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light, shading='auto')

    # Also plot the training points
    plt.scatter(x_train.loc[y==1,0],x_train.loc[y==1,1], c='red', label
= 'Divorce')
    plt.scatter(x_train.loc[y==0,0],x_train.loc[y==0,1], c='blue',label
= 'No Divorce')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(title)
    plt.legend()
```

In [12]:
```
# Plot the decision boundaries
gnb2_fit = gnb2.fit(x_train2,y_train)
plot_decision_boundary(gnb2_fit, 'Naive Bayes', x_train2, x_test2, y_tra
in)

log2_fit = log2.fit(x_train2,y_train)
plot_decision_boundary(log2_fit, 'Logistic Regression', x_train2, x_test
2, y_train)

knn2_fit = knn2.fit(x_train2,y_train)
plot_decision_boundary(knn2_fit, 'KNN', x_train2, x_test2, y_train)
```







Based on the plots above, the classifiers are very similar in decision boundary. Naives Bayes is a non-linear curve, Logistic Regression is linear, and KNN looks more like a soft classifier. KNN still fails to correctly classify points near the boundary.

2. Part Two (Handwritten digits classification). (35 points)

This question is to compare different classifiers and their performance for multi-class classifications on the complete MNIST dataset at http://yann.lecun.com/exdb/mnist/ (http://yann.lecun.com/exdb/mnist/). You can find the data file mnist 10digits.mat in the homework folder. The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. We will compare KNN, logistic regression, SVM, kernel SVM, and neural networks. 1 • We suggest you to “standardize” the features before training the classifiers, by dividing the values of the features by 255 (thus map the range of the features from [0, 255] to [0, 1]). • You may adjust the number of neighbors K used in KNN to have a reasonable result (you may use cross validation but it is not required; any reasonable tuning to get good result is acceptable). • You may use a neural networks function sklearn.neural network with hidden layer sizes = (20, 10). • For kernel SVM, you may use radial basis function kernel, and a heuristic called “median trick”: choose the parameter of the kernel K(x, x0) = exp{−kx − x 0k 2/(2σ 2)}. Choose the bandwidth as σ = p M/2 where M = the median of {kx i − x jk 2 , 1 ≤ i, j ≤ m0 , i 6= j} for pairs of training samples. Here you can randomly choose m0 = 1000 samples from training data to use for the “median trick”1 . • For KNN and SVM, you can randomly downsample the training data to size m = 5000, to improve computation efficiency. Train the classifiers on training dataset and evaluate on the test dataset.

In [26]:
```
# Load data mnist_10digits.mat
from scipy.io import loadmat
mnist = loadmat('data/mnist_10digits.mat')
xtrain = pd.DataFrame(mnist['xtrain'])
xtest = pd.DataFrame(mnist['xtest'])
ytrain = pd.DataFrame(mnist['ytrain']).T
ytest = pd.DataFrame(mnist['ytest']).T
#display(xtrain)
#display(xtest)
#display(ytrain)
#display(ytest)
```

(a) (25 points) Report confusion matrix, precision, recall, and F-1 score for each of the classifiers. For precision, recall, and F-1 score of each classifier, we will need to report these for each of the digits. So you can create a table for this. For this question, each of the 5 classifier, KNN, logistic regression, SVM, kernel SVM, and neural networks, accounts for 10 points.

In [25]:
```python
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
# KNN, logistic regression, SVM, kernel SVM, and neural networks
# KNN
knn = KNeighborsClassifier(n_jobs=-1)
knn_pred = knn.fit(xtrain,ytrain).predict(xtest)
knn_accuracy = accuracy_score(ytest, knn_pred)
knn_conf = confusion_matrix(ytest, knn_pred)
knn_prec = precision_score(ytest, knn_pred,average=None)
knn_recall = recall_score(ytest, knn_pred,average=None)
knn_f1 = f1_score(ytest, knn_pred,average=None)
print("KNN ------------------------------")
print("Accuracy %f" % knn_accuracy)
print("Precision")
print(knn_prec)
print("Recall")
print(knn_recall)
print("F-1 Score")
print(knn_f1)
#print("Number of mislabeled points out of a total %d points : %d" % (xt
est.shape[0], (ytest != knn_pred).sum()))

# Logistic Regression
log = LogisticRegression(max_iter=20,tol=0.001)
log_pred = log.fit(xtrain,ytrain).predict(xtest)
log_accuracy = accuracy_score(ytest, log_pred)
log_conf = confusion_matrix(ytest, log_pred)
log_prec = precision_score(ytest, knn_pred,average=None)
log_recall = recall_score(ytest, knn_pred,average=None)
log_f1 = f1_score(ytest, knn_pred,average=None)
print("Logistic Regression ------------------------------")
print("Accuracy %f" % log_accuracy)
print("Precision")
print(log_prec)
print("Recall")
print(log_recall)
print("F-1 Score")
print(log_f1)
#print("Number of mislabeled points out of a total %d points : %d" % (xt
est.shape[0], (ytest != log_pred).sum()))

# SVM
svm = SVC(kernel='linear',max_iter=200)
svm_pred = svm.fit(xtrain,ytrain).predict(xtest)
svm_accuracy = accuracy_score(ytest, svm_pred)
svm_conf = confusion_matrix(ytest, svm_pred)
svm_prec = precision_score(ytest, svm_pred,average=None)
svm_recall = recall_score(ytest, svm_pred,average=None)
svm_f1 = f1_score(ytest, svm_pred,average=None)
print("SVM ------------------------------")
print("Accuracy %f" % svm_accuracy)
print("Precision")
print(svm_prec)
print("Recall")
print(svm_recall)
print("F-1 Score")
print(svm_f1)

# kernel SVM
ksvm = SVC(kernel='poly', degree=8,max_iter=200)
ksvm_pred = ksvm.fit(xtrain,ytrain).predict(xtest)
ksvm_accuracy = accuracy_score(ytest, ksvm_pred)
ksvm_conf = confusion_matrix(ytest, ksvm_pred)
ksvm_prec = precision_score(ytest, ksvm_pred,average=None)
ksvm_recall = recall_score(ytest, ksvm_pred,average=None)
ksvm_f1 = f1_score(ytest, ksvm_pred,average=None)
print("Kernel SVM ------------------------------")
print("Accuracy %f" % ksvm_accuracy)
print("Precision")
print(ksvm_prec)
print("Recall")
print(ksvm_recall)
print("F-1 Score")
print(ksvm_f1)

# Neural Network
nn = MLPClassifier(random_state=1, max_iter=10)
nn_pred = nn.fit(xtrain,ytrain).predict(xtest)
nn_accuracy = accuracy_score(ytest, nn_pred)
nn_conf = confusion_matrix(ytest, nn_pred)
nn_prec = precision_score(ytest, nn_pred,average=None)
nn_recall = recall_score(ytest, nn_pred,average=None)
nn_f1 = f1_score(ytest, nn_pred,average=None)
print("Neural Network ------------------------------")
print("Accuracy %f" % nn_accuracy)
print("Precision")
print(nn_prec)
print("Recall")
print(nn_recall)
print("F-1 Score")
print(nn_f1)
```

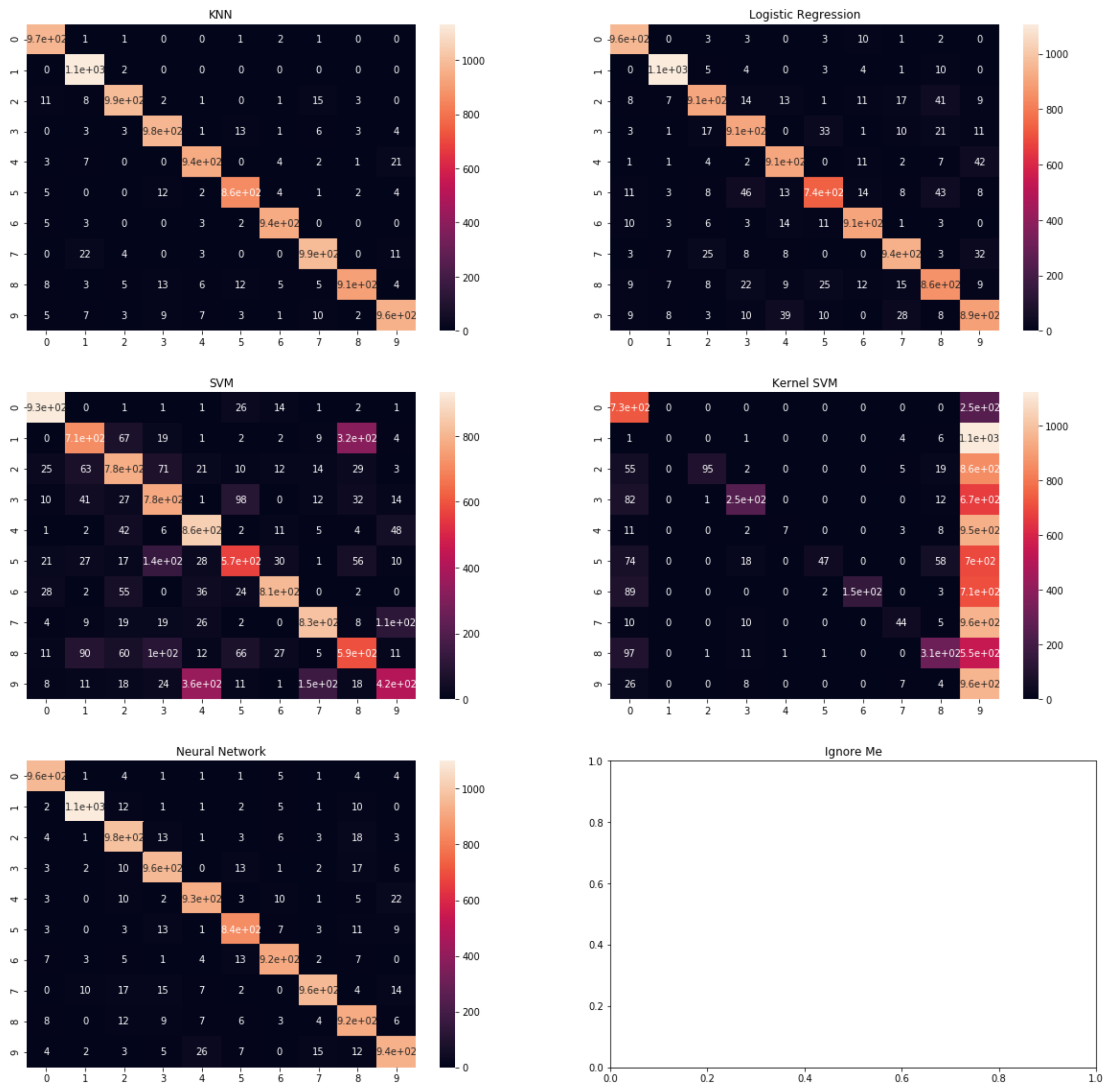
```
/Users/KatieBarthelson/opt/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:6: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_sample
s, ), for example using ravel().


KNN ------------------------------
Accuracy 0.968800
Precision
[0.96340257 0.95450716 0.98216056 0.96442688 0.9762151  0.96528555
 0.98130841 0.96108949 0.98809524 0.95626243]
Recall
[0.99387755 0.99823789 0.96027132 0.96633663 0.96130346 0.96636771
 0.98643006 0.96108949 0.93737166 0.95341923]
F-1 Score
[0.97840281 0.97588286 0.9710926  0.96538081 0.9687019  0.96582633
 0.98386257 0.96108949 0.96206533 0.95483871]

/Users/KatieBarthelson/opt/anaconda3/lib/python3.7/site-packages/sklear
n/utils/validation.py:760: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n
_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/KatieBarthelson/opt/anaconda3/lib/python3.7/site-packages/sklear
n/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to co
nverge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

Logistic Regression ------------------------------
Accuracy 0.914100
Precision
[0.96340257 0.95450716 0.98216056 0.96442688 0.9762151  0.96528555
 0.98130841 0.96108949 0.98809524 0.95626243]
Recall
[0.99387755 0.99823789 0.96027132 0.96633663 0.96130346 0.96636771
 0.98643006 0.96108949 0.93737166 0.95341923]
F-1 Score
[0.97840281 0.97588286 0.9710926  0.96538081 0.9687019  0.96582633
 0.98386257 0.96108949 0.96206533 0.95483871]
```

```
/Users/KatieBarthelson/opt/anaconda3/lib/python3.7/site-packages/sklear
n/utils/validation.py:760: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n
_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/KatieBarthelson/opt/anaconda3/lib/python3.7/site-packages/sklear
n/svm/_base.py:231: ConvergenceWarning: Solver terminated early (max_it
er=200).  Consider pre-processing your data with StandardScaler or MinM
axScaler.
  % self.max_iter, ConvergenceWarning)

SVM ------------------------------
Accuracy 0.728000
Precision
[0.8962536  0.74399164 0.71926606 0.67332754 0.640625   0.70136307
 0.89317181 0.81152344 0.55743879 0.6737013 ]
Recall
[0.95204082 0.62731278 0.75968992 0.76732673 0.87678208 0.63452915
 0.84655532 0.80836576 0.60780287 0.41129832]
F-1 Score
[0.92330529 0.68068834 0.73892554 0.71726053 0.74032674 0.66627428
 0.86923901 0.80994152 0.58153242 0.51076923]

/Users/KatieBarthelson/opt/anaconda3/lib/python3.7/site-packages/sklear
n/utils/validation.py:760: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n
_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/KatieBarthelson/opt/anaconda3/lib/python3.7/site-packages/sklear
n/svm/_base.py:231: ConvergenceWarning: Solver terminated early (max_it
er=200).  Consider pre-processing your data with StandardScaler or MinM
axScaler.
  % self.max_iter, ConvergenceWarning)
/Users/KatieBarthelson/opt/anaconda3/lib/python3.7/site-packages/sklear
n/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. U
se `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/KatieBarthelson/opt/anaconda3/lib/python3.7/site-packages/sklear
n/neural_network/_multilayer_perceptron.py:934: DataConversionWarning:
A column-vector y was passed when a 1d array was expected. Please chang
e the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
Kernel SVM ----------------------------
Accuracy 0.259800
Precision
[0.620954   0.         0.97938144 0.82608696 0.875      0.94
 1.         0.6984127  0.73004695 0.12472506]
Recall
[0.74387755 0.         0.09205426 0.24455446 0.00712831 0.05269058
 0.16075157 0.04280156 0.31930185 0.95540139]
F-1 Score
[0.67688022 0.         0.16829052 0.37738732 0.01414141 0.09978769
 0.27697842 0.08065995 0.44428571 0.22064546]
Neural Network ----------------------------
Accuracy 0.949200
Precision
[0.96572581 0.98303571 0.9280303  0.94094488 0.95071869 0.94394619
 0.96117524 0.96770938 0.91261172 0.93593594]
Recall
[0.97755102 0.97004405 0.9496124  0.94653465 0.94297352 0.94394619
 0.95615866 0.93287938 0.94353183 0.92666006]
F-1 Score
[0.97160243 0.97649667 0.93869732 0.94373149 0.94683027 0.94394619
 0.95866039 0.94997524 0.92781424 0.9312749 ]

/Users/KatieBarthelson/opt/anaconda3/lib/python3.7/site-packages/sklear
n/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Sto
chastic Optimizer: Maximum iterations (10) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
In [24]: # Plot confusion matrix
         figs, axes = plt.subplots(nrows=3, ncols=2,figsize=(20,20))
         ax0, ax1, ax2, ax3, ax4, ax5 = axes.flatten()
         ax0.set_title('KNN')
         ax1.set_title('Logistic Regression')
         ax2.set_title('SVM')
         ax3.set_title('Kernel SVM')
         ax4.set_title('Neural Network')
         ax5.set_title('Ignore Me')
         sns.heatmap(knn_conf, annot=True,ax=ax0)
         sns.heatmap(log_conf, annot=True,ax=ax1)
         sns.heatmap(svm_conf, annot=True,ax=ax2)
         sns.heatmap(ksvm_conf, annot=True,ax=ax3)
         sns.heatmap(nn_conf, annot=True,ax=ax4)
```

```
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8f335ee110>
```



(b) (10 points) Comment on the performance of the classifier and give your explanation why some of them perform better than the others.

Of all the classifiers attempted, KNN had the highest accuracy followed by the Neural Network and Logistic Regression. This implies non-linear decision boundaries for more complex shapes perform better, as the SVM and kernel SVM had poor performance.

2. Naive Bayes for spam filtering. (35 points)

In this problem, we will use the Naive Bayes algorithm to fit a spam filter by hand. This will enhance your understanding to Bayes classifier and build intuition. This question does not involve any programming but only derivation and hand calculation. Spam filters are used in all email services to classify received emails as “Spam” or “Not Spam”. A simple approach involves maintaining a vocabulary of words that commonly occur in “Spam” emails and classifying an email as “Spam” if the number of words from the dictionary that are present in the email is over a certain threshold. We are given the vocabulary consists of 15 words V = {secret, offer, low, price, valued, customer, today, dollar, million, sports, is, for, play, healthy, pizza}. We will use Vi to represent the ith word in V . As our training dataset, we are also given 3 example spam messages, • million dollar offer • secret offer today • secret is secret and 4 example non-spam messages • low price for valued customer • play secret sports today • sports is healthy • low price pizza 1Garreau, Damien, Wittawat Jitkrittum, and Motonobu Kanagawa. ”Large sample analysis of the median heuristic.” arXiv preprint arXiv:1707.07269 (2017). 2 Recall that the Naive Bayes classifier assumes the probability of an input depends on its input feature. The feature for each sample is defined as x (i) = [x (i) 1 , x (i) 2 , . . . , x (i) d] T , i = 1, . . . , m and the class of the ith sample is y (i) . In our case the length of the input vector is d = 15, which is equal to the number of words in the vocabulary V . Each entry x (i) j is equal to the number of times word Vj occurs in the i-th message.

V = {'secret', 'offer', 'low', 'price', 'valued', 'customer', 'today', 'dollar', 'million', 'sports', 'is', 'for', 'play', 'healthy', 'pizza'}

1. (5 points) Calculate class prior P(y = 0) and P(y = 1) from the training data, where y = 0 corresponds to spam messages, and y = 1 corresponds to non-spam messages. Note that these class prior essentially corresponds to the frequency of each class in the training sample. Write down the feature vectors for each spam and non-spam messages.

Given there are 3 cases where y = 0 (spam) and 4 cases where y = 1 (non-spam):

$$P(y=0) = 3/7 = .428$$
$$P(y=1) = 4/7 = .572$$

Spam

---	secret	offer	low	price	valued	customer	today	dollar	million	sports	is	for	play	healthy	pizza
million dollar offer	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0
secret offer today	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
secret is secret	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Not Spam

---	secret	offer	low	price	valued	customer	today	dollar	million	sports	is	for	play	healthy	pizza
low price for valued customer	0	0	1	1	1	1	0	0	0	0	0	1	0	0	0
play secret sports today	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0
sports is healthy	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0
low price pizza	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1

2. (15 points) In the Naive Bayes model, assuming the keywords are independent of each other (this is a simplification), the likelihood of a sentence with its feature vector x given a class c is given by

$$P(x|y=c) = \prod_{k=1}^{n} \theta_{c,k}^{x_k}$$

where 0 ≤ θc,k ≤ 1 is the probability of word k appearing in class c, which satisfies $\sum_{k=1}^{n} \theta_{c,k} = 1, \forall c$

Given this, the complete log-likelihood function for our training data is given by (θ0,1, . . . , θ0,d, θ1,1, . . . , θ1,d) = Xm i=1 X d k=1 x (i) k log θy(i),k (In this example, m = 7.) Calculate the maximum likelihood estimates of θ0,1, θ0,7, θ1,1, θ1,15 by maximizing the log-likelihood function above. (Hint: We are solving a constrained maximization problem and you will need to introduce Lagrangian multipliers and consider the Lagrangian function.)

Maximize $P(x|y{=}c)$ with respect to θc,k

$$P(x|y=c) = \prod_{i=1}^{m} P(y=1)P(X|y=c)$$

$$= \prod_{i=1}^{m} P(y) \prod_{k=1}^{n} \theta_{c,k}^{x_k}$$

Take the log of the previous step

$$= \sum_{i=1}^{m} \log\left(P(y)\prod_{k=1}^{n}\theta_{c,k}^{x_k}\right)$$

$$= \sum_{i=1}^{m} \log\left((P(y=1)) * (P(y=0)) * \prod_{k=1}^{n}\theta_{c,k}^{x_k}\right) =$$

$$= \sum_{i=1}^{m}\left(\log\left((P(y=1)) * (P(y=0))\right) + \sum_{k=1}^{n}\log\theta_{c,k}^{x_k}\right)$$

where $\sum_{k=1}^{n}\theta_{c,k} = 1$

Maximize this derivation using the Lagrangian function

$$L(P(x|y=c),\theta_{c,k}) = \sum_{i=1}^{m}\left(\log\left((P(y=1)) * (P(y=0))\right) + \sum_{k=1}^{n}\log\theta_{c,k}^{x_k}\right) + \lambda * \left(\sum_{k=1}^{n}\theta_{c,k} - 1\right)$$

Take the first derivative with respect to θc,k and set it to zero:

$$= \frac{\partial L(P(x|y=c),\theta_{c,k})}{\partial\theta_{c,k}} = 0$$

$$0 = \frac{1}{\theta_{c,k}}\sum_{i=1}^{m} x_k * 1 * (y=c) + \lambda$$

Isolate θc,k

$$\theta_{c,k} = -\frac{\sum_{i=1}^{m} x_k * 1 * (y=c)}{\lambda}$$

Substitute θc,k = 1

$$1 = -\frac{\sum_{i=1}^{m} x_k * 1 * (y=c)}{\lambda} = \frac{-\sum_{k=1}^{n}\sum_{i=1}^{m} x_k * 1 * (y=c)}{\lambda}$$

$$\lambda = -\sum_{k=1}^{n}\sum_{i=1}^{m} x_k * 1 * (y=c)$$

Substitute again for the final result

$$\theta_{c,k} = \frac{\sum_{i=1}^{m} x_k * 1 * (y=c)}{\sum_{k=1}^{n}\sum_{i=1}^{m} x_k * 1 * (y=c)}$$

Calculate log likelihood for training data (sum frequency of word where y=c / sum all words where y=c)

$$\theta[0,1] = P(secret|spam) = 3/9 = 0.333$$
$$\theta[0,7] = P(today|spam) = 1/9 = 0.111$$
$$\theta[1,1] = P(secret|notspam) = 1/15 = 0.067$$
$$\theta[1,15] = P(pizza|notspam) = 1/15 = 0.067$$

3. (15 points) Given a test message “today is secret”, using the Naive Bayes classier that you have trained

in Part (a)-(b), to calculate the posterior and decide whether it is spam or not spam.

---	secret	offer	low	price	valued	customer	today	dollar	million	sports	is	for	play	healthy	pizza
today is secret	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0

$$\theta[0,7] = P(today|spam) = 1/9 = 0.111$$
$$\theta[0,11] = P(is|spam) = 1/9 = 0.111$$
$$\theta[0,1] = P(secret|spam) = 3/9 = 0.333$$
$$P(Y=0|X=\text{"}\ todayissecret\ \text{"}) = \frac{P(X=\text{"}\ todayissecret\ \text{"}\ |Y=0) * P(Y=0)}{P(X=\text{"}\ todayissecret\ \text{"})}$$
$$= \frac{P(X_1 = today|Y=0) * P(X_2 = is|Y=0) * P(X_3 = secret|Y=0) * P(Y=0)}{P(X_1 = today) * P(X_2 = is) * P(X_3 = secret)}$$
$$P(Y=0|X=\text{"}\ todayissecret\ \text{"}) = (0.111) * (0.111) * (0.333) * .428 = 0.0041 * .428 = 0.00175$$
$$\theta[1,7] = P(today|notspam) = 1/15 = 0.0667$$
$$\theta[1,11] = P(is|notspam) = 1/15 = 0.0667$$
$$\theta[1,1] = P(secret|notspam) = 1/15 = 0.0667$$
$$P(Y=1|X=\text{"}\ todayissecret\ \text{"}) = \frac{P(X=\text{"}\ todayissecret\ \text{"}\ |Y=1) * P(Y=1)}{P(X=\text{"}\ todayissecret\ \text{"})}$$
$$= \frac{P(X_1 = today|Y=1) * P(X_2 = is|Y=1) * P(X_3 = secret|Y=1) * P(Y=1)}{P(X_1 = today) * P(X_2 = is) * P(X_3 = secret)}$$
$$P(Y=1|X=\text{"}\ todayissecret\ \text{"}) = (0.0667) * (0.0667) * (0.0667) * .572 = 0.0003 * .572 = 0.000169$$

Because P(Y=0|X="today is secret") > P(Y=1|X="today is secret"), the test message would be classified as spam.

3. Neural networks. (Bonus: 3 points)

Consider a simple two-layer network in the lecture slides. Given m training data (x i , yi), i = 1, . . . , m, the cost function used to training the neural networks (w, α, β) = Xm
i=1
(y
i − σ(w
T
z
i
))2
where σ(x) = 1/(1 + e
−x
) is the sigmoid function, z
i
is a two-dimensional vector such that z
i
1 = σ(α
T x
i
),
and z
i
2 = σ(β
T x
i
). Show the that the gradient is given by
∂ (w, α, β) ∂w = − Xm i=1 2(y i − σ(u i))σ(u i)(1 − σ(u i))z i , where u i = w T z i . Also show the gradient of `(w, α, β) with respect to α and β and write down their expression. 3

No time :(

```
In [ ]:
```