Assignment 3 - Supervised Learning

Iuliia Oblasova

Netid: io26

In [386]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
import plotly.plotly as py
import matplotlib.pyplot as plt

from scipy import interp
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.metrics import roc_curve, precision_recall_curve, auc, accu
racy_score
from sklearn.linear_model import LogisticRegression, LogisticRegressionC
V
from matplotlib.colors import ListedColormap
```

1

[40 points] From theory to practice: classification through logistic regression

Introduction

For this problem you will derive, implement through gradient descent, and test the performance of a logistic regression classifier for a binary classification problem.

In this case, we'll assume our logistic regression problem will be applied to a two dimensional feature space. Our logistic regression model is:

$$f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}_i)$$

where the sigmoid function is defined as $\sigma(x) = \frac{e^x}{1-e^x}$. Also, since this is a two-dimensional problem, we define $\mathbf{w}^T \mathbf{x}_i = w_0 x_{i,0} + w_1 x_{i,1} + w_2 x_{i,2}$ and here, $x_{i,0} \triangleq 1$

As in class, we will interpret the response of the logistic regression classifier to be the likelihood of the data given the model. For one sample, $(y_i, \mathbf{x_i})$, this is given as:

$$P(Y = y_i | X = x_i) = f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}_i)$$

Find the cost function that we can use to choose the model parameters, $\mathbf{w}$, that best fit the training data.

(a) What is the likelihood function of the data that we will wish to maximize?
For questions **a) - d).** was used materials from Coursera: https://www.coursera.org/lecture/neural-networks-deep-learning/explanation-of-logistic-regression-cost-function-optional-SmIbQ (https://www.coursera.org/lecture/neural-networks-deep-learning/explanation-of-logistic-regression-cost-function-optional-SmIbQ)
The likelihood function for logistic regression is a sigmoid function $h_w(x)$:

$$h_w(x) = \frac{1}{1 + e^{-\mathbf{w^T x_i}}}$$

(b) Since a logarithm is a monotonic function, maximizing the $f(x)$ is equivalent to maximizing $\ln[f(x)]$. Express part (a) as a cost function of the model parameters, $C(\mathbf{w})$, that is the negative of the logarithm of (a).

$$C(\mathbf{w}) = -ln \frac{e^{\mathbf{w^T x_i}}}{1 + e^{\mathbf{w^T x_i}}}) = ln \frac{1 + e^{\mathbf{w^T x_i}}}{e^{\mathbf{w^T x_i}}}$$

(c) Calculate the gradient of the cost function with respect to the model parameters $\nabla_{\mathbf{w}} C(\mathbf{w})$. Express this in terms of the partial derivatives of the cost function with respect to each of the parameters, e.g. $\nabla_{\mathbf{w}} C(\mathbf{w}) = \left[\frac{\partial C}{\partial w_0}, \frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2} \right]$.

Let $\mathbf{w}^T \mathbf{x}_i = w_0 x_{i,0} + w_1 x_{i,1} + w_2 x_{i,2} = z$, then

$\frac{\partial C}{\partial w_0} = \frac{e^z}{1+e^z} \cdot \frac{e^z \cdot e^z - (1+e^z) \cdot e^z}{e^{2z}} = \frac{e^z}{1+e^z} \cdot \frac{e^{2z} - e^z - e^{2z}}{e^{2z}} = -\frac{1}{1+e^z} = -\frac{1}{1+e^{\mathbf{w}^\mathbf{T}\mathbf{x_i}}}$,
$\frac{\partial C}{\partial w_1} = \frac{e^z}{1+e^z} \cdot \frac{e^z \cdot e^z \cdot x_{i1} - (1+e^z) \cdot e^z \cdot x_{i1}}{e^{2z}} = \frac{e^z}{1+e^z} \cdot \frac{e^z \cdot x_{i1} \cdot [e^z - 1 - e^z]}{e^{2z}} = -\frac{x_{i1}}{1+e^z} = -\frac{x_{i1}}{1+e^{\mathbf{w}^\mathbf{T}\mathbf{x_i}}}$,
$\frac{\partial C}{\partial w_2} = \frac{e^z}{1+e^z} \cdot \frac{e^z \cdot e^z \cdot x_{i2} - (1+e^z) \cdot e^z \cdot x_{i2}}{e^{2z}} = \frac{e^z}{1+e^z} \cdot \frac{e^z \cdot x_{i2} \cdot [e^z - 1 - e^z]}{e^{2z}} = -\frac{x_{i2}}{1+e^z} = -\frac{x_{i2}}{1+e^{\mathbf{w}^\mathbf{T}\mathbf{x_i}}}$, therefore

$$\nabla_\mathbf{w} C(\mathbf{w}) = \left[-\frac{1}{1+e^{\mathbf{w}^\mathbf{T}\mathbf{x_i}}}, -\frac{x_{i1}}{1+e^{\mathbf{w}^\mathbf{T}\mathbf{x_i}}}, -\frac{x_{i2}}{1+e^{\mathbf{w}^\mathbf{T}\mathbf{x_i}}}\right],$$

for $x_{i,j}$ given i - an observation and j is a parameter.

(d) Write out the gradient descent update equation, assuming η represents the learning rate.

$$\nabla_\mathbf{w} C(\mathbf{w}) = \left[\mathbf{w}^\mathbf{T}\mathbf{x_i} - \eta\frac{1}{1+e^{\mathbf{w}^\mathbf{T}\mathbf{x_i}}}, \mathbf{w}^\mathbf{T}\mathbf{x_i} - \eta\frac{x_{i1}}{1+e^{\mathbf{w}^\mathbf{T}\mathbf{x_i}}}, \mathbf{w}^\mathbf{T}\mathbf{x_i} - \eta\frac{x_{i2}}{1+e^{\mathbf{w}^\mathbf{T}\mathbf{x_i}}}\right],$$

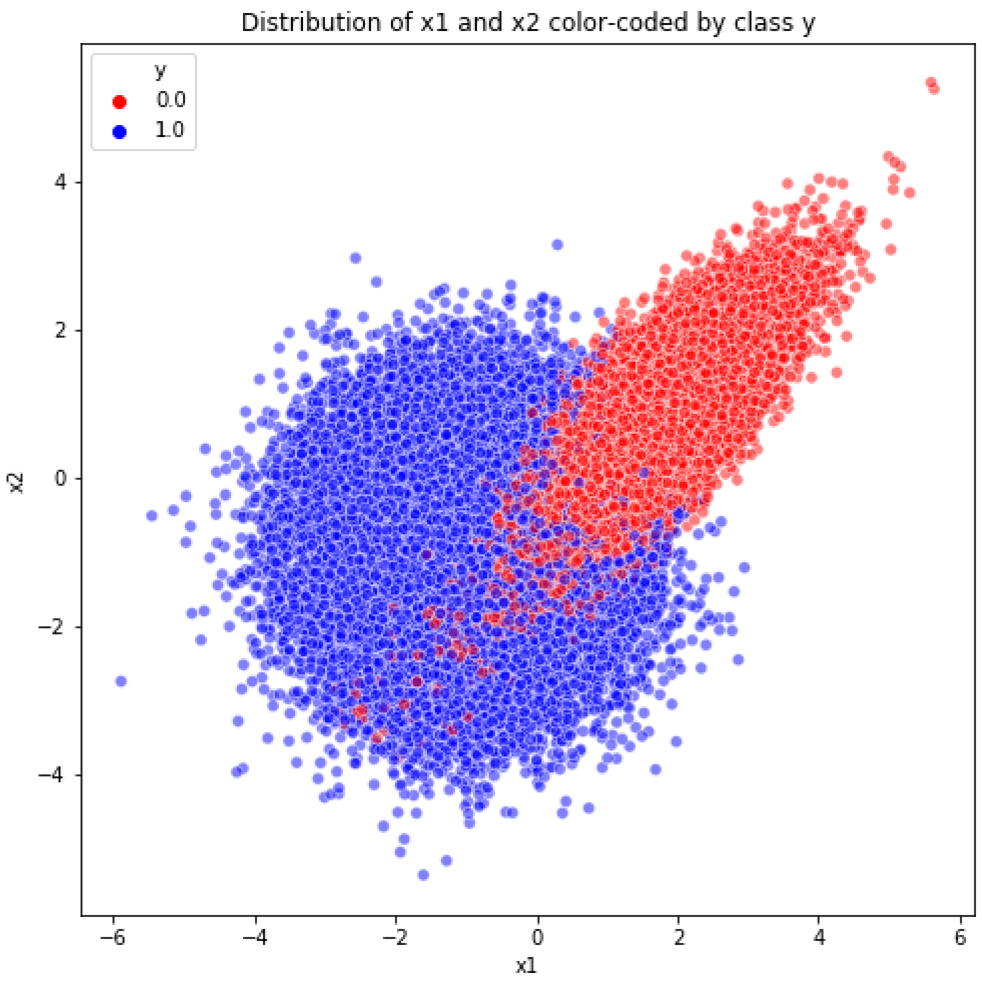
where η is a learning rate.

Prepare and plot your data

(e) Load the data and scatter plot the data by class. In the data folder in the same directory of this notebook, you'll find the data in `A3_Q1_data.csv`. This file contains the binary class labels, y, and the features x_1 and x_2. Comment on the data: do the data appear separable? Why might logistic regression be a good choice for these

In [38]:
```
data=pd.read_csv("A3_Q1_data.csv")
```

```
In [61]: plt.figure(figsize=(8, 8))
         col_list = ["red","blue"]
         sns.set_palette(col_list)
         data_plot = sns.scatterplot(x=data.x1, y=data.x2, hue=data.y, alpha=0.5)
         .set_title("Distribution of x1 and x2 color-coded by class y")
         plt.xlabel('x1')
         plt.ylabel('x2')

Out[61]: Text(0,0.5,'x2')
```



The data above appears to be easily separable. Logistic regression is a good choice because an outcome variable y is a binary variable.

(f) Do the data require any preprocessing due to missing values, scale differences, etc? If so, how did you remediate this?

```
In [40]: data.isnull().sum()

Out[40]: x1     0
         x2     0
         y      0
         dtype: int64
```

Data has no missing values. Without a codebook, we cannot define whether the data should be scaled. From the first view on the graphs above, data does not require any scaling.

(g) Create a function or class to implement your logistic regression. It should take as inputs the model parameters, $\mathbf{w} = [w_0, w_1, w_2]$, and output the class confidence probabilities, $P(Y = y_i | X = x_i)$.

In [371]:
```
# Reference: https://medium.com/@martinpella/logistic-regression-from-sc
ratch-in-python-124c5636b8ac
def sigmoid(x):
    return np.exp(x)/(1+np.exp(x))

def prob(w,data):
        return sigmoid(w[0]+w[1]*data.x1+w[2]*data.x2)
    #Returns the probability of data to belong to a certain class.
```

(h) Create a function that computes the cost function $C(\mathbf{w})$ for a given dataset and corresponding class labels.

In [286]:
```
def cost(w,data):
    prob2=prob(w,data)
    cost = np.where(data.y==1, np.log(prob2),np.log(1-prob2)).sum()
    return -cost*(1/len(data))
```

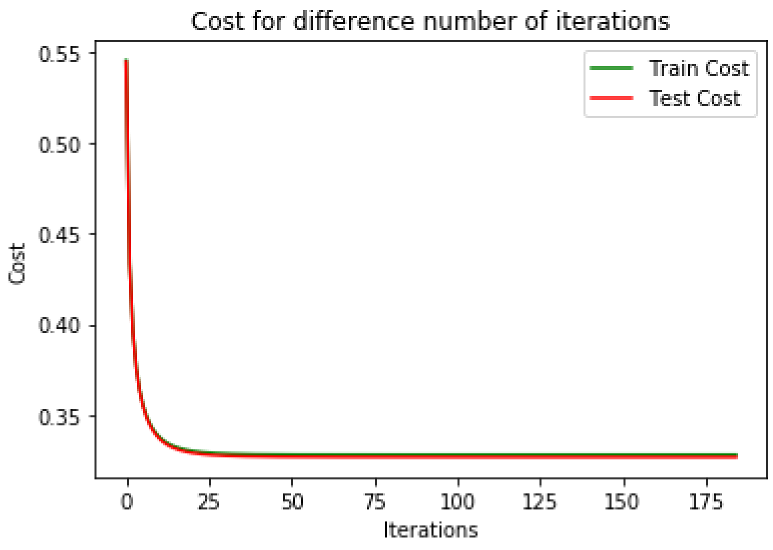
i). Create a function or class to run gradient descent on your training data. We'll refer to this as "batch" gradient descent since it takes into account the gradient based on all our data at each iteration (or "epoch") of the algorithm. Divide your data into a training and testing set where the test set accounts for 30 percent of the data and the training set the remaining 70 percent. In doing this we'll need to make some assumptions / experiment with the following:

The initialization of the algorithm - what should you initialize the model parameters to? For this, randomly initialize the weights to a different values between 0 and 1. The learning rate - how slow/fast should the algorithm proceed in the direction opposite the gradient? This you will experiment with. Stopping criteria - when should the algorithm be finished searching for the optimum? Set this to be when the cost function changes by no more than 10^{-6} between iterations. Since we have a weight vector, you can compute this by seeing if the L2 norm of the weight vector changes by no more than 10^{-6} between iterations.

```
In [355]: def gradient(data,lr):

              "caculates gradient for the respective LR"
              allweights=[]
              random_weights=np.random.random((1,3))
              wold=random_weights[0]
              wnew=random_weights[0]
              x0=np.ones((data.shape[0],), dtype=int)

              diff=100

              #print(1)

              while diff>10e-6:
                  wold=wnew

                  sig=prob(wold,data)
                  summation=np.array([np.dot((sig-data.y),x0),np.dot((sig-data.y),
          data.x1),np.dot((sig-data.y),data.x2)])
                  wnew=wold-(summation*lr)/data.shape[0]
                  diff=np.linalg.norm(wnew - wold)
                  allweights.append(wnew.tolist())

              return allweights,wold
```

(j) At each step in the gradient descent algorithm it will produce updated parameter estimates. For each set of estimates, calculate the cost function for both the training and the test data.

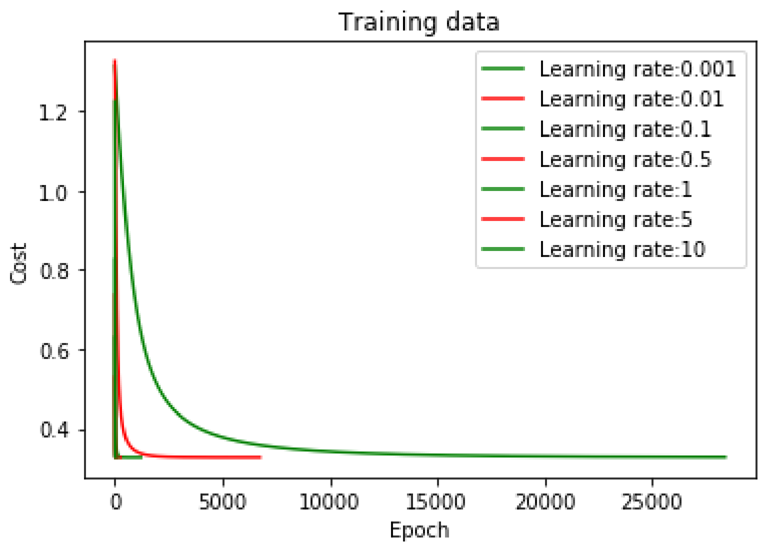
In [356]:
```
allweights,wold = gradient(train,1)

train_cost= []
test_cost= []
for w in allweights:
    current_train=cost(w,train)
    current_test=cost(w,test)
    train_cost.append(current_train)
    test_cost.append(current_test)

plt.plot(train_cost,label='Train Cost')
plt.plot(test_cost,label='Test Cost')
plt.legend()
plt.title("Cost for difference number of iterations")
plt.xlabel("Epoch")
plt.ylabel("Cost")
pass
```

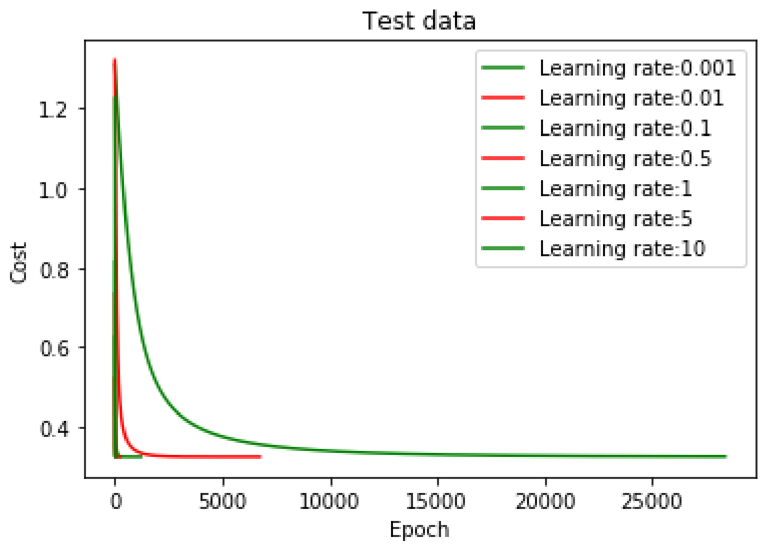


(k) Show this process for different learning rates by plotting the resulting cost as a function of iteration (or "epoch"). What is the impact that each parameter has on the process and the results? What choices did you make in your chosen approach and why? Use the parameter you choose here for the learning rate for the remainder of this question.

In [359]:
```
weights=[]
lr=[0.001,0.01,0.1,0.5,1,5,10]

#Training data
for i in lr:
    x,y = gradient(train,i)
    weights.append(x)
    train_cost = []
    for w in x:
        current_train=cost(w,train)
        train_cost.append(current_train)
    plt.plot(train_cost,label=("Learning rate:"+str(i)))
plt.legend()
plt.title("Training data")
plt.xlabel("Epoch")
plt.ylabel("Cost")
plt.show()
plt.grid()
#Test data
for j,i in enumerate(lr):
    test_cost = []
    for w in weights[j]:
        current_test=cost(w,test)
        test_cost.append(current_test)
    plt.plot(test_cost,label=("Learning rate:"+str(i)))
plt.legend()
plt.title("Test data")
plt.xlabel("Epoch")
plt.ylabel("Cost")
plt.grid()
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3: Runtime
Warning:

divide by zero encountered in log
```

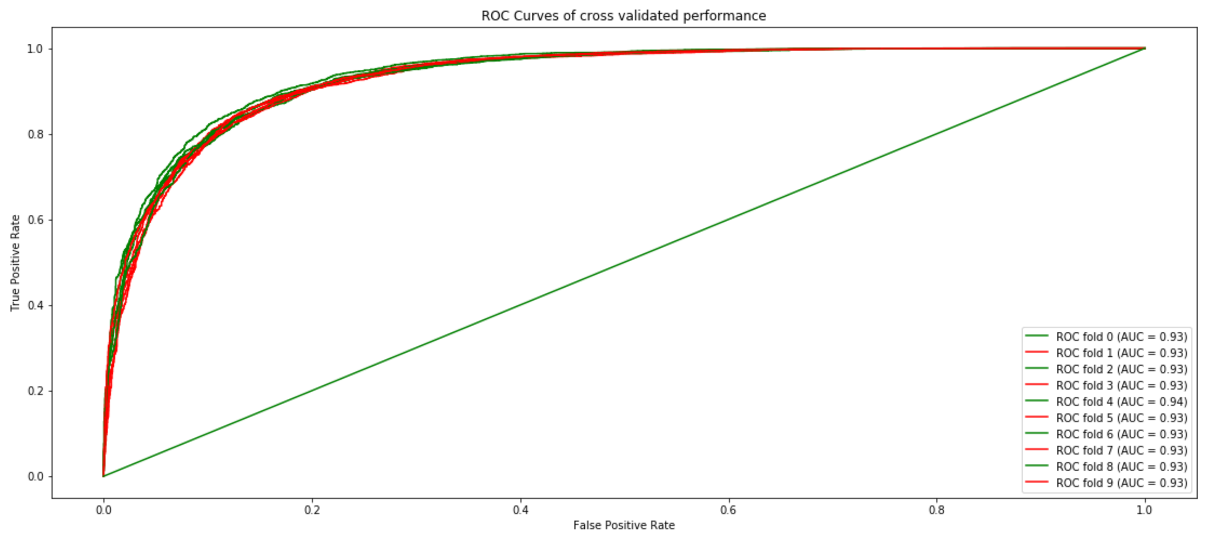




For small learning rates $\eta = 0.001$, cost function will decrease slowly. In contrast, for the large learning rate $\eta = 10$, cost function cannot reach the minimum.

Test your model performance through cross validation
(l)
Test the performance of your trained classifier using K-folds cross validation (while this can be done manually, the scikit-learn package StratifiedKFolds may be helpful). Produce Receiver Operating Characteristic Curves (ROC curves) of your cross validated performance.

In [410]:
```
from scipy import interp
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import LogisticRegression
X = train.iloc[:,0:2].values
y = train.y.values
k = StratifiedKFold(10)
tprs = []
#True positive rates
aucs = []
#Auc curves
i = 0
plt.figure(figsize=(19, 8))
for train_index, test_index in k.split(X, y):
    train_k = train.iloc[train_index,]
    test_k = train.iloc[test_index,]
    allweights,wold = gradient(train_k,0.25)
    predictions = prob(wold, test_k)
    fpr, tpr, thresh = roc_curve(test_k.y.values, predictions.values)
    tprs.append(interp(mean_fpr, fpr, tpr))
    tprs[-1][0] = 0.0
    roc_auc = auc(fpr, tpr)
    aucs.append(roc_auc)
    plt.plot(fpr, tpr,label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc))
    i += 1
plt.plot([0, 1], [0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves of cross validated performance')
plt.legend(loc="lower right")
plt.show()
```

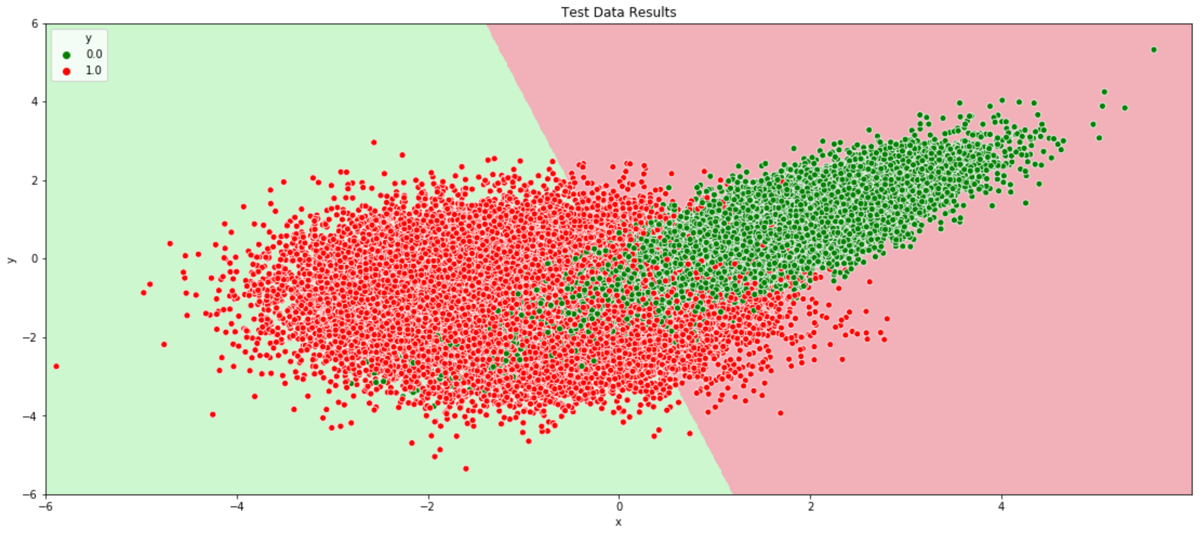


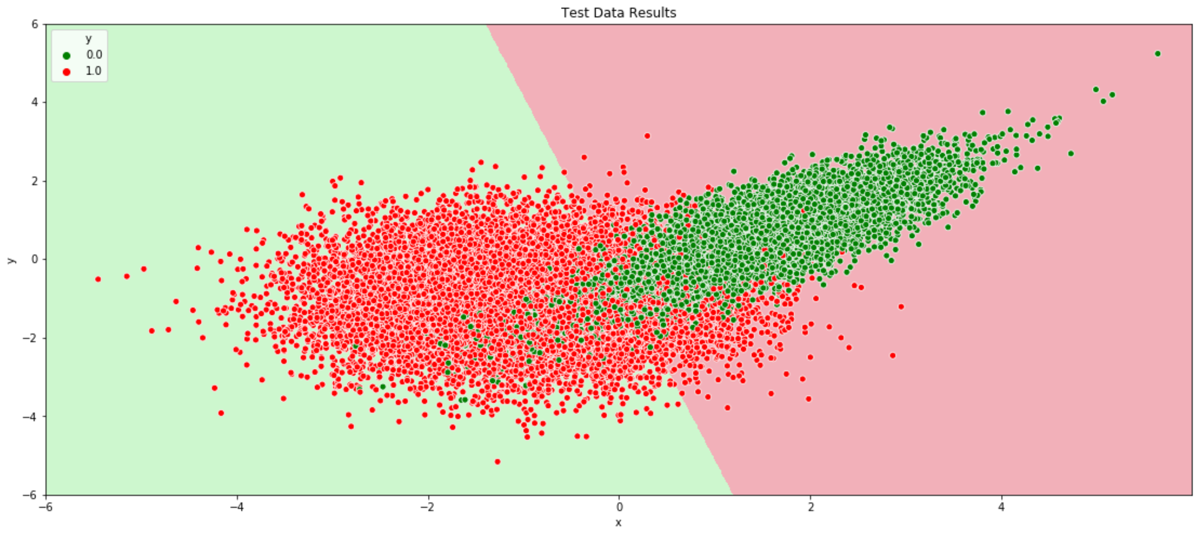
(m) Why do we use cross validation?
We use cross validation because it is a better approach to estimate test error than simply splitting data sets into train and test. With cross vaidation we are able to make predictions on all of our data and, therefore, estimate test error more accurately.

(n) Make two plots - one of your training data, and one for your test data - with the data scatterplotted and the decision boundary for your classifier. Comment on your decision boundary. Could it be improved?

(o) Compare your trained model to random guessing. Show the ROC curve for your model and plot the chance diagonal. What area under the curve (AUC) does your model achieve? How does your model compare in terms of performance?

In [398]:
```
meshsize = 0.01
xx, yy = np.meshgrid(np.arange(x_min, x_max, meshsize), np.arange(y_min,
y_max, meshsize))
data_mesh = np.array(prob(wold, pd.DataFrame({'x1':xx.ravel(),'x2':yy.ra
vel()})))
cmap_bold = ListedColormap(["#f2b0b9","#cdf7ce"])
data_mesh[data_mesh < .5] = 0
data_mesh[data_mesh >= .5] = 1
data_mesh = data_mesh.reshape(xx.shape)

plt.figure(figsize=(19, 8))
plt.pcolormesh(xx, yy, data_mesh,cmap=cmap_bold)
sns.scatterplot(x="x1", y="x2",hue="y", data=train)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Test Data Results")
plt.show()

plt.figure(figsize=(19, 8))
plt.pcolormesh(xx, yy, data_mesh,cmap=cmap_bold)
sns.scatterplot(x="x1", y="x2",hue="y", data=test)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Test Data Results")
plt.show()
```





In this case decision boundries are linear. It can be improved by choosing more flexible model.

```
In [411]: predictions = clf.predict(X_test)
          acc_test = accuracy_score(y_test, predictions)
```

o).

Logistic regression has a higher score for AUC (0.93 vs. 0.5 for random guessing), therefore logistic regression classifies data significantly better than random chance.

2

[20 points] Digits classification

(a) Construct your dataset from the MNIST dataset (http://yann.lecun.com/exdb/mnist/) of handwritten digits, which has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image.

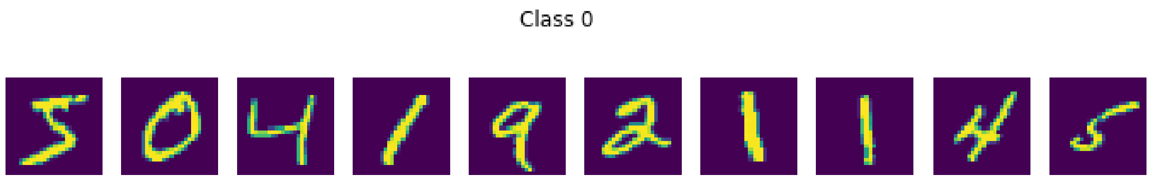
Your goal is to determine whether or not an example is a 3, therefore your binary classifier will seek to estimate $y = 1$ if the digit is a 3, and $y = 0$ otherwise. Create your dataset by transforming your labels into a binary format.

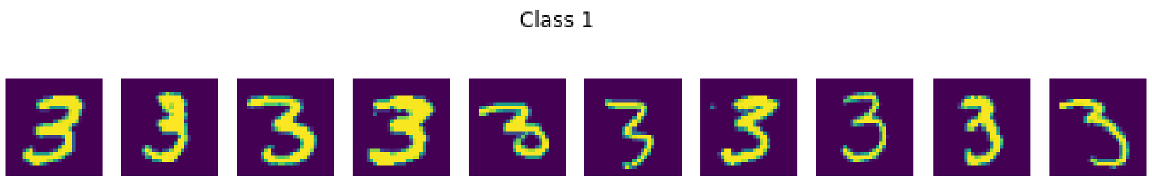
In [456]:
```
import keras
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [457]:
```
#Turn data to binary (0 if label = any number and 1 if it is equal to 3)
y_train = np.where(y_train != 3, 0, 1)
y_test = np.where(y_test != 3, 0, 1)

class0 = np.where(y_train==0)
random_nums = class0[0][:10]
class1 = np.where(y_train==1)
threes = class1[0][:10]
```

(b) Plot 10 examples of each class 0 and 1, from the training dataset.

In [466]:
```
plt.figure(figsize=(12,2))
#Plotting examples of class 0
plt.suptitle('Class 0')
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(x_train[random_nums[i]])
    plt.axis('off')
plt.figure(figsize=(12,2))
#Plotting examples of class 1
plt.suptitle('Class 1')
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(x_train[threes[i]])
    plt.axis('off')
```





(c) How many examples are present in each class? Are the classes balanced? What issues might this cause?

In [473]:
```
class0_count = len(np.where(y_train==0)[0])
class1_count = len(np.where(y_train==1)[0])
print('Number of examples in Class 0',class0_count, "vs ", class1_count,
"in Class 1")
```

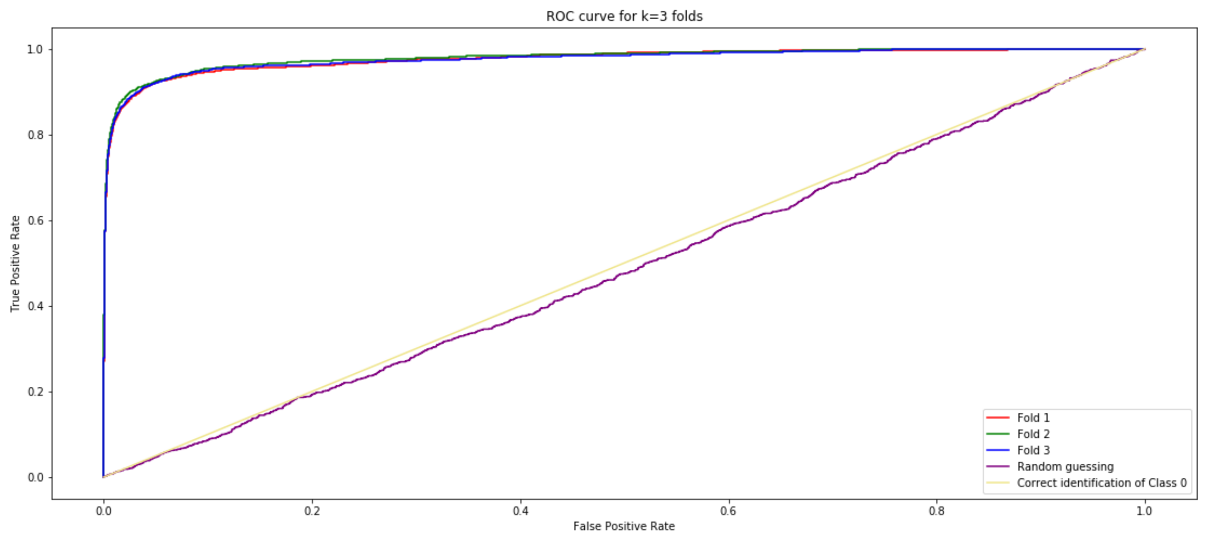
```
Number of examples in Class 0 53869 vs  6131 in Class 1
```

Because classes are highly unbalanced, it may cause an issue for a classifier to correctly predict class 1, whereas accuracy of predictions for class 0 will be high.

(d) Using cross-validation, train and test a classifier. Compare your performance against (1) a classifier that randomly guesses the class, and (2) a classifier that guesses that all examples are NOT 3's. Plot corresponding ROC curves and precision-recall curves. Describe the algorithm's performance and explain any discrepancies you find.

In [474]:
```
# reshape train and test sets
x_train = np.array([x_train[i].flatten() for i in range(0, x_train.shape
[0])])
x_test = np.array([x_test[i].flatten() for i in range(0,x_test.shape[0
])])
x_=[]; y_=[]; y_hat_=[]
k = StratifiedKFold(3)
for train_index, test_index in k.split(x_train, y_train):
  x_train1, x_test1 = x_train[train_index], x_train[test_index]
  y_train1, y_test1 = y_train[train_index], y_train[test_index]
  logreg = LogisticRegression(solver='liblinear')
  logreg = logreg.fit(x_train1, y_train1.ravel())
  pred = logreg.predict_proba(x_test1)
  x_.append(x_train1)
  y_.append(y_test1)
  y_hat_.append(pred)
```

In [483]:
```
# Random guessing
y_rand = np.random.rand(y_test.shape[0], 2)
y_hat_.append(y_rand)
y_.append(y_test)
#Identifies Class 0 correctly
y0 = np.zeros_like(y_rand)
y_hat_.append(y0)
y_.append(y_test)
colors = ['r', 'g', 'b', 'purple', 'khaki']
labels = ["Fold 1","Fold 2","Fold 3","Random guessing", "Correct identif
ication of Class 0"]
plt.figure(figsize=(19, 8))
for i in range(5):
    fpr1, tpr1, thresholds1  = (roc_curve(y_[i],y_hat_[i][:,1], pos_labe
l=1))
    plt.plot(fpr1,tpr1,c=colors[i],label=labels[i])
plt.legend()
plt.title('ROC curve for k=3 folds')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



(f) Using a logistic regression classifier (a linear classifier), apply lasso regularization and retrain the model and evaluate its performance over a range of values on the regularization coefficient. You can implement this using the LogisticRegression (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) module (DO NOT use your function from question 1) and activating the 'l1' penalty; the parameter C is the inverse of the regularization strength. As you vary the regularization coefficient, plot (1) the number of model parameters that are estimated to be nonzero; (2) the logistic regression cost function, which you created a function to evaluate in the Question 1; (3) F_1-score, and (4) area under the curve (AUC). Describe the implications of your findings.

As the regularization coefficient increases, the cost increases as well, whereas AUC and F1 score decrease.

3

[40 points] Supervised learning exploration

For this exercise, you will construct and implement a supervised learning problem solution/experiment. Describe your process and answer these questions clearly and thoroughly. Part of the grade in this assignment is devoted to the quality and professionalism of your work.

(a) Identify a question or problem that's of interest to you and that could be addressed using classification or regression. Explain why it's interesting and what you'd like to accomplish. This should exhibit creativity, and you are not allowed to use the Iris dataset, the Kaggle Titanic dataset, or the Kaggle chocolate dataset.

The question of research: the characteristics of breast cancer with a goal to predict whether the cancer is benign or malignant. This dataset was downloaded from Kaggle (available at https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/version/2#data.csv (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/version/2#data.csv)). The variable of interest is "diagnosis" which is a binary variable, taking values "M" or "B" (malignant and benign respectively). Other 31 variables represent different characteristics of a tumor, such as a radius, symmetry, compactness etc., all of which are numeric continuous variables. The detailed description of all variables can be found in a description of the data set at Kaggle. According to the breastcancer.org, this is the most common type of cancer for women (25.4 % of all cancers). In 2018 was diagnosed over 2 million new cases worldwide, according to https://www.wcrf.org/dietandcancer/cancer-trends/worldwide-cancer-data (https://www.wcrf.org/dietandcancer/cancer-trends/worldwide-cancer-data). This question is important for me because I would like to apply my machine learning skills to bring the real value to the world, and contributing to the solution of the cancer problem is extremely fulfilling.

(b) Download the data and plot the data to describe it.

```
In [164]: cancer = pd.read_csv('cancer.csv')
```

Many variables are highly correlated (correlation coefficient R >0.7, because some of the variables are linear transformations of other variables, such as radius, area, perimeter. We will use only one variable from a pair of correlated ones in the analysis to avoid the collinearity issue and decrease the dimensionality for a faster computing. Also, we will not include a variable "id" because it does not bring value to the model.

In [165]:
```
cancer=cancer.drop(columns=['id', 'perimeter_mean','area_mean','concave
 points_mean','radius_worst','perimeter_worst','area_worst','concave poi
nts_worst','texture_worst',
                            'radius_se','area_se','perimeter_se','smoothn
ess_worst','concavity_mean','compactness_se','compactness_worst','Unname
d: 32','concavity_worst','fractal_dimension_worst','fractal_dimension_s
e','symmetry_worst','concave points_se'])
```

In [166]:
```
corr = cancer.corr()
corr.style.background_gradient()
```

Out[166]:

	radius_mean	texture_mean	smoothness_mean	compactness_
radius_mean	1	0.323782	0.170581	0.506124
texture_mean	0.323782	1	-0.0233885	0.236702
smoothness_mean	0.170581	-0.0233885	1	0.659123
compactness_mean	0.506124	0.236702	0.659123	1
symmetry_mean	0.147741	0.071401	0.557775	0.602641
fractal_dimension_mean	-0.311631	-0.0764372	0.584792	0.565369
texture_se	-0.0973174	0.386358	0.0684064	0.0462048
smoothness_se	-0.2226	0.00661378	0.332375	0.135299
concavity_se	0.194204	0.143293	0.248396	0.570517
symmetry_se	-0.104321	0.00912717	0.200774	0.229977

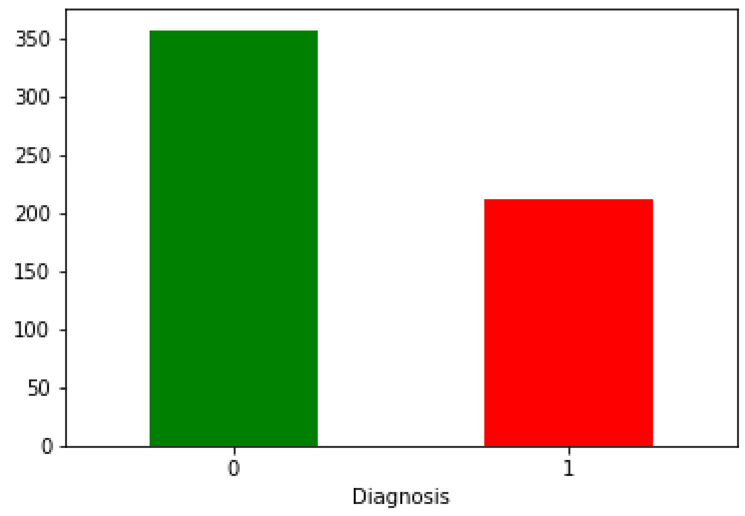
In [167]:
```
print (cancer.shape)
```

```
(569, 11)
```

After removing all correlated variables, the data set contains 10 predictor variables and one binary variable "diagnosis", which is our outcome variable.

In [168]:
```
cancer.loc[cancer['diagnosis'] == 'M', ['diagnosis']] = 1
cancer.loc[cancer['diagnosis'] == 'B', ['diagnosis']] = 0
cases = pd.DataFrame({'Diagnosis':['0', '1'], 'Cases':[(len(cancer)-sum(
cancer['diagnosis'])),sum(cancer['diagnosis'])]})
```

```
In [169]: freq = cases.plot.bar(x='Diagnosis', y='Cases', rot=0, legend=False, col
          or=['green','red'])
```

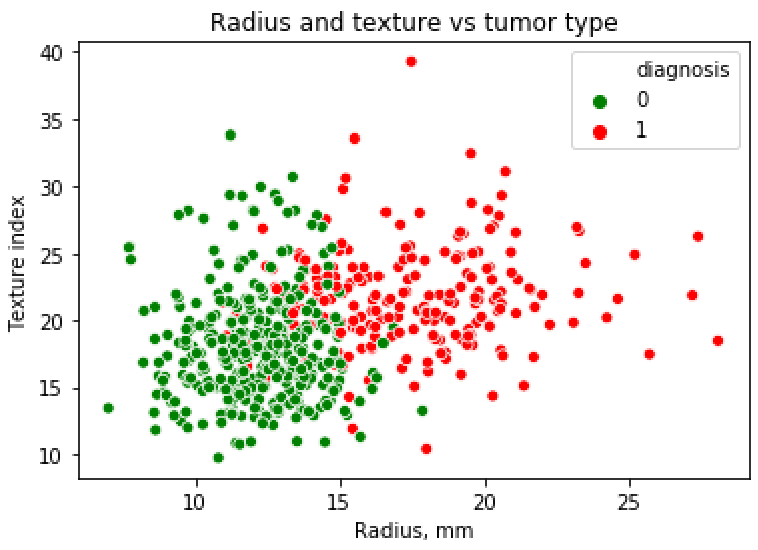


Note: the classes are unbalanced, so it potentially may cause an issue for the model which would predict benign tumor better than a malignant tumor. However, the data contains 212 cases with a malignant tumor, so the sample is still sufficient to attempt the modeling.

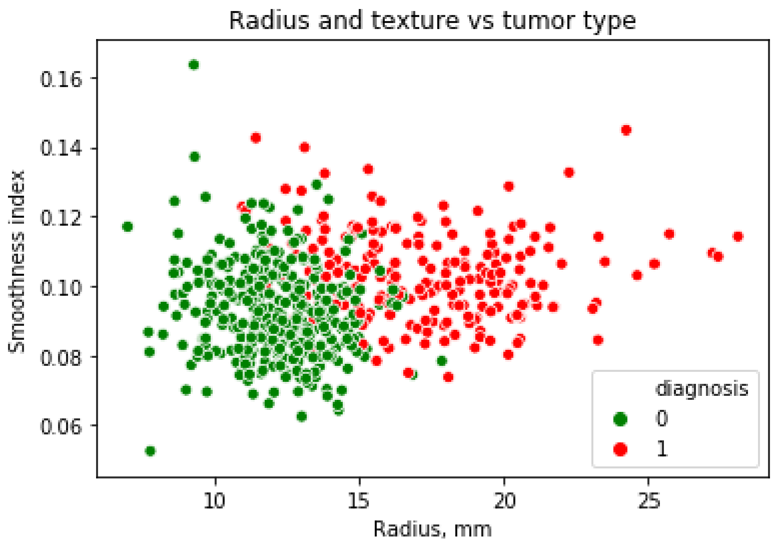
```
In [170]: cancer.isnull().sum()

Out[170]: diagnosis                 0
          radius_mean               0
          texture_mean              0
          smoothness_mean           0
          compactness_mean          0
          symmetry_mean             0
          fractal_dimension_mean    0
          texture_se                0
          smoothness_se             0
          concavity_se              0
          symmetry_se               0
          dtype: int64
```

Dataset has no missing values and is ready for the analysis. The scatter plots below show that data is well separable. Now let's plot the data to see whether it could be separated by class by using a combination of different parameters.

```
In [171]: col_list = ["green","red"]
          sns.set_palette(col_list)
          tumor = sns.scatterplot(x=cancer.radius_mean, y=cancer.texture_mean, hue
          =cancer.diagnosis).set_title("Radius and texture vs tumor type")
          plt.xlabel('Radius, mm')
          plt.ylabel('Texture index')

Out[171]: Text(0,0.5,'Texture index')
```

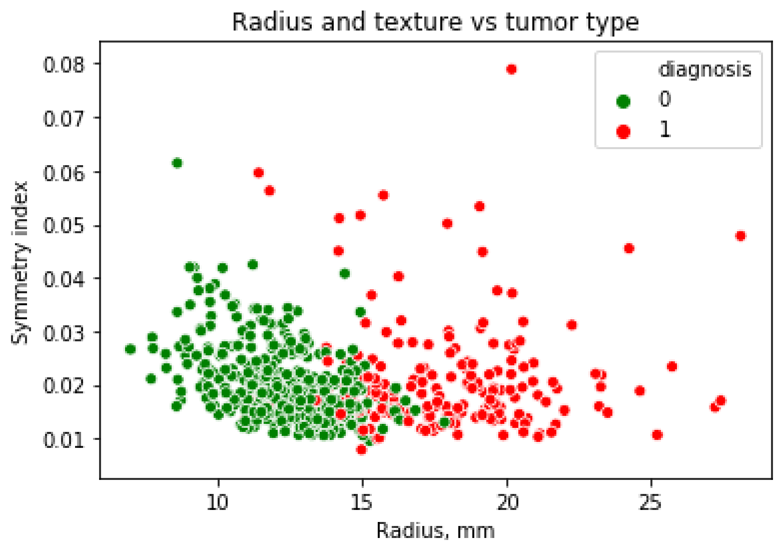


```
In [172]: tumor = sns.scatterplot(x=cancer.radius_mean, y=cancer.smoothness_mean,
          hue=cancer.diagnosis).set_title("Radius and texture vs tumor type")
          plt.xlabel('Radius, mm')
          plt.ylabel('Smoothness index')

Out[172]: Text(0,0.5,'Smoothness index')
```



In [173]:
```
tumor = sns.scatterplot(x=cancer.radius_mean, y=cancer.symmetry_se, hue=
cancer.diagnosis).set_title("Radius and texture vs tumor type")
plt.xlabel('Radius, mm')
plt.ylabel('Symmetry index')
```

Out[173]:
```
Text(0,0.5,'Symmetry index')
```



(c) Formulate your supervised learning question: (a) What is your target variable (what are you trying to predict) and what predictors do you have available? Does your dataset require any preprocessing: is it clean (no missing values or erroneous data) and normalized (are each of the predictors of the same magnitude)?

We are trying to predict whether the tumor is malignant or benign based on the characteristics of the tumor such as radius and smoothness. The detailed desription of all varialbes is available at https://www.kaggle.com/uciml/breast-cancer-wisconsin-data (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data). We have already excluded the correlated variables decresing the number of dimensions to $n =$ 11 vs 32 which were presented in the original set. Data does not have missing values and does not require further cleaning. All of the predictors are of the same magnitude.

(d) What supervised learning technique will you use and why?

Logistic regression, because we have a binary outcome variable.

(e) How will you evaluate performance and know whether you succeeded (e.g. ROC curves for binary classification, mean square error or R^2 for regression)?

We will use ROC curve because the outcome variable is binary (class 1 for a malignant tumor and class 0 for a benign tumor).

(f) Divide your dataset into training and testing datasets OR implement cross validation. Explain your approach and why you adopted it.

Because the data set is relatively small and classes are unbalanced, cross validation is a better choice.

(g) Run your analysis and show your performance. Include plots of your data and of performance.

In [259]:
```
X = cancer.iloc[:,1:11]
y = cancer.iloc[:,0:1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [260]:
```
clf = LogisticRegressionCV()
clf = LogisticRegressionCV(cv=10, multi_class='ovr').fit(X_train, y_trai
n)
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py:578:
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().
```

In [261]:
```
acc_train=clf.score(X_train, y_train)
print ("Accuracy for the train data:", round(acc_train,5))
```

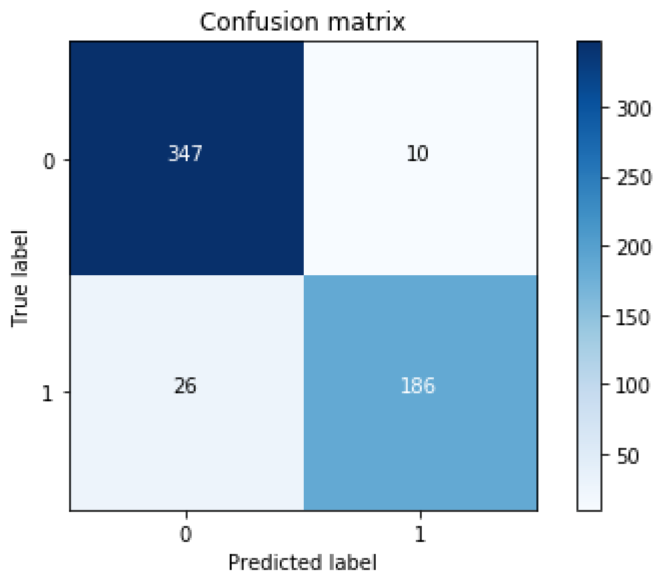
```
Accuracy for the train data: 0.94286
```

In [262]:
```
predictions = clf.predict(X_test)
acc_test = accuracy_score(y_test, predictions)
print ("Accuracy for the train data:",round(acc_test,5))
```

```
Accuracy for the train data: 0.90351
```

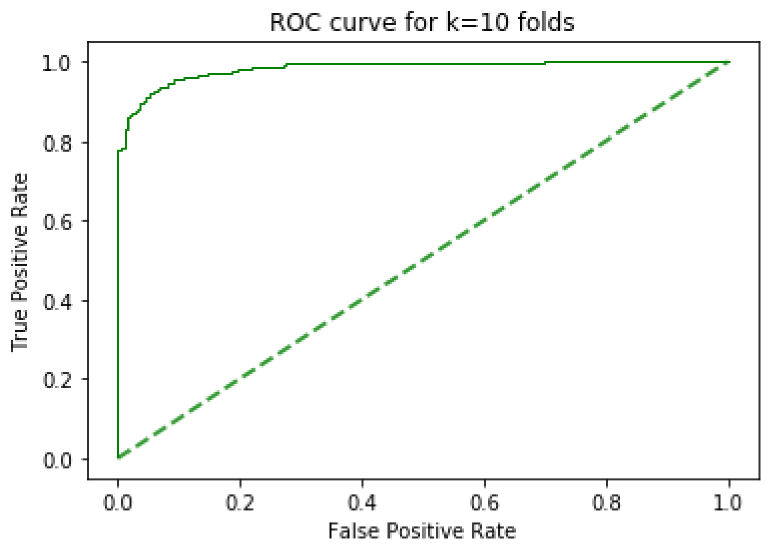
In [234]:
```
#Reference: https://www.kaggle.com/wilsonf/logistic-regression-cv-test
import itertools
cnf_matrix = metrics.confusion_matrix(y, ypredict)
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1
])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['0','1'],
                      title='Confusion matrix')
```

```
Confusion matrix, without normalization
```



In [271]:
```
plt.plot(fpr, tpr, lw=1, label='ROC fold %d (area = %0.2f)' % (0,auc(fpr
, tpr) ))
plt.title('ROC curve for k=10 folds')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='g',label='Luck', a
lpha=.8)
```

Out[271]:
```
[<matplotlib.lines.Line2D at 0x1a5eb69a58>]
```



In addition to machine learning algorithm, we want to interpret the model correctly.

In [282]:
```
print ("coefficients:", clf.coef_)
```

```
coefficients: [[  1.4394038    0.52710058 115.68395853   3.4516301   2
3.81790957
   -2.71633835  -1.77685029   0.90680059  19.16543053 -15.40556782]]
```

Therefore, if tumor is increasing by 1mm in radius, the odds it is malignant is 1.43 times higher.

(h) Describe how your system performed, where your supervised learning algorithm performed well, and where it did not, and how you could improve it.

The logistic regression algorithm showed the accuracy 0.9 for the test data, which is a high result. More advanced models such as random forest of CNN could show higher accuracy. However, those models are harder to interpret, which is the key purpose of the research question. Given the high accuracy and interpretability, logistic regression is a good choice for this data.

(i) Write a brief summary / elevator pitch for this work that you would put on LinkedIn to describe this project to future employers. This should focus on the high level impact and importance and overall takeaways and not on the nitty-gritty details.

Using machine learning algorithms humanity may solve many crucially important issues including cancer determination and classification on its early stages which can save millions of lives. While more advanced models could give better accuracy score, the logistic regression is easy to interpret. The suggested model shows given an increase in radius by 1 mm increases the odds for tumor to be malignant 1.43 times.