In [1]:
```
"""
ECGR 5105 - Intro to Machine Learning
Homework 2 - Part 4
Phillip Harmon
"""
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
#Load and Build the Dataset
from sklearn.datasets import load_breast_cancer
loaded  = load_breast_cancer()
labels  = np.reshape(loaded.target, (len(loaded.target),1))
inputs  = pd.DataFrame(loaded.data)
names   = np.append(loaded.feature_names, 'label')
dataset = pd.DataFrame(np.concatenate([inputs,labels],axis=1))
dataset.columns = names
dataset
```

Out[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mea symmetr
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.241
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.181
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.206
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.259
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.180
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.172
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.175
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.159
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.239
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.158

569 rows × 31 columns

In [3]:
```
#Sort Dataset
x = dataset.iloc[:,0:-1].values
y = dataset.iloc[:,-1].values
```

In [4]:
```
#Clean the Dataset
from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()
 # scaler = MinMaxScaler() #StandardScaler gave better results here
x = scaler.fit_transform(x)
```

In [5]:
```
#Perform the Training with K=5
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
kcup = KFold(n_splits=5, random_state=1337, shuffle=True)
model = LogisticRegression(random_state=1337)
results = cross_val_score(model,x,y,cv=kcup)
print("K=5 | Accuracy: {:.3f}% ({:.3f}%)".format(results.mean()*100, results.s
```

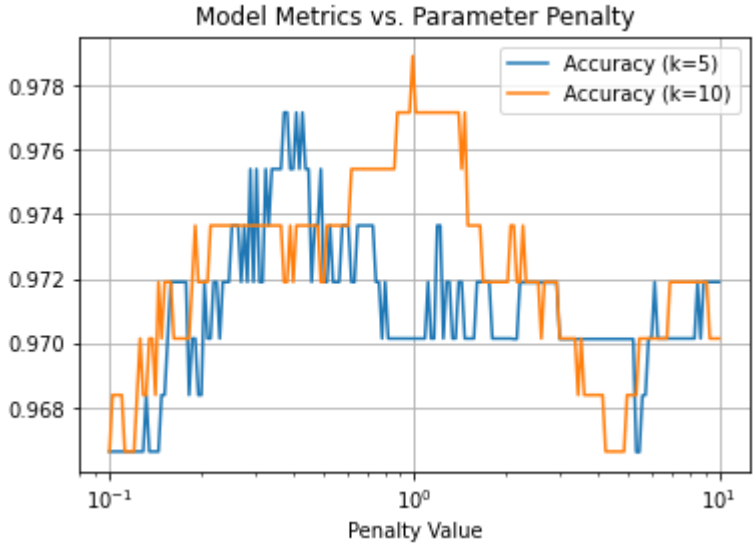
```
K=5 | Accuracy: 97.539% (1.290%)
```

In [6]:
```
#Perform the Training with K=10
kcup = KFold(n_splits=10, random_state=1337, shuffle=True)
model = LogisticRegression(random_state=1337)
results = cross_val_score(model,x,y,cv=kcup)
print("K=10 | Accuracy: {:.3f}% ({:.3f}%)".format(results.mean()*100, results.
```

```
K=10 | Accuracy: 97.716% (1.122%)
```

In [7]:
```
#Reevaluate using a variety of weight penalties
lambdas = np.logspace(-1,1,num=200)
k5_acc_log = []
kcup = KFold(n_splits=5, random_state=1337, shuffle=True)
for lam in lambdas:
    model = LogisticRegression(penalty='l1',C=lam,solver='liblinear',random_st
    results = cross_val_score(model,x,y,cv=kcup)
    k5_acc_log.append(results.mean())
k10_acc_log = []
kcup = KFold(n_splits=10, random_state=1337, shuffle=True)
for lam in lambdas:
    model = LogisticRegression(penalty='l1',C=lam,solver='liblinear',random_st
    results = cross_val_score(model,x,y,cv=kcup)
    k10_acc_log.append(results.mean())
```

In [8]:
```
#Plot the results
plt.semilogx(lambdas,k5_acc_log,label='Accuracy (k=5)')
plt.semilogx(lambdas,k10_acc_log,label='Accuracy (k=10)')
plt.grid()
plt.xlabel('Penalty Value')
plt.title('Model Metrics vs. Parameter Penalty')
plt.legend();
```



In [9]:
```
#Best k=5 weight is about 0.35
kcup = KFold(n_splits=5, random_state=1337, shuffle=True)
model = LogisticRegression(penalty='l1',C=0.35,solver='liblinear',random_state
results = cross_val_score(model,x,y,cv=kcup)
print("K=5 | Accuracy: {:.3f}% ({:.3f}%)".format(results.mean()*100, results.s
```

```
K=5 | Accuracy: 97.541% (0.653%)
```

In [10]:
```
#Best k=10 weight is about 1
kcup = KFold(n_splits=10, random_state=1337, shuffle=True)
model = LogisticRegression(penalty='l1',C=1,solver='liblinear',random_state=13
results = cross_val_score(model,x,y,cv=kcup)
print("K=10 | Accuracy: {:.3f}% ({:.3f}%)".format(results.mean()*100, results.
```

```
K=10 | Accuracy: 97.892% (1.312%)
```

In [ ]: