FEATURE SELECTION AND PREDICTION WITH MARKOV-CHAIN RANDOMIZATION IN BREAST CANCER DATA

OLIVER BRADLEY, JULIANNE HIGGINS, SHUYI TAN

Abstract. This report is an analysis of the Breast Cancer Wisconsin (Diagnostic) data set, and is an exploration of various methods of feature selection. The aim was to find an efficient and effective feature selection method. Using logistic regression as our model, we compared methods of selecting features by using a computationally extensive iterative loop, and Monte Carlo Simulations. Comparing the accuracy and computational time from the fitting the models, we found that Monte Carlo Simulations are an effective method to quickly select models that perform accurately, and include less features. For example, by using Monte Carlo Simulations, we found a model that performed with a 93% accuracy, and only used 3 features. This can be compared to the model found with the iterative loop that resulted in a 94% accuracy and used 15 features. We decided that a highly performing model, with less features is a more important result, and a better model for public health professionals to use.

1. Introduction. Breast cancer is "the most common cancer diagnosed among US women (excluding skin cancers), and is the second leading cause of cancer deaths among women" [1]. Breast cancer affects millions of women, and men, and accounts for over 25% of all cancers [3]. As with other cancers, early detection is very important in breast cancer because early treatment can greatly increase one's chance at survival. The most important screening test for breast cancer is the mammogram, which is an X-ray of the breast. If a patient is showing possible signs of breast cancer, doctors then recommend a fine needle aspirate biopsy. The Breast Cancer dataset was collected by observing phenotypic features from these biopsies. However, there is a tremendous amount of data about the tumors that can be collected. The goal of our project is to filter through these variables to find the most predictive combination of features. Feature selection "aims at building better classifier[s] by listing significant features which also helps in reducing computational overload" [4]. We will start with an exhaustive feature selection method, by testing every possible combination of predictors. Then, we will explore another way to select features by using Monte Carlo Simulations. We will iterate over all combinations of predictors including interaction terms to find a

method with great balance between accuracy and computational cost. We will also look into stable predictors that maintain consistent significance with other predictors being added or dropped from the model. We explored and cleaned the dataset in 2.1-2.2, and ran simple logistic regression on the cleaned data frame and record the run-time in 2.3. Additionally, we ran logistic regression on all combinations of predictor with/without pairwise predictors in 2.4 and 2.5. Owing to the computational limit and overfitting, we conduct randomization on possible combinations for predators in 2.7.

2. Methods.

2.1. Basic Overview of Data. In this project we used the Breast Cancer Wisconsin (Diagnostic) data set from the UCI Machine Learning Repository[6]. This data was collected by Dr. William H. Wolberg from the Computer Science Department at the University of Wisconsin. The data set contains 569 observations and 32 variables. The response variable is binary and has two levels: M = Malignant and B = Benign. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. Ten types of real-valued features are computed for each cell nucleus: radius (distance from center to points on the perimeter), texture (standard deviation of gray-scale values), perimeter, area, smoothness (local variation in local lengths), compactness ($perimeter^2$ /area - 1.0), concavity (severity of concave portions of the contour), concave points (number of concave portions of the contour), symmetry, and fractal dimension ("coastline approximation" - 1). The mean, standard error, and the "worst" value, or the largest mean value, are calculated for each image.

2.2. Data Exploration.

2.2.1. Exploratory Analysis. First, we dropped the variable for ID numbers from our data set and visualize the diagnosis counts. There are three groups of predictors: mean, standard error, and the worst mean value. In this project, we will only be using the 10 variables from the mean group.

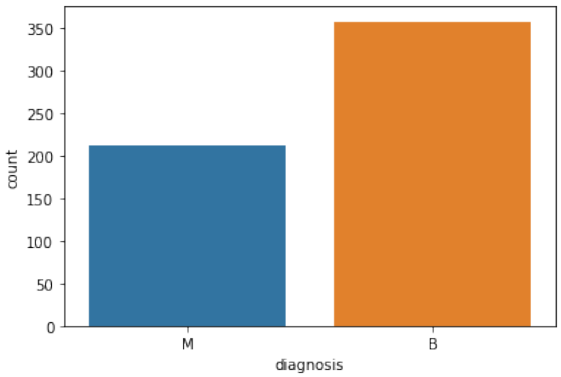

FIG. 1. *Counts of malignant and benign.*

It is demonstrated in the bar chart that there are 357 benign diagnoses and 212 malignant diagnoses in out data set.

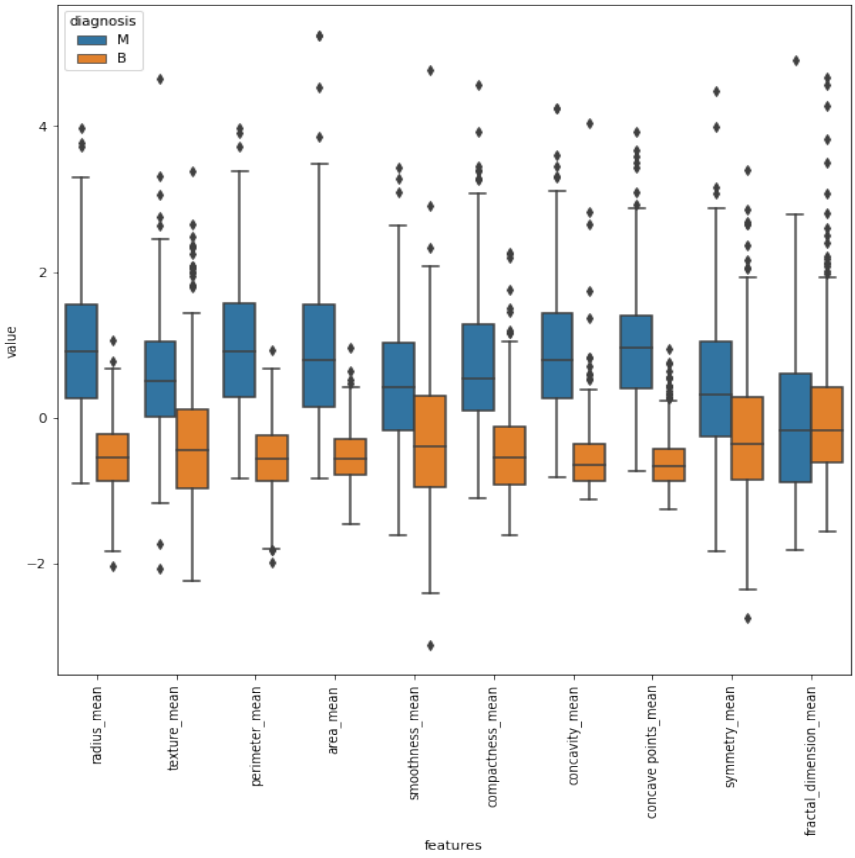

FIG. 2. *Boxplots of variables separated by diagnosis*

This plot provides preliminary information on potential significance of each variable. For example, in the concavity-mean feature, the median of the Malignant and Benign plots are very separated so we infer that it could be useful in our model. However, in fractal dimension feature, the medians of Malignant and Benign are very close, which indicates that they may not be as useful or predictive. We used this boxplot as an explorative method of viewing the data

2.2.2. Heat Map. To examine the correlation among predictors, we construct a heatmap. The presence of multicollinearity affects the precise effects of predictors and makes the predictions very sensitive to minor changes in the model[2].

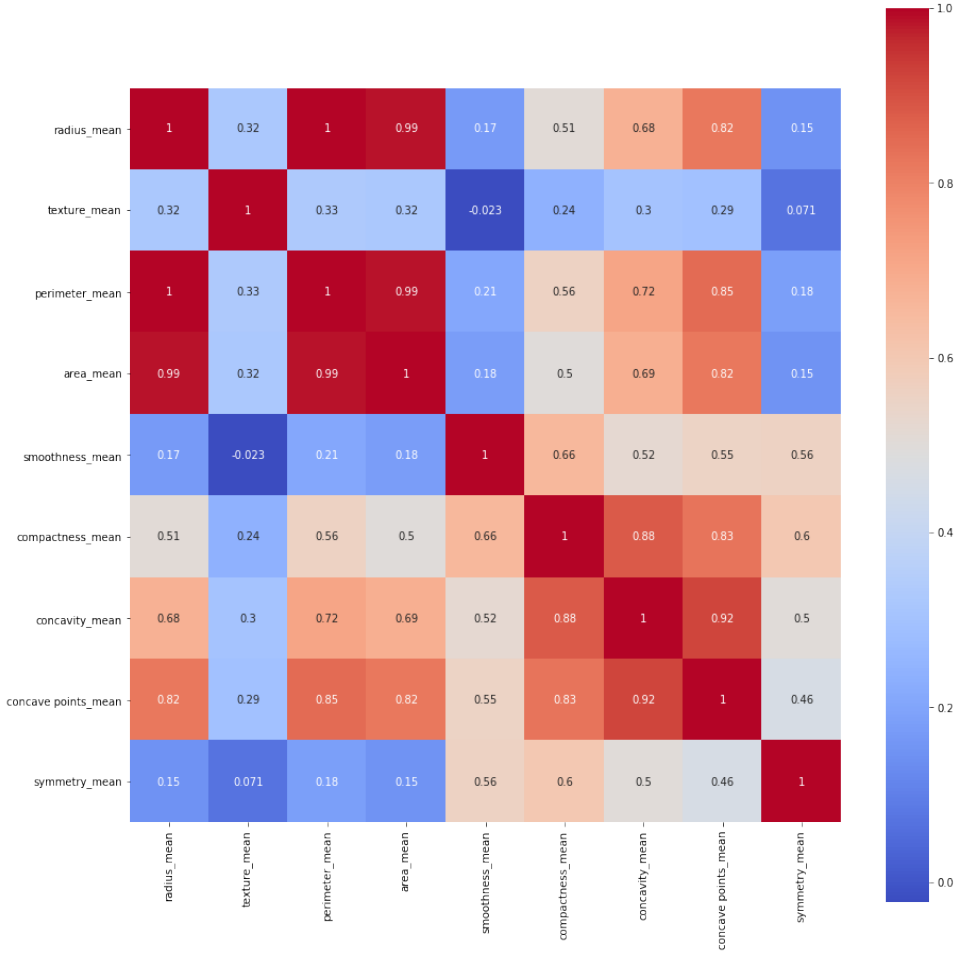

FIG. 3. *Heatmap of Predictors*

It is shown in the heatmap that there is multicollinearity between the 10 variables. For example, there is a strong correlation between area mean, radius mean ,and perimeter mean; compactness mean, concavity mean and concavity points mean are also highly correlated. In order to reduce this multicollinearity, we drop area mean, perimeter mean, compactness mean, and the concavity mean.

2.2.3. Creating a new data frame. Based on the interpretation of the heat map, we are only keeping five out of ten predictors: radius, texture, smoothness, concavity, and symmetry. A new data frame "data_clean" is created with these five predictors.

2.3. Run simple logistic regression on all 5 selected predictor variables. Now that we have explored and wrangled the data sufficiently, we fit this data to a model. We aim to predict future cases with high accuracy and hopefully confirm some reliable predictors.

2.3.1. Logistic regression. For our model of choice, we decided to use logistic regression. The main reason for this decision is to start our process with a simpler model before we could move on to more complex ones. Generally, logistic regression is suited for examining hypotheses about the relationship between one categorical dependent variable and continuous predictors [7]. Therefore, we use logistic regression because our variable of interest is malignancy, which is a binary variable. Linear regression is not appropriate here. Our model will not only try to predict whether a tumor is benign or malignant as it will also provide the probability of that outcome.

2.3.2. Record run-time. Since we planned on mapping and testing all possible iterations of a logistic regression, we must also first see if it is computationally possible to try all these models. We will estimate this run time by recording the amount of time it takes to fit a single logistic model and to cross validate it with 10 folds. We will then multiply that time by the amount of feature combinations possible. First, we mapped a logistic regression with our five predictors and cross validated it with 10 folds. It achieved a cross validation score of 0.89657 and took .143929 seconds to run.

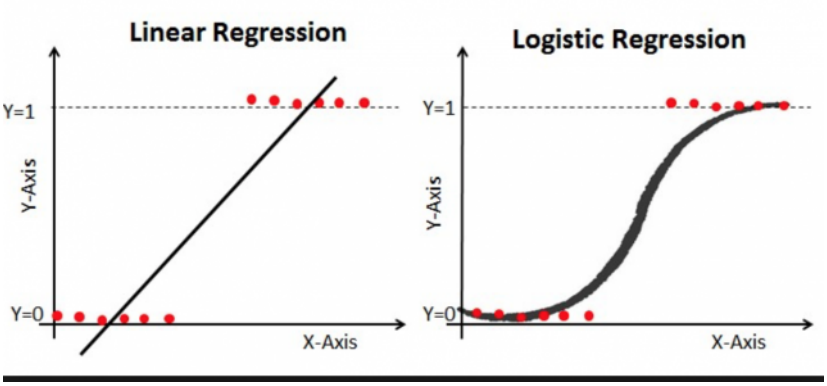

FIG. 4. *Comparison between Logistic and Linear Regression plots* [5]

2.4. Loop through all combinations of the 5 variables. The amount of possible combinations is a simple formula. This amount follows the summation:

$$\sum_{i=1}^{5} \binom{5}{i} = 31$$

Our end result is that there are 31 possible combinations of features from the 5 predictors. Therefore, we can make 31 logistic regression models. Our total estimated run time will be the average time of fitting one model and cross validating it (.13 seconds) times 31. The estimated runtime should only take approximately 4.4618 seconds. Even if our estimation is off, this should take no time at all to run. This run time calculation will be more important later as the number of predictors increases. As we increase dimensionality and number of predictors, we expect to hit a computational limit, where it may take days or longer to run through all possible models.

2.4.1. Create a list of all possible combinations of the 5 variables. Before we can actually map all these models, we need to make a list of all the possible combinations of predictors. Since we can specify which variables we want to use by giving a data frame and specifying the variables with a list, we generated a nested list. Each element is a list containing the names of the predictors to be used for that iteration. For example, elements in this list included [‘radius_mean’], and [’concavity_mean’, ’symmetry_mean’].

2.4.2. Fit logistic regression to all 31 feature combinations. After creating the nested list, we looped through all the possible logistic regression models to find the one with the highest cross validated accuracy. At the end of the loop, the model with the highest accuracy was one that used all five predictors. We decided that this was a fairly high accuracy, and since we preemptively selected our features, using all 5 predictors should not pose a serious risk of overfitting.

2.5. Include interaction terms. While the accuracy score is relatively high using the 5 predictor variables, we wanted to investigate if we could increase the score by including interaction variables in the model. Adding interaction terms to a regression model can expand the understanding of the relationships among the variables in the model. The presence of a significant interaction indicates that the effect of one predictor variable on the response variable is different at different values of the other predictor variable. We decided to stick to interaction variables including only 2 variables, as the interpretation of 3 or more variables in interaction becomes very difficult to interpret. So, from our 5 variables we added all combination of pairwise interaction terms into the data frame. There are 10 possible interaction terms, as 5Choose2 = 10. We wrote a loop that combined our original dataframe, called data_clean, with these 10 new interaction variable columns. This panda dataframe is called data_interactions.

2.5.1. Create a list of Combinations. Now, using the new dataframe, data_interactions, we repeated step 2.4.1 to create a list of all possible combinations of the 15 variables. This result is a nested list, containing 32,767 possible combinations.

$$\sum_{i=1}^{15} \binom{15}{i} = 32,767$$

In the code, this list of all feature combinations is called "combination_list_int".

2.5.2. Loop through all 15 Variables. Clearly, this number is much larger than the previous 31 feature combinations that we had computed earlier. Therefore, we expected that the loop to calculate the accuracy score of each feature combination in the logistic regression model to take much longer. We found this to be true, as the

loop took around 2 hours to complete. We found a similar result to the original 31 feature loop as the loop found the model with the highest accuracy to be the model which included all 15 features. While this model may have the highest accuracy score, we realize that this does not mean that it is the "best" model. It seems like this method of choosing models will just select the model that includes all the features. However, this leads to overfitting and an uninterpretable result. Also it is very computationally extensive. We will now explore the power of Monte Carlo simulations in feature selection.

2.6. Randomization loop. We found that following the logic of Monte Carlo simulations to be an effective method of feature selection. In this loop, we first specified how many models we wanted to create. Then we chose a random number of variables to include in the model. The last step was to randomly select the chosen number of variables to be chosen for the model. This randomization not only helped with computational limits, but also resulted in an approximately good model with less chance of being overfitting. The idea of randomization is selecting the features randomly, measuring the model performances by k-fold cross-validation, and repeating it many times. The feature combination that gives the best performances and has the least number of features is the one we are looking for [8]. Refer to the appendix to see the code for this randomization process.

3. Summary of Results. At the end of our process, we have a new method using random search to find accurate models that are not overftting. In addition, looking through the resulting models can help indicate which predictors are most present in accurate models. Compared to our model in our complete search, the random search model had significantly less predictors and complexity with negligible accuracy loss.

Method	# of Vars	Accuracy	Runtime
Complete	15	94.05%	2 hrs
Random	3	92.98%	2 mins

4. Conclusion. In this report, we design and implement two methods of feature selection. Using logistic regression as our model, we compare a computationally exhaustive iterative loop to a random selection of feature combinations. We found random search to be effective feature selection method, while allowing us to control the computational load. In choosing the best combination of features, we considered both the accuracy of the model's performance, but also the number of features included. Ultimately, random search resulted in a more balanced model, as well reducing computational load. Now that we have a method to search for viable feature combinations, we consider future directions. Future research would include scanning for common predictors which appear in higher performing models. Also, this random search method can be applied to different models. Lastly, as the computational ceiling is lower in random search, we could include higher dimensionality and more variables in the model. Overall, feature selection is just as important, if not more important than model selection and random selection is a powerful tool in selecting features.

REFERENCES

[1] DeSantis, Carol *Breast Cancer Statistics.* , Et.al, CA: A Cancer Journal for Clinicians, 2019.

[2] Gharibdousti, Maryam *Breast Cancer Diagnosis using feature extraction techniques* , Et.al, Applied Medical Information, 2019.

[3] Ghoncheh, Mahshid *Incidence and Mortality and Epidemiology of Breast Cancer in the World* , Et.al, Asian Pacific Journal of Cancer Prevention, 2016.

[4] Khaire, Utkarsh *Stability of feature selection algorithm: A review* , Et.al, Science Direct, 2019.

[5] Maklin, Cory *Logistic Regression with Python* Medium Machine Learning, 2019.

[6] Wolberg, William.H *Breast Cancer Wisconsin (Diagnostic) Data Set* UCI Machine Learning Repository,1992

[7] Enders,CK *Applied Missing Data Analysis. Methodology in the Social Sciences Series* Guilford Press,2010

[8] Malato, Gianluca *Feature selection by random search in Python* Towards Data Science, 2019

5. Appendix.

5.1. Code Used.

```
# -*- coding: utf-8 -*-
"""Appendix.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1vZnkLdPrmhv_wZZJf_vrtAcUDPUlm-Il

Importing packages and Data
"""

# Commented out IPython magic to ensure Python compatibility.
#Import all necessary packages

import numpy as np

import pandas as pd

import sklearn.model_selection as model_selection
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

from sklearn.model_selection import cross_val_predict, cross_validate
from sklearn import linear_model

import seaborn as sns

from datetime import datetime

import matplotlib.pyplot as plt
# %matplotlib inline

import itertools
import sklearn.datasets

#Import data locally
#Call dataset, "data"
data = pd.read_csv('/content/Breast_Cancer_data.csv')

"""Cleaning and Exploring the Data"""

#Exploratory analysis
pred_drop = ['id','diagnosis']
pred_only = data.drop(pred_drop,axis = 1 )
data.head()
```

```
#Analyze the distribution of the dependent variable
sns.countplot(data['diagnosis'],label="Count")


#Separate data into a list of mean variables
predictors_mean = list(data.columns[2:11])


#Heat map
corr = data[predictors_mean].corr()
plt.figure(figsize=(15,15))
sns.heatmap(corr,cbar=True, square = True, annot=True, xticklabels = predictors_mean, yticklabels = pred

"""Exploratory Feature Selection

- Radius, perimeter and area are highly correlated, so we will use radius

- Compact_mean, concavity_mean and concavepoint_mean are highly correlated, so we will use concavity_me

Final total list of predictor variables:

- radius_mean, texture_mean, smoothness_mean, concavity_mean, symmetry_mean

Fitting a single logistic model, including all 5 independent variables. We record the time to compute t
"""

data_clean = data[['radius_mean', 'texture_mean', 'smoothness_mean', 'concavity_mean', 'symmetry_mean']]


print(data_clean)

Xs = data_clean
Y = data['diagnosis']

start=datetime.now()

logreg = LogisticRegression(solver='lbfgs')

scores = cross_validate(logreg, Xs, Y, cv=10, return_train_score=False)
scores
np.mean(scores['test_score'])

runtime = datetime.now()-start
print(runtime)

"""Loop through every possible combination of features and output best model. There are 31 possible com

Xss = ['radius_mean', 'texture_mean', 'smoothness_mean', 'concavity_mean', 'symmetry_mean']
combination_list = []
for L in range(1, 6):
```

```
    for subset in itertools.combinations(Xss, L):
        list_sub = list(subset)
        combination_list.append(list_sub)
Y = data['diagnosis']

print(combination_list)
#This is a list of all combinations of features
#A list of 31 lists

#Run all combinations of features in combination_list in logistic regression model
#Find all possible logistic regressions and CV scores and save the best accuracy score

for i in combination_list:
    max_score = 0

    X_sub = data[i]

    logreg2 = LogisticRegression(solver='lbfgs')

    scores = cross_validate(logreg2, X_sub, Y, cv=5, return_train_score=False)
    score = np.mean(scores['test_score'])
    if score > max_score:
        max_score = score
        best_i = i
        best_model = logreg2

print(best_i)
#Cross validation score --> average of 10 cross folds
print(max_score)

"""Now, let's use bivariate interaction terms in our model. We will now have 15 predictor terms (5 inde

original_x = ['radius_mean', 'texture_mean', 'smoothness_mean', 'concavity_mean', 'symmetry_mean']

interaction_list = []
interaction_value_data = []
data_interactions = data_clean.copy()
for subset in itertools.combinations(Xss, 2):
    list_sub = list(subset)
    name_col = str(list_sub[0]) +"*"+ str(list_sub[1])
    data_interactions[name_col] = data_interactions[list_sub[0]]*data_interactions[list_sub[1]]

    interaction_list.append(list_sub)


print(interaction_list)
print(len(interaction_list))

data_interactions.head()
```

```
#data_interactions is a dataframe that includes the 15 variables

"""Loop through every possible combination of features and output best model. There are 32,767 possible

start=datetime.now()
for i in combination_list_int:
    max_score = 0

    X_sub = data_interactions[i]

    logreg3 = LogisticRegression(solver='lbfgs')
    scores = cross_validate(logreg3, X_sub, Y, cv=10, return_train_score=False)
    score = np.mean(scores['test_score'])
    if score > max_score:
        max_score = score
        best_i = i
        best_model = logreg3

runtime = datetime.now()-start
print(runtime)
print(best_i)
print(max_score)

"""Comparing log regression to Random Forest"""

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score,confusion_matrix
from sklearn.metrics import accuracy_score

x_train, x_test, y_train, y_test = train_test_split(Xs, Y, test_size=0.3, random_state=1234)
clf_rf = RandomForestClassifier(random_state=1234)

clr_rf = clf_rf.fit(x_train,y_train)

ac = accuracy_score(y_test,clf_rf.predict(x_test))
print('Accuracy is: ',ac)
cm = confusion_matrix(y_test,clf_rf.predict(x_test))
sns.heatmap(cm,annot=True,fmt="d")

"""What if we selected features randomly? We will follow the monte carlo simulation idea here."""

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
import numpy as np
#data_clean = data[['radius_mean', 'texture_mean', 'smoothness_mean', 'concavity_mean', 'symmetry_mean'
#Y = data['diagnosis']

lr = LogisticRegression()
#measure the average accuracy in k-fold CV with all the features.
```

```
#np.mean(cross_val_score(lr,data_interactions,Y,cv=5,scoring="accuracy"))

result = []
N_search = 200
np.random.seed(123)
for i in range(N_search):
    N_columns =  list(np.random.choice(range(15),1)+1)
    columns = list(np.random.choice(range(15), N_columns, replace=False))


    #print(columns)

    all_columns = list(data_interactions)
    chosen_columns =[]
    index =0
    for i in range (0,len(columns)):
      chosen_columns.append(all_columns[columns[i]])

      index = index+1


    X = data_interactions[chosen_columns]

    scores = cross_val_score(lr,X, Y, cv=5, scoring="accuracy")
    result.append({'columns':columns,'performance':np.mean(scores)})
result.sort(key=lambda x : -x['performance'])

print(result)

"""Using original 10 variables"""

X_all_mean = data[['radius_mean','texture_mean','perimeter_mean','area_mean','smoothness_mean','compactn

full_combination_list = []
for L in range(1, 10):
    for subset in itertools.combinations(X_all_mean, L):
        list_sub = list(subset)
        full_combination_list.append(list_sub)
Y = data['diagnosis']

print(full_combination_list)

"""Original 10 variables fitting and scores"""

start=datetime.now()
for i in full_combination_list:
    max_score = 0

    X_sub = data[i]
```

```
    logreg2 = LogisticRegression(solver='lbfgs')
    scores = cross_validate(logreg2, X_sub, Y, cv=10, return_train_score=False)
    score = np.mean(scores['test_score'])
    if score > max_score:
        max_score = score
        best_i = i
        best_model = logreg2

runtime = datetime.now()-start

print(best_i)
print(max_score)
#How long it took
print(runtime)
```