Data-X Spring 2019: Homework 06

Name :

Anish Saha

SID :

26071616

Course (IEOR 135/290) :

Machine Learning

In this homework, you will do some exercises with prediction. We will cover these algorithms in class, but this is for you to have some hands on with these in scikit-learn. You can refer - https://github.com/ikhlaqsidhu/data-x/blob/master/05a-tools-predicition-titanic/titanic.ipynb (https://github.com/ikhlaqsidhu/data-x/blob/master/05a-tools-predicition-titanic/titanic.ipynb)

Display all your outputs.

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
 # machine learning libraries
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import Perceptron
from sklearn.tree import DecisionTreeClassifier
```

1. Read `diabetesdata.csv` file into a pandas dataframe. About the data:

1. **TimesPregnant**: Number of times pregnant
2. **glucoseLevel**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. **BP**: Diastolic blood pressure (mm Hg)
4. **insulin**: 2-Hour serum insulin (mu U/ml)
5. **BMI**: Body mass index (weight in kg/(height in m)^2)
6. **pedigree**: Diabetes pedigree function
7. **Age**: Age (years)
8. **IsDiabetic**: 0 if not diabetic or 1 if diabetic)

In [3]:
```
#Read data & print the head
df = pd.read_csv("diabetesdata.csv")
df.head()
```

Out[3]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	6	148.0	72	0	33.6	0.627	50.0	1
1	1	NaN	66	0	26.6	0.351	31.0	0
2	8	183.0	64	0	23.3	0.672	NaN	1
3	1	NaN	66	94	28.1	0.167	21.0	0
4	0	137.0	40	168	43.1	2.288	33.0	1

2. Calculate the percentage of Null values in each column and display it.

In [4]:
```
df.isna().sum() / len(df)
```

Out[4]:
```
TimesPregnant    0.000000
glucoseLevel     0.044271
BP               0.000000
insulin          0.000000
BMI              0.000000
Pedigree         0.000000
Age              0.042969
IsDiabetic       0.000000
dtype: float64
```

3. Split `data` into `train_df` and `test_df` with 15% as test.

In [5]:
```
np.random.seed(999)

idx = np.random.rand(len(df)) < 0.85
train_df, test_df = df[idx], df[~idx]
len(df), len(train_df), len(test_df)
```

Out[5]:
```
(768, 662, 106)
```

4. Display the means of the features in train and test sets. Replace the null values in `train_df` and `test_df` with the mean of EACH feature column separately for train and test. Display head of the dataframes.

In [6]:
```
print(train_df.mean())
print(test_df.mean())
train_df.fillna(train_df.mean(), inplace=True)
test_df.fillna(test_df.mean(), inplace=True)
```

```
TimesPregnant      3.983384
glucoseLevel     120.787639
BP                68.978852
insulin           80.598187
BMI               31.998943
Pedigree           0.472144
Age               33.606635
IsDiabetic         0.345921
dtype: float64
TimesPregnant      2.981132
glucoseLevel     122.417476
BP                69.896226
insulin           74.811321
BMI               31.952830
Pedigree           0.470208
Age               31.784314
IsDiabetic         0.367925
dtype: float64
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pa
ckages/pandas/core/generic.py:5430: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  self._update_inplace(new_data)
```

5. Split `train_df` & `test_df` into `X_train`, `Y_train` and `X_test`, `Y_test`. `Y_train` and `Y_test` should only have the column we are trying to predict, `IsDiabetic`.

In [7]:
```
X_train, X_test = train_df.drop("IsDiabetic", axis=1), test_df.drop("IsD
iabetic", axis=1)
y_train, y_test = train_df["IsDiabetic"], test_df["IsDiabetic"]
print("X_train")
print(X_train.head())
print("\nX_test")
print(X_test.head())
print("\ny_train")
print(y_train.head())
print("\ny_test")
print(y_test.head())
```

```
X_train
   TimesPregnant  glucoseLevel  BP  insulin   BMI  Pedigree        Age
0              6    148.000000  72        0  33.6     0.627  50.000000
1              1    120.787639  66        0  26.6     0.351  31.000000
2              8    183.000000  64        0  23.3     0.672  33.606635
3              1    120.787639  66       94  28.1     0.167  21.000000
4              0    137.000000  40      168  43.1     2.288  33.000000

X_test
    TimesPregnant  glucoseLevel  BP  insulin   BMI  Pedigree   Age
16              0    122.417476  84      230  45.8     0.551  31.0
26              7    147.000000  76        0  39.4     0.257  43.0
31              3    158.000000  76      245  31.6     0.851  28.0
40              3    180.000000  64       70  34.0     0.271  26.0
41              7    133.000000  84        0  40.2     0.696  37.0

y_train
0    1
1    0
2    1
3    0
4    1
Name: IsDiabetic, dtype: int64

y_test
16    1
26    1
31    1
40    0
41    0
Name: IsDiabetic, dtype: int64
```

6. Use this dataset to train perceptron, logistic regression and random forest models using 15% test split. Report training and test accuracies. Try different hyperparameter values for these models and see if you can improve your accuracies.

In [8]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import GridSearchCV

# 6a. Logistic Regression
mod1 = LogisticRegression()
mod1.fit(X_train, y_train)

print("Logistic Regression Model Performance:")
y_pred_train = mod1.predict(X_train)
train_rmse = accuracy_score(y_train, y_pred_train)
print("Training Accuracy: " + str(train_rmse))
y_pred_test = mod1.predict(X_test)
test_rmse = accuracy_score(y_test, y_pred_test)
print("Test Accuracy: " + str(test_rmse))

print("\n")

penalty = ['l1', 'l2']
C = np.logspace(0, 5, 10)
hyperparameters = dict(C=C, penalty=penalty)
clf = GridSearchCV(mod1, hyperparameters, cv=5, verbose=0)
mod2 = clf.fit(X_train, y_train) # optimized hyperparameters

print("Optimized Logistic Regression Model Performance:")
y_pred_train = mod2.predict(X_train)
train_rmse = accuracy_score(y_train, y_pred_train)
print("Training Accuracy: " + str(train_rmse))
y_pred_test = mod2.predict(X_test)
test_rmse = accuracy_score(y_test, y_pred_test)
print("Test Accuracy: " + str(test_rmse))
```

```
Logistic Regression Model Performance:
Training Accuracy: 0.7658610271903323
Test Accuracy: 0.7830188679245284


Optimized Logistic Regression Model Performance:
Training Accuracy: 0.7719033232628398
Test Accuracy: 0.7924528301886793
```

In [9]:
```
from sklearn.neural_network import MLPClassifier

# 6b. Perceptron
mod3 = Perceptron()
mod3.fit(X_train, y_train)

print("Perceptron Model Performance:")
y_pred_train = mod3.predict(X_train)
train_rmse = accuracy_score(y_train, y_pred_train)
print("Training Accuracy: " + str(train_rmse))
y_pred_test = mod3.predict(X_test)
test_rmse = accuracy_score(y_test, y_pred_test)
print("Test Accuracy: " + str(test_rmse))

hyperparameters = { 'alpha': [0.0001, 0.05], 'tol': [None],
                    'fit_intercept': [True, False],
                    'max_iter': [100, 1000],
                    'penalty': penalty }
clf = GridSearchCV(mod3, hyperparameters, cv=5, verbose=0)
mod4 = clf.fit(X_train, y_train) # optimized hyperparameters

print("Perceptron Model Performance:")
y_pred_train = mod4.predict(X_train)
train_rmse = accuracy_score(y_train, y_pred_train)
print("Training Accuracy: " + str(train_rmse))
y_pred_test = mod4.predict(X_test)
test_rmse = accuracy_score(y_test, y_pred_test)
print("Test Accuracy: " + str(test_rmse))

print("\n")

# 6b. Multi-Layer Perceptron
mod3 = MLPClassifier()
mod3.fit(X_train, y_train)

print("Multi-Layer Perceptron Model Performance:")
y_pred_train = mod3.predict(X_train)
train_rmse = accuracy_score(y_train, y_pred_train)
print("Training Accuracy: " + str(train_rmse))
y_pred_test = mod3.predict(X_test)
test_rmse = accuracy_score(y_test, y_pred_test)
print("Test Accuracy: " + str(test_rmse))

print("\n")

hyperparameters = { 'hidden_layer_sizes': [(50,50,50), (50,100,50), (100
,)],
                    'alpha': [0.0001, 0.05], 'activation': ['tanh', 'rel
u'],
                    'learning_rate': ['constant','adaptive'] }
clf = GridSearchCV(mod3, hyperparameters, cv=5, verbose=0)
mod4 = clf.fit(X_train, y_train) # optimized hyperparameters

print("Optimized Multi-Layer Perceptron Model Performance:")
y_pred_train = mod4.predict(X_train)
train_rmse = accuracy_score(y_train, y_pred_train)
print("Training Accuracy: " + str(train_rmse))
```

```
y_pred_test = mod4.predict(X_test)
test_rmse = accuracy_score(y_test, y_pred_test)
print("Test Accuracy: " + str(test_rmse))
```

```
Perceptron Model Performance:
Training Accuracy: 0.3776435045317221
Test Accuracy: 0.39622641509433965

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pa
ckages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning:
max_iter and tol parameters have been added in <class 'sklearn.linear_m
odel.perceptron.Perceptron'> in 0.19. If both are left unset, they defa
ult to max_iter=5 and tol=None. If tol is not None, max_iter defaults t
o max_iter=1000. From 0.21, default max_iter will be 1000, and default
tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)

Perceptron Model Performance:
Training Accuracy: 0.6993957703927492
Test Accuracy: 0.7169811320754716


Multi-Layer Perceptron Model Performance:
Training Accuracy: 0.6963746223564955
Test Accuracy: 0.7075471698113207


Optimized Multi-Layer Perceptron Model Performance:
Training Accuracy: 0.7537764350453172
Test Accuracy: 0.7452830188679245
```

In [10]:
```
# 6c. Random Forest
mod5 = RandomForestClassifier()
mod5.fit(X_train, y_train)

print("Random Forest Model Performance:")
y_pred_train = mod5.predict(X_train)
train_rmse = accuracy_score(y_train, y_pred_train)
print("Training Accuracy: " + str(train_rmse))
y_pred_test = mod5.predict(X_test)
test_rmse = accuracy_score(y_test, y_pred_test)
print("Test Accuracy: " + str(test_rmse))

print("\n")

max_depth = [int(x) for x in np.linspace(10, 110, num = 10)]
max_depth.append(None)
hyperparameters = { 'max_depth': max_depth, 'min_samples_split': [2, 5,
10],
                    'max_features': ['auto', 'sqrt'], 'bootstrap': [True
, False] }
clf = GridSearchCV(mod5, hyperparameters, cv=5, verbose=0)
mod6 = clf.fit(X_train, y_train) # optimized hyperparameters

print("Optimized Random Forest Model Performance:")
y_pred_train = mod6.predict(X_train)
train_rmse = accuracy_score(y_train, y_pred_train)
print("Training Accuracy: " + str(train_rmse))
y_pred_test = mod6.predict(X_test)
test_rmse = accuracy_score(y_test, y_pred_test)
print("Test Accuracy: " + str(test_rmse))
```

```
Random Forest Model Performance:
Training Accuracy: 0.9833836858006042
Test Accuracy: 0.7830188679245284


Optimized Random Forest Model Performance:
Training Accuracy: 0.918429003021148
Test Accuracy: 0.8018867924528302
```

7. For your logistic regression model -

a . Compute the log probability of classes in `IsDiabetic` for the first 10 samples of your train set and display it. Also display the predicted class for those samples from your logistic regression model trained before.

In [11]:
```
print("Log Probabilities for first 10 training samples")
print(mod2.predict_log_proba(X_train)[:10])
print("\n")
print("Predicted Class for first 10 training samples")
print(mod2.predict(X_train)[:10])
```

```
Log Probabilities for first 10 training samples
[[-1.10942339 -0.40010307]
 [-0.17359275 -1.83658434]
 [-1.61647411 -0.22139221]
 [-0.1537507  -1.94831341]
 [-1.70591041 -0.20041259]
 [-0.18852481 -1.7613076 ]
 [-0.07912209 -2.57606337]
 [-0.99331984 -0.46258348]
 [-1.47067037 -0.26106793]
 [-0.05511131 -2.92582943]]


Predicted Class for first 10 training samples
[1 0 1 0 1 0 0 1 1 0]
```

b . Now compute the log probability of classes in `IsDiabetic` for the first 10 samples of your test set and display it. Also display the predicted class for those samples from your logistic regression model trained before. (using the model trained on the training set)

In [12]:
```
print("Log Probabilities for first 10 training samples")
print(mod2.predict_log_proba(X_test)[:10])
print("\n")
print("Predicted Class for first 10 training samples")
print(mod2.predict(X_test)[:10])
```

```
Log Probabilities for first 10 training samples
[[-0.47680148 -0.96960119]
 [-1.21847992 -0.35052146]
 [-0.88339004 -0.5333766 ]
 [-1.35006788 -0.30005519]
 [-1.03682156 -0.43785386]
 [-0.04436754 -3.137349  ]
 [-0.08413521 -2.51710276]
 [-0.20566261 -1.68258782]
 [-0.01692857 -4.08720499]
 [-0.02651673 -3.64320836]]


Predicted Class for first 10 training samples
[0 1 1 1 1 0 0 0 0 0]
```

c . What can you interpret from the log probabilities and the predicted classes?

In the outputs above, the first column represents the log probability that the sample is of class 0 [`IsDiabetic` = 0], while the second column represents the log probability that the sample is of class 1 [`IsDiabetic` = 1]. The probability that a sample is of a certain class is computed using the formula:

$$P(\text{ sample}_j \text{is not diabetic}) = e^{\,a[j][0]} \;\mid\; P(\text{ sample}_j \text{is diabetic}) = e^{\,a[j][1]}$$
$$\text{for the } j^{th} \text{ sample, and } a \text{ is the array displayed above}$$

The predicted class corresponds to the column with the higher log probability (and consequently, higher probability) value – or in other words, whichever log probability value is closer to 0 since all log probability values are negative. This can be confirmed by observing the outputs above.

8. Is mean imputation is the best type of imputation (as we did in 4.) to use? Why or why not? What are some other ways to impute the data?

Mean imputation is not the best type of imputation to use. This is because it often does not preserve relationships between variables (imputed values have zero correlation with other variables), presents biased metrics of standard error and variance, and can result in a biased sample mean. The only advantage is that it preserves the sample size. Some other ways to impute the data include hot-deck imputation, cold-deck imputation, regression imputation, and multiple imputation (ex: MICE, using chained equations, for when data is randomly missing).

Extra Credit (2 pts) - MANDATORY for students enrolled in IEOR 290

9. Implement the K-Nearest Neighbours ([https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm (https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)) algorithm for k=1 from scratch in python (do not use KNN from existing libraries). KNN uses Euclidean distance to find nearest neighbors. Split your dataset into test and train as before. Also fill in the null values with mean of features as done earlier. Use this algorithm to predict values for 'IsDiabetic' for your test set. Display your accuracy.

In [13]:
```
# K-Nearest Neighbors Classifier for k=1
class KNN_Classifier():
    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def euclidean_dist(self, x1, x2):
        distance = 0
        for i in range(len(x1)):
            distance = distance + (x1[i] - x2[i])**2
        return distance

    def k_nearest(self, row, k=1):
        best_dist = self.euclidean_dist(row, self.X_train[0])
        best_idx = 0
        for i in range(k, len(self.X_train)):
            dist = self.euclidean_dist(row, self.X_train[i])
            if dist < best_dist:
                best_dist = dist
                best_idx = i
        return self.y_train[best_idx]

    def predict(self, X_test, k=1):
        result = []
        for row in X_test:
            label = self.k_nearest(row, k)
            result.append(label)
        return result

mod7 = KNN_Classifier()
mod7.fit(X_train.values, y_train.values)

print("MANUAL IMPLEMENTATION | K-Nearest Neighbors Model Performance:")
y_pred_train = mod7.predict(X_train.values)
train_rmse = accuracy_score(y_train, y_pred_train)
print("Training Accuracy: " + str(train_rmse))
y_pred_test = mod7.predict(X_test.values)
test_rmse = accuracy_score(y_test, y_pred_test)
print("Test Accuracy: " + str(test_rmse))
```

```
MANUAL IMPLEMENTATION | K-Nearest Neighbors Model Performance:
Training Accuracy: 1.0
Test Accuracy: 0.7264150943396226
```

In [14]:
```
# Checking Implementation against SciKitLearn Library Implementation
from sklearn.neighbors import KNeighborsClassifier

mod8 = KNeighborsClassifier()
mod8.fit(X_train, y_train)

print("SKLEARN IMPLEMENTATION | K-Nearest Neighbors Model Performance:")
y_pred_train = mod8.predict(X_train.values)
train_rmse = accuracy_score(y_train, y_pred_train)
print("Training Accuracy: " + str(train_rmse))
y_pred_test = mod8.predict(X_test.values)
test_rmse = accuracy_score(y_test, y_pred_test)
print("Test Accuracy: " + str(test_rmse))
```

```
SKLEARN IMPLEMENTATION | K-Nearest Neighbors Model Performance:
Training Accuracy: 0.8096676737160121
Test Accuracy: 0.7264150943396226
```