Homework #3

_____Qiyu Wang_____

(put your name above (incl. any nicknames))

Total grade: _____ out of ___100___ points

ATTENTION: HW3 has two parts. Please first complete the Quiz "HW3_Part1" on Canvas. Then, proceed with Part 2 in the following page. You will need to submit (a) a PDF file with your answers and screenshots of Python code snippets and (b) the Python code.

(100 points) [Mining publicly available data] Use Python for this Exercise.

Please use the dataset on breast cancer research from this link: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data We have worked with this dataset in HW2. The description of the data and attributes can be found at this link: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names . Each record of the data set represents a different case of breast cancer. Each case is described with 30 real-valued attributes: attribute 1 represents case id, attributes 3-32 represent various physiological characteristics, and attribute 2 represents the type (benign or malignant). If the dataset has records with missing values, you can filter out these records using Python. Alternatively, if the data set has missing values, you could infer the missing values.

[We have seen this data before – No need to explore the data for this exercise]

a) **We would like to perform a predictive modeling analysis on this same dataset using the a) decision tree, b) the k-NN technique and c) the logistic regression technique. Using the nested cross-validation technique, try to optimize the parameters of your classifiers in order to improve the performance of your classifiers (i.e., f1-score) as much as possible. Please make sure to always use a random state of "42" whenever applicable. What are your optimal parameters and what is the corresponding performance of these classifiers? Please provide screenshots of your code and explain the process you have followed.**

[part a is worth 25 points in total:

Decision Tree

```
[370]: gs = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                         param_grid=[{'min_samples_leaf': list(range(1, 30)),
                                      'max_depth': list(range(3, 11)),
                                     "criterion": ["gini","entropy"]}],
                         scoring='f1_macro', # Specifying multiple metrics for evaluation
                         cv=inner_cv)

[371]: gs_dt = gs.fit(X,y)

[372]: print("\n Parameter Tuning #1")
       print("Non-nested CV f1_macro: ", gs_dt.best_score_)
       print("Optimal Parameter: ", gs_dt.best_params_)
       print("Optimal Estimator: ", gs_dt.best_estimator_)
       nested_score_gs_dt = cross_val_score(gs_dt, X=X, y=y, cv=outer_cv)
       print("Nested CV f1_macro: ",nested_score_gs_dt.mean(), " +/- ", nested_score_gs_dt.std())
```

```
 Parameter Tuning #1
Non-nested CV f1_macro:  0.9546228099297324
Optimal Parameter:  {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 6}
Optimal Estimator:  DecisionTreeClassifier(max_depth=5, min_samples_leaf=6, random_state=42)
Nested CV f1_macro:  0.9358106154761549  +/-  0.02102737853698179
```

Here I did a grid search on three parameters including "min_samples_leaf" within 1~29, "max_depth" within 3~10, "criterion" of "gini" or "entropy".
I choose the metric of f1-measure, and grid search report me a parameter combination of {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 6}.
The f1-measure reported is at 0.936 with a confidence interval of +/- 0.021

KNN

[347]:
```
sc = StandardScaler()
sc.fit(X)
X_std = sc.transform(X)
```

[376]:
```
gs = GridSearchCV(estimator=neighbors.KNeighborsClassifier(metric='minkowski'),
                  param_grid=[{'n_neighbors': list(range(1, 21)),
                               "p": [1,2],
                               'weights':['uniform','distance']}],
                  scoring='f1_macro',
                  cv=inner_cv)
```

[377]:
```
gs_knn = gs.fit(X_std,y)
```

[378]:
```
print("\n Parameter Tuning #3")
print("Non-nested CV f1_macro: ", gs_knn.best_score_)
print("Optimal Parameter: ", gs_knn.best_params_)
print("Optimal Estimator: ", gs_knn.best_estimator_)
nested_score_gs_knn = cross_val_score(gs_knn, X=X_std, y=y, cv=outer_cv)
print("Nested CV f1_macro: ",nested_score_gs_knn.mean(), " +/- ", nested_score_gs_knn.std())
```

```
 Parameter Tuning #3
Non-nested CV f1_macro:  0.9696104312219758
Optimal Parameter:  {'n_neighbors': 8, 'p': 2, 'weights': 'distance'}
Optimal Estimator:  KNeighborsClassifier(n_neighbors=8, weights='distance')
Nested CV f1_macro:  0.9635898901904698  +/-  0.009706404783664106
```

Here I did a grid search on three parameters including “'n_neighbors” within 1~20, “p” (distance mertics) of 1 or 2 which stands for Manhattan and Euclidean, “'weights'” of “distance” or “uniform”, which gives a weighted effect to predictions.
I choose the metric of f1-measure, and grid search report me a parameter combination of {'n_neighbors': 8, 'p': 2, 'weights': 'distance'}
The f1-measure reported is at 0.964 with a confidence interval of +/- 0.01.

Logistic

[343]:
```
from  warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)
```

[373]:
```
gs = GridSearchCV(estimator=LogisticRegression(random_state=42),
                  param_grid=[{'C': [ 0.00001, 0.0001, 0.001, 0.01, 0.1 ,1 ,10 ,100, 1000, 10000, 100000, 1000000],
                               "penalty":["l2","l1"], 'solver' : ['liblinear','saga']}],
                  scoring='f1_macro',
                  cv=inner_cv)
```

[374]:
```
gs_lr = gs.fit(X,y)
```

[375]:
```
print("\n Parameter Tuning #2")
print("Non-nested CV f1_macro: ", gs_lr.best_score_)
print("Optimal Parameter: ", gs_lr.best_params_)
print("Optimal Estimator: ", gs_lr.best_estimator_)
nested_score_gs_lr = cross_val_score(gs_lr, X=X, y=y, cv=outer_cv)
print("Nested CV f1_macro:",nested_score_gs_lr.mean(), " +/- ", nested_score_gs_lr.std())
```

```
 Parameter Tuning #2
Non-nested CV f1_macro:  0.9661594539900953
Optimal Parameter:  {'C': 100, 'penalty': 'l1', 'solver': 'liblinear'}
Optimal Estimator:  LogisticRegression(C=100, penalty='l1', random_state=42, solver='liblinear')
Nested CV f1_macro: 0.9458561807238033  +/-  0.0336060096841347
```

Here I did a grid search on three parameters including “C” within [0.00001, 0.0001, 0.001, 0.01, 0.1 ,1 ,10 ,100, 1000, 10000, 100000, 1000000], “'penalty'” (L1 or L2 regularization) of l1 or l2, “'solver'” of “'liblinear'” or “saga”.
I choose the metric of f1-measure, and grid search report me a parameter combination of {'C': 100, 'penalty': 'l1', 'solver': 'liblinear'}
The f1-measure reported is at 0.946 with a confidence interval of +/- 0.034.

Based on the performance score reported, KNN model is the best out of all, which has better f1-measure and more stable(lower standard deviation based on cross-validation scores). Since the problem is diagnose of cancer (M/B), I believe that recall/precision is the most important metrics here. Because it can get extremely risky and costly to miss a true malignant, which may risk a person's life. By having a high precision/recall score, we can make sure that we don't make as less as possible mispredictions of true malignant, which can delay a patient's treatment and reduce its chance of recovery. So for here, the KNN regression had a highest f1 of 0.964 on malignant and very low variation of a std of 0.01. Thus, KNN model presents as the best model here.

b) Build and visualize a learning curve for the logistic regression technique (visualize the performance for both training and test data in the same plot). Please provide screenshots of your code and explain the process you have followed.

[part b is worth 25 points in total:
8 points for correct visualization of learning curve for in-sample sample performance – show the performance for 10 different sizes - provide screenshots of your code and explain the process you have followed.
8 points for correct visualization of learning curve for out-sample sample performance – show the performance for 10 different sizes - provide screenshots of your code and explain the process you have followed.
9 points for discussing what we can learn from this specific learning curve – what are the insights that can be drawn]

Learning Curve

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, score = "accuracy",
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 10)):
    """

# Class learning_curve determines cross-validated training and test scores for different training set sizes
train_sizes, train_scores, test_scores = learning_curve(
    estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, scoring = score)
```

I used the function prof. todri defined in notes. But I added a parameter of score to it to value it on F-1 (and I set the default to accuracy).

```
np.random.seed(42)

title = "Learning Curve (Logistic Regression)"

# Class ShuffleSplit is a random permutation cross-validator
# Parameter n_splits = Number of re-shuffling & splitting iterations
# Parameter test_size = represents the proportion of the dataset to include in the test split
# Parameter random_state = the seed used by the random number generator
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
estimator = LogisticRegression(random_state=42, solver='liblinear',C = 100, penalty = 'l1') #
# Plots the learning curve based on the previously defined function for the logistic regressi
plot_learning_curve(estimator, title, X, y, (0.9, 1.01), cv=cv, n_jobs=4, score = "f1_macro")

plt.show() # Display the figure
```

Then I runed the learning curve function on the optimal logistic model I found through grid search from the previous step.

The model is: LogisticRegression(C=100, penalty='l1', random_state=42, solver='liblinear')

We can see that the score on training and cv are far apart at beginning and getting closer and closer to each other. Which means the model is overfitting at the beginning, and as sample size increasing, it have less and less overfitting problem and therefore perform

better on cv. The curve stopped at 500-something cases, but we see that there's a trend for two curves to keep getting closer, so as we obtain more data, we can expect this model to perform even better. Overall, logistic is doing a pretty good job and end up with a F-1 above 0.96 based on the graph.

Also the variance (confidence interval) can be higher on training set, but there's nothing abnormal with that.

c) Build a fitting graph for different depths of the decision tree (visualize the performance for both training and test data in the same plot). Please provide screenshots of your code and explain the process you have followed.

[part c is worth 25 points in total:
8 points for correct visualization of fitting graph for in-sample sample performance – show the performance for 15 different values- provide screenshots of your code and explain the process you have followed
8 points for correct visualization of fitting graph for out-of-sample performance – show the performance for 15 different values- provide screenshots of your code and explain the process you have followed
9 points for discussing what we can learn from this specific fitting graph – what are the insights that can be drawn]

Fitting curve

```
# Fitting curve (aka validation curve)
# Determine training and test scores for varying parameter values.
from sklearn.model_selection import validation_curve
# Split validation
from sklearn.model_selection import train_test_split

np.random.seed(42) #the seed used by the random number generator for np
```

```
param_range=list(range(1, 15))
```

```
# Determine training and test scores for varying parameter values.
train_scores, test_scores = validation_curve(
                estimator=DecisionTreeClassifier(random_state=42),
                X=x_train,
                y=y_train,
                param_name="max_depth",
                param_range=param_range,
                cv=10,      #10-fold cross-validation
                scoring="f1_macro",
                n_jobs=4) # Number of CPU cores used when parallel


# Cross validation statistics for training and testing data (mean
train_mean = np.mean(train_scores, axis=1) # Compute the arithmeti
train_std = np.std(train_scores, axis=1)   # Compute the standard
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)
```

Same as before, I used the function from prof. Todri's nots, but I changed again the measurement to F1-measure to stay constant with previous tests. Moreover, I used max_depth for complexity measure for this particular graph, and it ranges from 1~15.

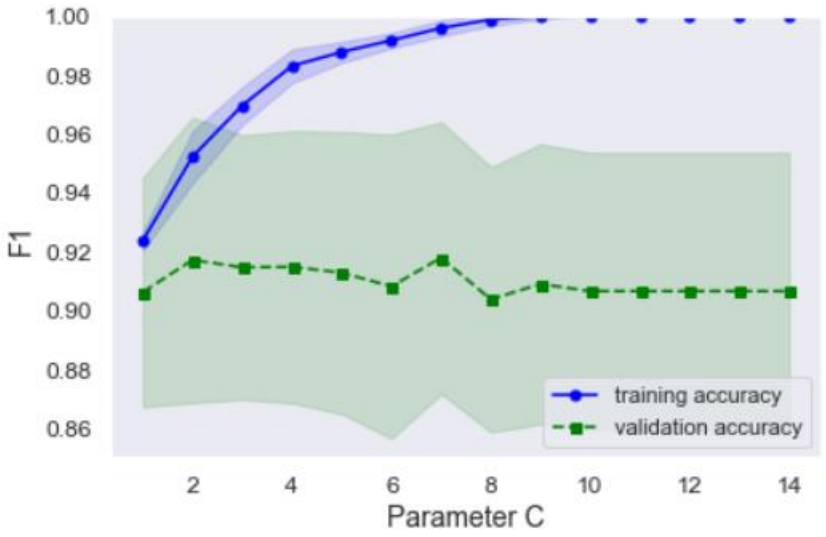
```
# Plot train accuracy means of cross-validation for all
plt.plot(param_range, train_mean,
         color='blue', marker='o',
         markersize=5, label='training accuracy')

# Fill the area around the line to indicate the size of
plt.fill_between(param_range, train_mean + train_std,
                 train_mean - train_std, alpha=0.15,
                 color='blue')

# Plot test accuracy means of cross-validation for all
plt.plot(param_range, test_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='validation accuracy')

# Fill the area around the line to indicate the size of
plt.fill_between(param_range,
                 test_mean + test_std,
                 test_mean - test_std,
                 alpha=0.15, color='green')

# Grid and Axes Titles
plt.grid()
plt.legend(loc='lower right')
plt.xlabel('Parameter C')
plt.ylabel('F1')
plt.ylim([0.85, 1.0]) # y limits in the plot
plt.tight_layout()
# plt.savefig('Fitting_graph_LR.png', dpi=300)
plt.show()          # Display the figure
```



The fitting graph come from bottom up which is normal for tree models (where it starts at 0.) The curve for cv first climbs up along with training curve, and them have a slight trend of going downward, which tells us high complexity doesn't work well with this dataset and model. The optimal depth is pretty low, and if we keep increasing the depth over that optimal point, the model overfits. We see a optimal point from the graph at 5, which is the best depth score. The training curve keeps climbing up show its keep learning from the data, but the downward of cv curve on the other hand should the trend of overfitting. Finally, the optimal point return a F1 at 0.92 based on the graph.

d) Create an ROC curve for k-NN, decision tree, and logistic regression. Discuss the results. Which classifier would you prefer to choose? Please provide screenshots of your code and explain the process you have followed.

[part d is worth 25 points in total:
5 points for correct visualization of ROC graph for kNN – use optimal kNN from part a
5 points for correct visualization of ROC graph for Decision Tree – use optimal Decision Tree from part a
5 points for correct visualization of ROC graph for Logistic Regression – use optimal Logistic Regression from part a
2 points for showing all the ROC graphs in one single plot
3 points for showing AUC estimators in the ROC graph
5 points for discussing and correctly identifying which classifier you would use]

```
clf1 = LogisticRegression(penalty='l1',
                          C=100,
                          random_state=42,
                          solver='liblinear')

# Decision Tree Classifier
clf2 = DecisionTreeClassifier(max_depth=5,
                              criterion='gini',
                              min_samples_leaf = 6,
                              random_state=42)

# kNN Classifier
clf3 = KNeighborsClassifier(n_neighbors=8,
                            p=2,
                            metric='minkowski',
                           weights = "distance")

# Label the classifiers
clf_labels = ['Logistic regression', 'Decision tree', 'kNN']
all_clf = [clf1, clf2, clf3]
```

I first define the tree model based my optimal models from part 1:

DecisionTreeClassifier(max_depth=5, min_samples_leaf=6, random_state=42)

LogisticRegression(C=100, penalty='l1', random_state=42, solver='liblinear')

KNeighborsClassifier(n_neighbors=8, weights='distance')

```
print('10-fold cross validation:\n')
# Note: We are assuming here that the data is standardi
for clf, label in zip([clf1, clf2, clf3], clf_labels): 
    scores = cross_val_score(estimator=clf,  #Estimate 
                             X=X,
                             y=y,
                             cv=10,
                             scoring='roc_auc')
    print("ROC AUC: %0.2f (+/- %0.2f) [%s]" #Print pefo
          % (scores.mean(), scores.std(), label))
```

```
10-fold cross validation:

ROC AUC: 0.99 (+/- 0.01) [Logistic regression]
ROC AUC: 0.96 (+/- 0.03) [Decision tree]
ROC AUC: 0.97 (+/- 0.02) [kNN]
```

Then I run the functions came with Sklearn, and got cross validation scores and AUC scores for all three models.

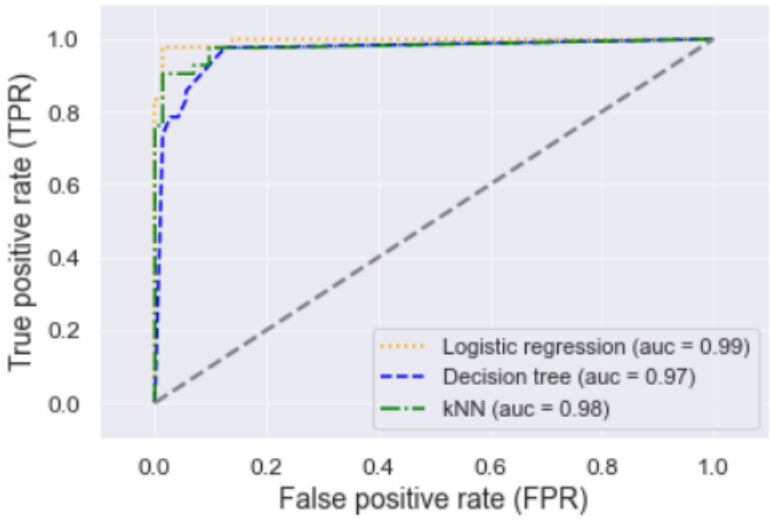
```
colors = [ 'orange', 'blue', 'green']      # Colors for vi
linestyles = [':', '--', '-.', '-']        # Line styles f
for clf, label, clr, ls in zip(all_clf,
               clf_labels, colors, linestyles):

    # Assuming the label of the positive class is 1 and do
    y_pred = clf.fit(X_train,
                     y_train).predict_proba(X_test)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_true=y_test, # Buil
                                     y_score=y_pred)
    roc_auc = auc(x=fpr, y=tpr)                # Compute A
    plt.plot(fpr, tpr,                         # Plot ROC
             color=clr,
             linestyle=ls,
             label='%s (auc = %0.2f)' % (label, roc_auc))

plt.legend(loc='lower right')    # Where to place the lege
plt.plot([0, 1], [0, 1], # Visualize random classifier
         linestyle='--',
         color='gray',
         linewidth=2)

plt.xlim([-0.1, 1.1])   #limits for x axis
plt.ylim([-0.1, 1.1])   #limits for y axis
plt.grid(alpha=0.5)
plt.xlabel('False positive rate (FPR)')
plt.ylabel('True positive rate (TPR)')


#plt.savefig('ROC_all_classifiers', dpi=300)
plt.show()
```



Finally, I came up with the graph which contains all three ROC curves. From the curve we see that Decision tree is the worst model here in all sense. The other two perform different at different cutoff ratios. I would say logistic regression is the best overall model, since it has a high AUC score which means it does a better job at capturing malignant. But KNN is also a good option at some of the cutoff points. Since the ROC doesn't include conditions(dataset proportion information and cost/benefit information) we can not tell which one of them is the best for this problem. But purely based on capturing malignant, the Logistic is the winner here by having a AUC of 0.99. However, when we include all information, such as the standard deviation scores we seen in part 1, KNN might be a better option since it's more stable. Also we need to

fight out whats the better cutoff point after includes conditions, so we can tell which model work well at that specific level of cutoff.