Project 2 report:

- Project 2, for course CIS6930 – Applied Machine Learning using Python, is a two-fold programming assignment. First is a classification/regression problem and second one is a clustering problem.
- Please find the individual report for these problems in the following sections.

1. NBA Player Longevity Prediction

Questions:

1. When you prepare the data for training the models, did you discover any attribute to remove or any new attribute to add? If you did, discuss the choices.

Answer: During the data preprocessing step I realized that field goal percent (FG%) is calculated using field goals made (FGM) / field goal attempts (FGA). Similar is the case with three point percent (3P%) and free throw percent (FT%). I tried both scenarios, 1) where only 'Name' and 'TAR' was deleted for the features data and 2) where 'FG%', '3P%' and 'FT%' were deleted too along with name and target. For most of the models, results were better for the first case where only 'Name' and 'TAR' was deleted.

Duplicate records : There are 12 duplicate records(for all attributes) in the given dataset. Generally its not recommended to delete duplicates from the dataset since every record has a weight to it, unless there is a strong reason. My rationale: The provided dataset represents a player's rookie statistics, stats of that player's first season which implies every player should have 1 record. Multiple rows having different values for the same player is completely acceptable(for ex: name "Charles Bradley") but that is not the case with dataset. Example: same data for players with name '' Charles Smith"," Charles Jones"," Chris Smith" etc).
Every model performed better when these duplicate records were deleted.

2. Normalizing (a.k.a., scaling) features is desirable for distance-based models, e.g., knearest neighbors. Did you try feature normalization for some of the models? If so, talk about if any improvement.

Answer: During the initial analysis I tried normalizing the features for all the models. Observation : I didn't see any improvement in the model behavior with the use of normalization.

Model	f1 Mean : (No normalization)	f1 Mean : (Normalizer())
KNeighborsClassifier	0.7234	0.7209
RandomForestClassifier	0.7456	0.7456
LogisticRegression	0.7691	0.7691
MLPClassifier	0.7199	0.7199

3. Regularization is a common practice to battle overfitting. How is varying the penalty parameter in logistic regression affect the performance F1 score on testing? (The logistic regression penalty parameter may be 'none', 'l1', 'l2' or 'elasticnet'.)

Answer: Penalty 'elasticnet' model was not converging most of the times with the error "ConvergenceWarning: The max_iter was reached which means the coef_ did not converge "the coef_ did not converge", ConvergenceWarning)".

On basis of f1 score ‘l1’ performed better most of the times. F1 scores for ‘l2’ and ‘elasticnet’ (with “newton-cg” solver) were comparable to each other most of the times.
Below are the details from the initial analysis done in this regard.

Feature scaling	penalty='l1'	penalty='l2'	penalty='elasticnet'	penalty='none' +newton-cg	Performance(f1 score)
No processing	0.7691	0.7664	0.7618	0.7661	l1 > l2>n>e
Standard scaler	0.7701	0.7668	0.7677	0.7661	l1>e>l2>n
Normalizer	0.7691	0.7664	0.7618	0.7661	l1>l2>none>e
MinMaxscaler	0.7677	0.7699	0.7707	0.7661	e>l2>l1>n
Robust scaler	0.7745	0.7691	0.7733	0.7661	l1>e>l2>n
Quantile	0.7674	0.7736	0.7710	0.7612	l2>e>l1>n
QuantileTransformer + Chi Sampler	0.7728	0.7744	0.7757	0.7437	e>l2>l1>n

4. These models have hyperparameters. When training, experiment using GridSearch to select hyperparameters for your models. What are the best hyperparameters among those you tried?

Answer: Below table lists the best hyperparameters I could get for the four models:

Model	**Hyperparameter**
KNeighborsClassifier	KNeighborsClassifier(algorithm='auto', leaf_size=60, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=15, p=3, weights='uniform')
RandomForestClassifier	RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=120, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
LogisticRegression	LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='warn', n_jobs=None, penalty='l1', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
MLPClassifier	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant', learning_rate_init=0.001, max_iter=300, momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)

5. Which model you experimented with gives the best F1 score on testing?

Answer: LogisticRegression with RobustScaler() feature scaling gave the best f1 score of 0.82927.

Model	Feature scaling	Hyperparameter	Best f1 score
KNeighborsClassifier	Normalizer()	KNeighborsClassifier(algorithm='auto', leaf_size=60, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=15, p=3, weights='uniform')	0.81127
RandomForestClassifier	None	RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=120, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)	0.8078
LogisticRegression	RobustScaler()	LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='warn', n_jobs=None, penalty='l1', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)	0.82927
MLPClassifier	RobustScaler()	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant', learning_rate_init=0.001, max_iter=300, momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)	0.80769