Notes regarding techniques used in the project

Splitting Data

1. scikit-learn **shuffles data** using pseudorandom number generator **and splits** it into 75% training data and 25% testing data.
2. Shuffling the data is important so that test data contains data from all classes. scikit-learn uses capital X for data and lowercase y for labels.
3. The random state parameter in split, if integer, random state is the seed used by the random number generator and as it is a fixed number the outcome of split will be deterministic means split for Run 1 = split for Run2, we can use any number for random state parameter like 0,42, 21, etc.
4. For categorical target variable, it is recommended to use stratified random sampling. We are using stratified random splitting which splits data such that target variable is fairly split in training and testing data. Distribution of target variable in each partition will be similar to the original data.

Why Normalize/ Standardize data(Feature Scaling) before dimensionality reduction?

1. Normalization" onto [0,1] is called "feature scaling" or "unity-based normalization" . "Normalization" based on the observed mean and standard deviation (called "Student's t-statistic". "standardization" in more frequent but not universal usage) is typically what you want for PCA.
2. **Feature scaling** is a method used to **standardize the range of independent variables** or features of data. Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization.
   For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature.
   Therefore, the range of all features should be normalized so that **each feature contributes approximately proportionately to the final distance.**
3. Another reason why feature scaling is applied is that gradient descent converges much faster with feature scaling than without it.
4. Few useful links for it-
   https://en.wikipedia.org/wiki/Feature_scaling
   https://stats.stackexchange.com/questions/69157/why-do-we-need-to-normalize-data-before-principal-component-analysis-pca

Feature Extraction / Dimensionality Reduction

1. Dimensionality Reduction plays a really important role in machine learning, especially when you are working with thousands of features.
2. Principal Components Analysis are one of the top dimensionality reduction algorithm, it is not hard to understand and use it in real projects. This technique, in addition to making the work of feature manipulation easier, it still helps to improve the results of the classifier.
3. It means reducing dimensions of feature space - fewer relationships between variable sto consider - less likely to overfit our model. PCA is technique for Feature Extraction.
4. **Feature elimination** is what it sounds like: we reduce the feature space by eliminating features. As a disadvantage, though, we gain no information from those variables we've dropped.
5. **Feature extraction** Say we have ten independent variables. In feature extraction, we create ten "new" independent variables, where each "new" independent variable is a combination of

each of the ten “old” independent variables. However, we create these new independent variables in a specific way and order these new variables by how well they predict our dependent variable.

6. “Where does the dimensionality reduction come into play?” Well, we keep as many of the new independent variables as we want, but we drop the “least important ones.” Because we ordered the new variables by how well they predict our dependent variable, we know which variable is the most important and least important. But — and here’s the kicker — because these **new independent variables** are combinations of our old ones, we’re still keeping the most valuable parts of our old variables, even when we drop one or more of these “new” variables!
7. Useful Links – https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c
8. Explained variance is amount of variance explained by each of the selected component. In other words, it is the eigenvalue.
9. **Explained Variance Ratio** is Explained Variance divided by sum of eigenvalues.
10. We can set “n_components” to be a float between 0.0 to 1.0, indicating the ratio of variance we wish to preserve.
11. **Incremental PCA** is useful for large training datasets. Also, can try Randomized PCA
12. **Randomized PCA** is stochastic(randomly determined) algorithm that quickly finds an approximation of the first d principal components. It is dramatically faster than other algorithms where d is much smaller than n. But, results in this project after using randomized PCA does not change.
    I got once confused between **K-means and KNN** so though to jot down the difference
    These are completely different methods. The fact that they both have the letter K in their name is a coincidence.
    K-means is a clustering algorithm that tries to partition a set of points into K sets (clusters) such that the points in each cluster tend to be near each other. It is unsupervised because the points have no external classification.
    K-nearest neighbors is a classification (or regression) algorithm that in order to determine the classification of a point, combines the classification of the K nearest points. It is supervised because you are trying to classify a point based on the known classification of other points.

Evaluation of Models/Classifiers

1. We have implemented three models – K-nearest-neighbors, Decision Tree and Logistic Regression.
2. By accuracy, **KNN is working better** than others with 84.21% accuracy on testing data.
3. When the target value is binary, we have three metrics:
    a. Area Under Curve(of Receiver Operating Characteristics)
    b. Root Averaged Squared Error
    c. Misclassification Rate
4. Higher the AUC above 0.5, better is the model.
5. Smaller the RASE below 0.5, better is the model.
6. For misclassification rate, a common choice for threshold $0 \leq t \leq 1$
    a. The observed proportion of a particular target category in the training partition or in the entire data
    b. The uninformative value of 1 / *k* where *k* is the number of target categories
7. In our case k =2 , so threshold for misclassification rate will be 0.5. Or the observed proportion of target category is target
   0   0.455446
   1   0.544554
   so we can even keep threshold as 0.45.

Notes regarding techniques used in the project

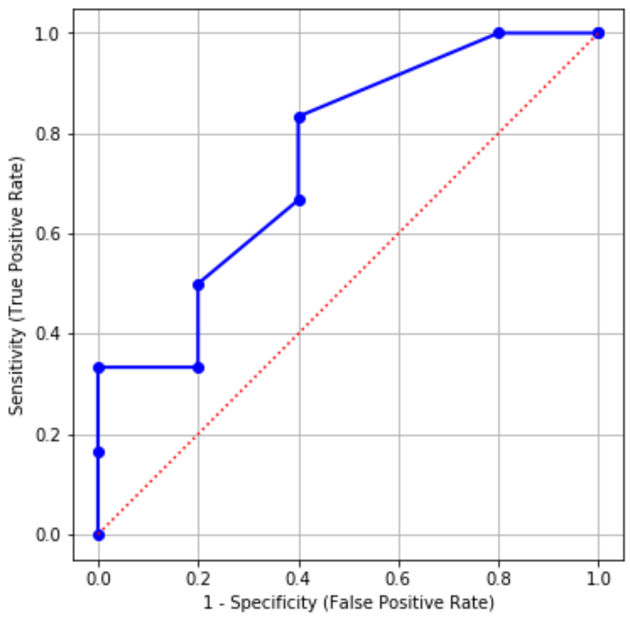
ROC Curve Example

If I can tolerate a particular percent of False Positive, then what percent of True Positive I will get?

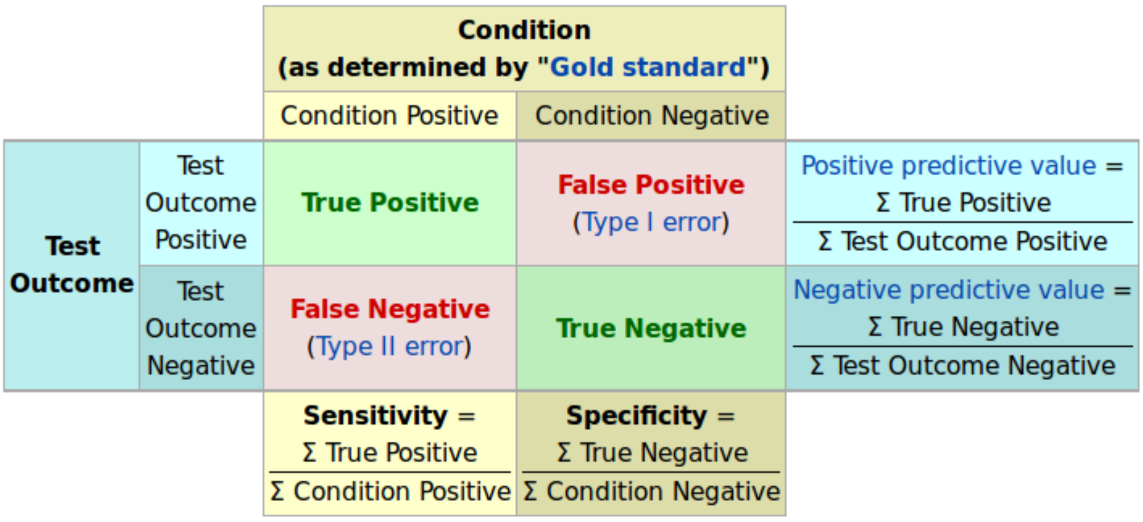
a. If I can tolerate up to 10% of False Positive, then I can get up to 33% of True Positive (Sensitivity)
b. If I can tolerate up to 20% of False Positive, then I can get up to 50% of True Positive

If I want a certain percent of True Positive, then what percent of False Positive do I need to tolerate?

a. If I want at least 80% of True Positive, then I need to tolerate at least 40% of False Positive (1 – Specificity)
b. If I want at least 40% of True Positive, then I need to tolerate at least 20% of False Positive



Sensitivity -> true positive /(true positive + false negative)
The *sensitivity* of a test is defined as the proportion of people with the disease who will have a positive result. This fact is very useful to physicians when deciding which test to use.
Confusion Matrix



		Condition (as determined by "Gold standard")		
		Condition Positive	Condition Negative	
Test Outcome	Test Outcome Positive	**True Positive**	**False Positive** (Type I error)	Positive predictive value = $\frac{\Sigma \text{ True Positive}}{\Sigma \text{ Test Outcome Positive}}$
	Test Outcome Negative	**False Negative** (Type II error)	**True Negative**	Negative predictive value = $\frac{\Sigma \text{ True Negative}}{\Sigma \text{ Test Outcome Negative}}$
		Sensitivity = $\frac{\Sigma \text{ True Positive}}{\Sigma \text{ Condition Positive}}$	**Specificity** = $\frac{\Sigma \text{ True Negative}}{\Sigma \text{ Condition Negative}}$	