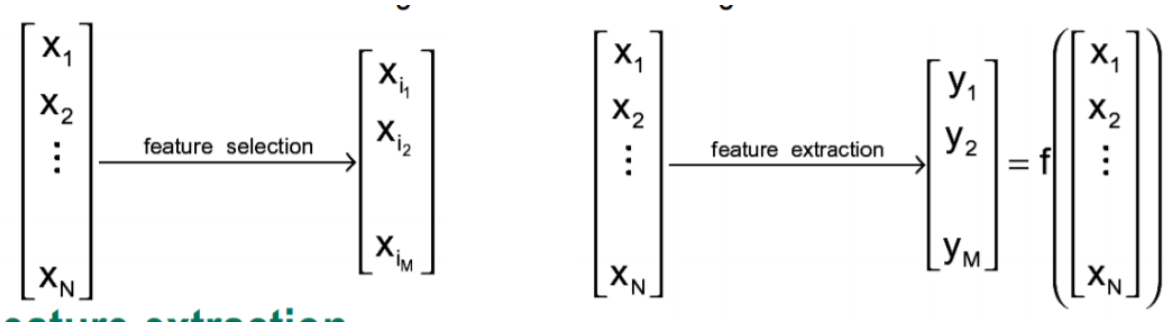
Assignment 3

1. **What are feature selection and feature extraction? Please compare and contrast this and then provide suitable example for the use of algorithms regarding these.**

This two are two general approaches for dimensionality reduction. Feature selection is selecting subset of the original feature set while feature extraction is building a new set of features from the original feature set.

Feature Selection aims to find most useful features which explained the dataset briefly, so these selected features can be input for the selected ML algorithm. PCA and ICA are feature selection methods. When we mention about PCA, It is motivated by reducing features dimensionality with small amount of information loss. It aims to maximize the variance (because max variance means max information) of the projected data on the certain dimension while minimize the mean squared distance(error) between the data and its projections.



Example of feature selection; extraction of contours in images.

Source:

http://vision.psych.umn.edu/users/schrater/schrater_lab/courses/PattRecog09/Lec17PattRec09.pdf

2. **What is feature scaling, what does it do? What is the importance of feature scaling in data analysis (especially for the application of machine learning methods)? Please provide examples for different types of feature scaling methods.**

   Feature scaling (aka data normalization) is a method which used to standardize the range of features of data. It is equalizing some features due to same scale. It is important because one feature cannot dominate to others after scaling when we apply an algorithm. Its advantage is that scaling gives a reliable number for output. As a disadvantage, if dataset has outliers, they corrupt scaling performance.
   For example; when we have weight and height as features, we should scale data. Because their respective scales are different.

   SVM, K-Means, K-nearest neighbors, logistic regression and PCA need scaled dataset before applying while Linear regression and Decision tree do not need scaling because they don't use a trade-off value.

   Source:
   http://sebastianraschka.com/Articles/2014_about_feature_scaling.html
   https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e

   BDA502_2_WEEK-7 GENERAL REVIEW.pdf slides.

3. **What is the crucial point about the necessity for the use of feature scaling methods? In other words, do we need to use feature scaling on the whole dataset or on some of the features?**

   We should use feature scaling on the whole dataset if we do not know which features needs rescaling. Otherwise, we can scale some features which needs to scale.

4. **What are the algorithms that we are suggested using scaling as well as not suggested? What is meant by information lost due to feature scaling?**

   SVM, K-Means, K-nearest neighbors, logistic regression and PCA need scaled dataset before applying while Linear regression and Decision tree do not need scaling. Decision trees use vertical and horizontal lines so there is no trade-off. In linear regression, the coefficient and the feature always together, it is not affected by rescaling. SVM and requires making trade-offs in dimensions. Algorithms in which two dimensions affect the outcome will be affected by rescaling.

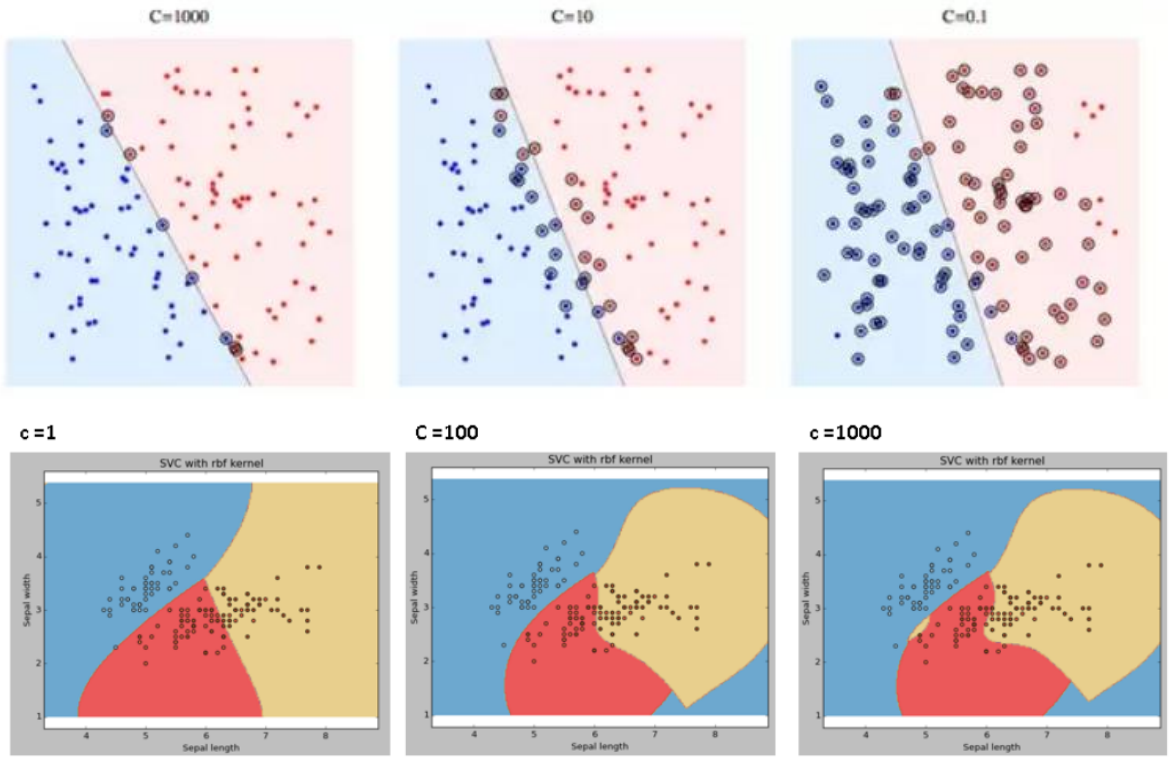
5. **What is cross-validation? What does it do on a given dataset? Why is it relevant/advised to use cross-validation?**
   - Cross validation: train and test the model with different part of dataset.
   - Use for avoiding bias and overfitting.

   1. **Holdout method:** split the dataset as train and test part. Hold a part of the training data and use the rest of data to predict. After, calculate the accuracy.
   2. **K-fold Cross Validation:** Removing some part of data on training part can cause underfitting. It may also cause losing important patterns and bias. So, K-Fold cross validation is the solution to avoid these risks. In K-fold CV method, the data is divided into "k" subsets and holdout method is repeated k times. For each time, one of the k subsets is used as test/validation set and rest k-1 subsets are used as train dataset. Error estimation is average error of all errors of k trials. As a result of this k-time repeated method, every data point involves in validation/test dataset once and k-1 times in training dataset.

Source:
https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f

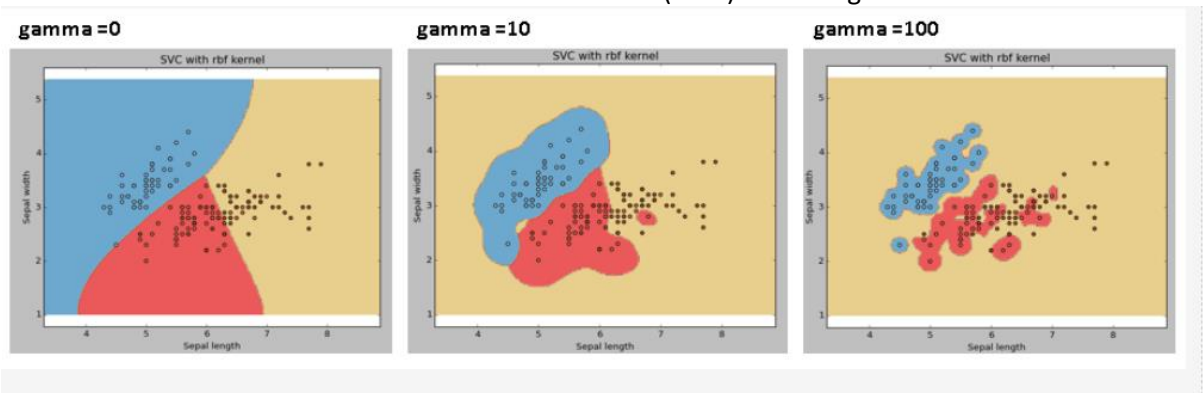
6. **What is the influence of parameters, gamma and C, related to the use of Support Vector Machine (SVM) method. Suppose that you have 2 classes that are separated with a linear hyperline/kernel.**

-C: controls trade-off between smooth decision boundary and classifying training points correctly. It is the parameter controls the influence of each individual support vector. The most important parameter when kernel=linear. Low C value causes a smooth decision surface because it is low level of penalty for misclassification.



-Gamma: defines how far the influence of a single training example reaches. No effect when kernel=linear but it is important when kernel=rbf -aka Gaussian-

We can use GridSearchCV to find the best combination (tune) of C and gamma.



Source:

https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

7. **Please work on the iris dataset with k-means algorithm (that was part of labwork activity of WEEK-5). How do you decide on the appropriate number of clusters? Please explain it in relation to the appropriate metrics.**

Code chunk:

```
print('init\t\ttime\tinertia\thomo\tcompl\tv-meas\tARI\t\tAMI\t\tsilhouette')
print(82 * '_')
#bench_k_means(KMeans(init='k-means++', n_clusters=2, n_init=10), name="k-means++",
data=iris.data) # scaled data
#bench_k_means(KMeans(init='random', n_clusters=2, n_init=10), name="random",
data=iris.data)
bench_k_means(KMeans(init='k-means++', n_clusters=no_spec, n_init=10), name="k-
means++", data=iris.data) # scaled data
bench_k_means(KMeans(init='random', n_clusters=no_spec, n_init=10), name="random",
data=iris.data)
bench_k_means(KMeans(init='k-means++', n_clusters=4, n_init=10), name="k-means++",
data=iris.data)
bench_k_means(KMeans(init='random', n_clusters=4, n_init=10), name="random",
data=iris.data)
bench_k_means(KMeans(init='k-means++', n_clusters=5, n_init=10), name="k-means++",
data=iris.data)
bench_k_means(KMeans(init='random', n_clusters=5, n_init=10), name="random",
data=iris.data)
bench_k_means(KMeans(init='k-means++', n_clusters=8, n_init=10), name="k-means++",
data=iris.data)
bench_k_means(KMeans(init='random', n_clusters=8, n_init=10), name="random",
data=iris.data)
print(82 * '_')
```

I tried with 3, 4, 5 and 8 clusters and the output is:

init	time	inertia	homo	compl	v-meas	ARI	AMI	silhouette
k-means++	0.03s	78	0.751	0.765	0.758	0.730	0.748	0.553
random	0.02s	78	0.751	0.765	0.758	0.730	0.748	0.553
k-means++	0.03s	57	0.808	0.652	0.722	0.650	0.647	0.498
random	0.02s	57	0.805	0.651	0.720	0.646	0.646	0.497
k-means++	0.02s	46	0.824	0.599	0.694	0.608	0.592	0.489
random	0.02s	46	0.828	0.601	0.696	0.612	0.593	0.491
k-means++	0.03s	30	0.926	0.508	0.656	0.453	0.495	0.353
random	0.02s	30	0.928	0.498	0.648	0.429	0.486	0.341

> V-measure: average of *homogeneity* and *completeness,* both of which are desired.
> Homogeneity: for a given *cluster*, do all the points in it come from the same class?**[0,1]**
> Completeness: for a given *class*, are all its points placed in one cluster? **[0,1]**
> Silhouette score can be used to examine the separation distance between the resulting clusters. **[-1-1].**

-1: indicate wrong cluster.
0: Overlapping.
1: the best.

> **adjusted_rand_score(ARI):** Similarity score between -1.0 and 1.0. Random labeling causes an ARI close to 0.0. 1.0 → perfect match. **[-1,1]**
> **adjusted_mutual_info_score(AMI):** Adjusted Mutual Information between two clusters. The AMI returns a value of 1 when the two partitions are perfectly matched.

> **inertia:** Sum of squared distances of samples to their closest cluster center.

When we look at the table, I select n_cluster=3. Firstly, when we look at homogeneity score, it is always increasing with number of clusters but completeness score is decreasing. When we consider V-measure score which is harmonic average of homogeneity and completeness, the best ratio is 0.722 with 3 clusters. When we consider silhouette score which represent the separation distance between clusters, again, n_clusters=3 has the best score=0.497. Even n_clusters=4 has similar values on V-score and silhouette score with n_clusters=3, it has smaller ARI and AMI score. So, the best choice is 3 clusters for this dataset.

When I tried with scaled data, silhouette score decreased, why? Because silhouette score is calculating due to Z-Score. So, our scaled data has not normal distribution.

init	time	inertia	homo	compl	v-meas	ARI	AMI	silhouette
k-means++	0.02s	140	0.652	0.653	0.653	0.610	0.648	0.458
random	0.02s	140	0.659	0.660	0.659	0.620	0.655	0.459
k-means++	0.02s	114	0.659	0.546	0.597	0.472	0.538	0.389
random	0.03s	114	0.652	0.540	0.591	0.461	0.532	0.389
k-means++	0.03s	63	0.826	0.451	0.584	0.377	0.437	0.340
random	0.02s	63	0.746	0.401	0.522	0.307	0.386	0.326

Source:
https://plot.ly/scikit-learn/plot-kmeans-silhouette-analysis/

http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

8. **Please calculate the distances between the centroids (for k = 3) for an implementation you have run in the previous question. What do these centroids show?**

```
kmeans = KMeans(init='k-means++' ,n_clusters=no_spec, n_init=10).fit(iris.data)
print(kmeans.cluster_centers_)
```

Output:

```
[[6.85       3.07368421 5.74210526 2.07105263]
 [5.006      3.418      1.464      0.244     ]
 [5.9016129  2.7483871  4.39354839 1.43387097]]
```

Euclidean distance calculator:

1 To 2 : 5.013715
2 To 3 : 3.355442
1 To 3: 1.795086

9. **For the piece of code that I have provided below, please explain what this code does in relation to the use of clustering method:**

1. K-Means Parameters:

```
#Cluster using k-means
from sklearn.cluster import KMeans
kmns = KMeans(n_clusters=2, init='k-means++', n_init=10, max_iter=
300, tol=0.0001, precompute_distances='auto', verbose=0, random_st
ate=None, copy_x=True, n_jobs=1, algorithm='auto')
kY = kmns.fit_predict(X)
```

- **n_clusters:** number of cluster to form centroids. (default=8 but in this case set 2)

- **init:** method for initialization for K-Means algo's "assign centroid" part. It can be random or "k-means++" . "k-means++" is more smart way than random.

- **n_init**: number of time the algo will be run with different centroid seeds. Final result will be the best output. (default=10)

- **max_iter:** Maximum iteration of K-Means for a single execution.(assign and optimize process for max_iter times) (default=300)

- **tol:** relative tolerance to inertia. (default=0.0001)

- precompute_distances: compute distance before clustering. "auto" do not precompute distances if n_samples*n_clusters > 12M. Faster but it takes memory.

- **verbose:** Verbosity mode.(default=0)

- random_state: the seed for random generator. If None, used by np.random.

-**copy_x:** If true, the orginal data is not changed. (default=True)

-n_jobs: about parallel computing. If 1, all CPUs are used.

-algorithm: K-Means algorithm to use; "auto" means "elkan"

2. fit_predict(X,[y]):

Compute K-Means algorithm and generate cluster centers. After, predict cluster index for each sample data point.

3. Graph:

```
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
ax1.scatter(Y[:,0],Y[:,1],  c=kY, cmap = "jet", edgecolor = "None"
, alpha=0.35)
ax1.set_title('k-means clustering plot')
ax2.scatter(Y[:,0],Y[:,1],  c = datas['diagnosis'], cmap = "jet",
edgecolor = "None", alpha=0.35)
ax2.set_title('Actual clusters')
```

This part of code chunk provides graph after clustering. It shows real clusters versus predicted clusters.