dsc-project

November 30, 2016

```
In [1]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt

        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import cross_val_score
        from sklearn.model_selection import StratifiedKFold
        from sklearn.metrics.cluster import v_measure_score
        from sklearn.metrics import classification_report
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import f1_score
        from time import time

        from sklearn.model_selection import GridSearchCV
        from sklearn.model_selection import StratifiedShuffleSplit

        import warnings
        warnings.filterwarnings('ignore')
```

```
In [2]: from sklearn.utils import resample
        from sklearn import preprocessing

        X = np.loadtxt('../data.txt')
        Y = np.loadtxt('../label.txt').astype(int)

        X = np.nan_to_num(X)

        Xp, Yp = resample(X, Y, n_samples = 5000)


        Xp_scaled = preprocessing.scale(Xp)
```

```
In [3]: def trial(X, Y, method, name, scoring='f1'):
            print (name)
            start_time = time()
            method.fit(X, Y)
            y_pred = method.predict(X)
```

```
    print(classification_report(Y, y_pred))
    print ("\tDone in %.2f s" % (time() - start_time))
    print ("Cross validation...")
    start_time = time()
    skf = StratifiedKFold(n_splits=10, shuffle=True)
    scores = cross_val_score(method, X, Y, cv=skf, scoring=scoring, n_jobs=
    accu = cross_val_score(method, X, Y, cv=skf, n_jobs=-1)
    print ("\tAccuracy: %0.2f (+/- %0.2f)" % (accu.mean(), accu.std() * 2))
    print ("\tF1 score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() *
    print ("\tDone in %.2f s" % (time() - start_time))
```

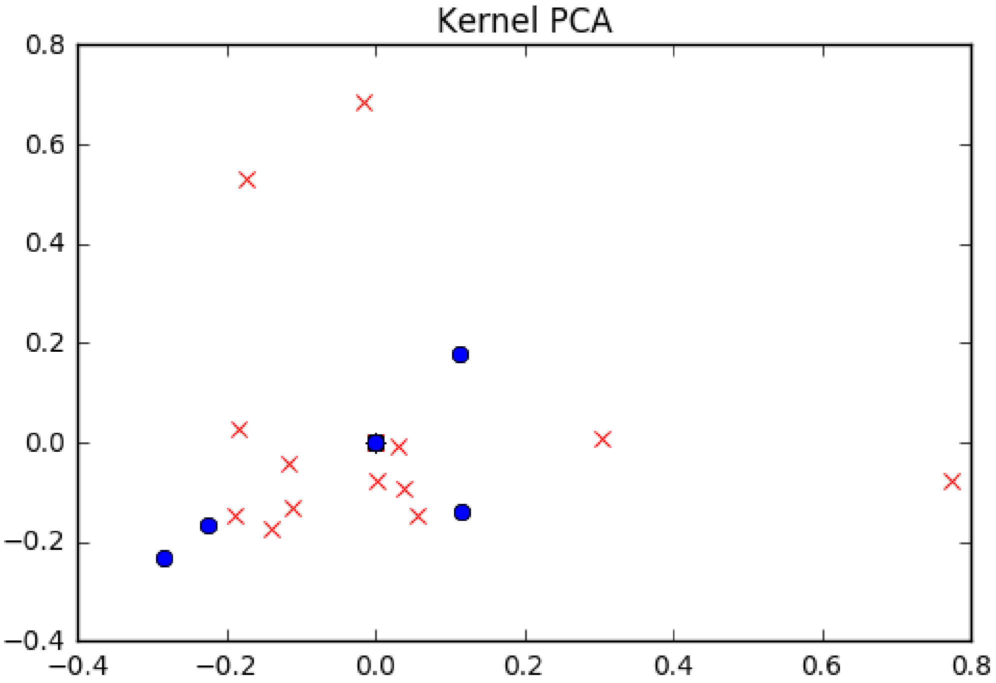
0.1 Visualization

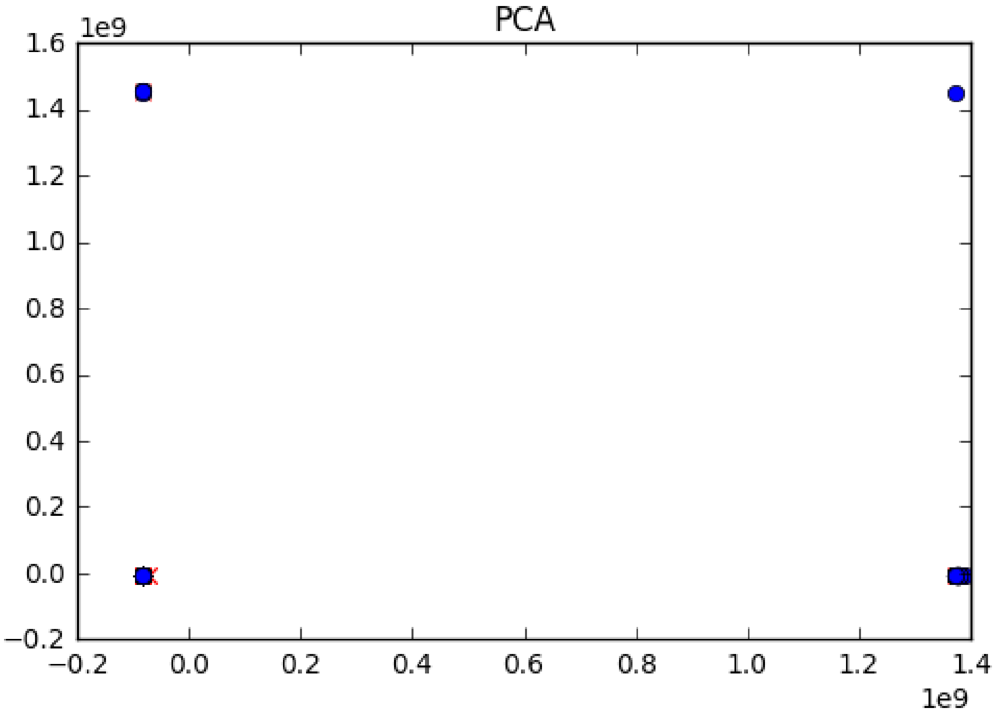
0.1.1 PCA and Kernel PCA

```
In [4]: from sklearn.decomposition import PCA, KernelPCA

        kpca = KernelPCA(n_components = 2, kernel="rbf")
        X_kpca = kpca.fit_transform(Xp)
        pca = PCA(n_components=2)
        X_pca = pca.fit_transform(Xp)
        reds = Yp == 0
        blues = Yp == 1

        plt.plot(X_kpca[reds, 0], X_kpca[reds, 1], "rx")
        plt.plot(X_kpca[blues, 0], X_kpca[blues, 1], "bo")
        plt.title('Kernel PCA')
        plt.show()
        plt.plot(X_pca[reds, 0], X_pca[reds, 1], "rx")
        plt.plot(X_pca[blues, 0], X_pca[blues, 1], "bo")
        plt.title('PCA')
        plt.show()
```


Kernel PCA
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8


PCA
1e9
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e9

0.2 random guess

```
In [5]: y_pred = np.random.randint(0, 2, Y.shape[0])
        scores = np.array([f1_score(Y, np.random.randint(0, 2, Y.shape[0])) for i i
        accu = np.array([accuracy_score(Y, np.random.randint(0, 2, Y.shape[0])) for
        print ("Accuracy: %0.2f (+/- %0.2f)" % (accu.mean(), accu.std() * 2))
        print ("F1 score: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.50 (+/- 0.00)
F1 score: 0.40 (+/- 0.00)
```

0.3 Linear methods

```
In [6]: from sklearn.linear_model import SGDClassifier, Perceptron
        from sklearn.linear_model import PassiveAggressiveClassifier
        from sklearn.linear_model import LogisticRegression
```

```
In [7]: trial(X, Y, Perceptron(), 'Perceptron')
```

```
Perceptron
             precision    recall  f1-score   support

          0       0.71      0.99      0.83    324701
          1       0.93      0.17      0.28    161347

avg / total       0.78      0.72      0.64    486048

        Done in 0.99 s
Cross validation...
        Accuracy: 0.60 (+/- 0.35)
        F1 score: 0.37 (+/- 0.21)
        Done in 15.18 s
```

```
In [8]: trial(X, Y, LogisticRegression(), 'Log Reg')
```

```
Log Reg
             precision    recall  f1-score   support

          0       0.71      0.99      0.83    324701
          1       0.93      0.17      0.28    161347

avg / total       0.78      0.72      0.64    486048

        Done in 2.40 s
Cross validation...
        Accuracy: 0.72 (+/- 0.00)
        F1 score: 0.28 (+/- 0.01)
        Done in 68.48 s
```

```
In [9]: trial(X, Y, SGDClassifier(), 'SGD Clasifier')

SGD Clasifier
             precision    recall  f1-score   support

          0       0.58      0.00      0.00    324701
          1       0.33      1.00      0.50    161347

avg / total       0.50      0.33      0.17    486048

        Done in 1.48 s
Cross validation...
        Accuracy: 0.68 (+/- 0.23)
        F1 score: 0.30 (+/- 0.25)
        Done in 15.49 s
```

```
In [10]: trial(X, Y, SGDClassifier(loss="log", penalty="l2"), 'SGD with log')

SGD with log
             precision    recall  f1-score   support

          0       0.00      0.00      0.00    324701
          1       0.33      1.00      0.50    161347

avg / total       0.11      0.33      0.17    486048

        Done in 1.05 s
Cross validation...
        Accuracy: 0.60 (+/- 0.35)
        F1 score: 0.39 (+/- 0.22)
        Done in 15.78 s
```

```
In [11]: pac = PassiveAggressiveClassifier(random_state=9,
                                           class_weight='balanced',
                                           n_jobs=-1,
                                           n_iter=9)
         trial(X, Y, pac, 'Passive Aggresive Clasifier')

Passive Aggresive Clasifier
             precision    recall  f1-score   support

          0       0.71      0.99      0.83    324701
          1       0.93      0.17      0.28    161347

avg / total       0.78      0.72      0.64    486048

        Done in 1.87 s
```

```
Cross validation...
        Accuracy: 0.56 (+/- 0.38)
        F1 score: 0.37 (+/- 0.21)
        Done in 23.27 s
```

0.4 Non - Linear models

```
In [12]: from sklearn import svm
         import warnings
         warnings.filterwarnings('ignore')
```

```
In [13]: # Subsample
         trial(Xp, Yp, svm.SVC(), 'SVC')
```

```
SVC
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      3343
          1       1.00      1.00      1.00      1657

avg / total       1.00      1.00      1.00      5000

        Done in 3.95 s
Cross validation...
        Accuracy: 0.67 (+/- 0.01)
        F1 score: 0.02 (+/- 0.02)
        Done in 28.05 s
```

```
In [14]: # Subsample
         trial(Xp, Yp, svm.NuSVC(gamma=1e9), 'Nu SVC')
```

```
Nu SVC
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      3343
          1       1.00      1.00      1.00      1657

avg / total       1.00      1.00      1.00      5000

        Done in 3.78 s
Cross validation...
        Accuracy: 0.67 (+/- 0.00)
        F1 score: 0.01 (+/- 0.02)
        Done in 29.39 s
```

0.5 Non linear Transformations

```
In [15]: from sklearn.ensemble import RandomTreesEmbedding, ExtraTreesClassifier

         # use RandomTreesEmbedding to transform data
         hasher = RandomTreesEmbedding(n_jobs=-1)
         X_randomTrees = hasher.fit_transform(X)
```

```
In [16]: trial(X_randomTrees, Y, LogisticRegression(), 'Random Trees Embedding')
```

```
Log Reg
             precision    recall  f1-score   support

          0       0.71      0.99      0.83    324701
          1       0.92      0.17      0.28    161347

avg / total       0.78      0.72      0.65    486048

        Done in 9.45 s
Cross validation...
        Accuracy: 0.72 (+/- 0.00)
        F1 score: 0.28 (+/- 0.01)
        Done in 141.27 s
```

```
In [17]: trees = ExtraTreesClassifier()
         trial(Xp, Yp, trees, 'trees')
```

```
trees
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      3343
          1       1.00      1.00      1.00      1657

avg / total       1.00      1.00      1.00      5000

        Done in 0.10 s
Cross validation...
        Accuracy: 0.70 (+/- 0.03)
        F1 score: 0.41 (+/- 0.06)
        Done in 1.25 s
```

```
In [18]: from sklearn.naive_bayes import BernoulliNB
         nb = BernoulliNB()
         trial(X_randomTrees, Y, nb, 'Naive bayes')
         trial(X, Y, nb, 'Naive bayes')
```

```
Naive bayes
             precision    recall  f1-score   support
```

```
          0       0.71      0.96      0.82    324701
          1       0.73      0.19      0.31    161347

avg / total       0.72      0.71      0.65    486048

        Done in 0.35 s
Cross validation...
        Accuracy: 0.71 (+/- 0.00)
        F1 score: 0.31 (+/- 0.01)
        Done in 4.56 s
Naive bayes
             precision    recall  f1-score   support

          0       0.72      0.86      0.78    324701
          1       0.54      0.33      0.41    161347

avg / total       0.66      0.68      0.66    486048

        Done in 1.18 s
Cross validation...
        Accuracy: 0.68 (+/- 0.00)
        F1 score: 0.41 (+/- 0.01)
        Done in 41.89 s
```

```
In [19]: from sklearn.kernel_approximation import RBFSampler
```

```
In [20]: rbf_feature = RBFSampler(gamma=1, random_state=1)
         X_rbf = rbf_feature.fit_transform(Xp)
         trial(X_rbf, Yp, trees, 'RBF transformation')
```

```
RBF transformation
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      3343
          1       1.00      1.00      1.00      1657

avg / total       1.00      1.00      1.00      5000

        Done in 0.29 s
Cross validation...
        Accuracy: 0.64 (+/- 0.03)
        F1 score: 0.16 (+/- 0.08)
        Done in 1.54 s
```

0.6 Manifold learning

```
In [21]: from sklearn import neighbors
         nnc = neighbors.KNeighborsClassifier(n_neighbors = 1,
                                              weights='uniform',
                                              algorithm='kd_tree')
         trial(Xp, Yp, nnc, 'K Neighbors')
```

```
Nearest neigbors
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      3343
          1       1.00      1.00      1.00      1657

avg / total       1.00      1.00      1.00      5000

        Done in 0.83 s
Cross validation...
        Accuracy: 0.61 (+/- 0.04)
        F1 score: 0.41 (+/- 0.05)
        Done in 1.58 s
```

```
In [22]: nnc = neighbors.KNeighborsClassifier(n_neighbors = 4,
                                              weights='distance',
                                              algorithm='auto')
         trial(Xp, Yp, nnc, 'Nearest neigbors')
```

```
Nearest neigbors
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      3343
          1       1.00      1.00      1.00      1657

avg / total       1.00      1.00      1.00      5000

        Done in 0.69 s
Cross validation...
        Accuracy: 0.63 (+/- 0.03)
        F1 score: 0.39 (+/- 0.04)
        Done in 1.44 s
```

```
In [23]: from sklearn.neighbors import RadiusNeighborsClassifier

         rnc = RadiusNeighborsClassifier(radius=100)
         trial(Xp_scaled, Yp, rnc, 'Radius neighbors classiflier')
```

```
Radius neigbors classiflier
             precision    recall  f1-score   support
```

```
          0       0.67      1.00      0.80      3343
          1       0.00      0.00      0.00      1657

avg / total       0.45      0.67      0.54      5000

        Done in 15.36 s
Cross validation...
        Accuracy: 0.67 (+/- 0.00)
        F1 score: 0.00 (+/- 0.00)
        Done in 12.97 s
```

0.7 Multi layer peceptron

```
In [24]: from sklearn.neural_network import MLPClassifier

         mlp = MLPClassifier(solver='lbfgs', alpha=1e-5, activation='relu',
                             hidden_layer_sizes=(100,33), random_state=10, tol=1e-9
                             max_iter=400)

         trial(Xp, Yp, mlp, 'Multi layer perceptron')
```

```
Multi layer perceptron
             precision    recall  f1-score   support

          0       0.00      0.00      0.00      3343
          1       0.33      1.00      0.50      1657

avg / total       0.11      0.33      0.16      5000

        Done in 0.80 s
Cross validation...
        Accuracy: 0.33 (+/- 0.00)
        F1 score: 0.50 (+/- 0.00)
        Done in 11.18 s
```

```
In [25]: parameters = {'solver': ['lbfgs'],
                       'alpha': 10.0 ** -np.arange(-1, 7),
                       'hidden_layer_sizes': [(100,33)],
                       'activation': ['identity', 'logistic', 'tanh', 'relu'],
                       'max_iter' : [400],
                       'tol': [1e-5, 1e-9],
                       'learning_rate': ['constant', 'invscaling', 'adaptive']
                      }
         mlp = MLPClassifier()
         # cv = StratifiedShuffleSplit(n_splits=10)
         skf = StratifiedKFold(n_splits=10, shuffle=True)
         clf = GridSearchCV(mlp, parameters, cv=skf, n_jobs=-1, scoring='f1')
         start_time = time()
         # clf.fit(Xp, Yp)
         #
         # The best parameters are {'tol': 1e-05, 'activation': 'identity', 'alpha
         # Done in 11507.70 s

         print("The best parameters are %s with a score of %0.2f"
               % (clf.best_params_, clf.best_score_))
         print ("\tDone in %.2f s" % (time() - start_time))

The best parameters are {'tol': 1e-05, 'activation': 'identity', 'alpha': 1.0, 'lea
        Done in 11507.70 s
```

```
In [26]: mlp = MLPClassifier(solver='lbfgs', alpha=1e-6,
                             hidden_layer_sizes=(100,33),
                             learning_rate='invscaling',
                             tol = 1e-5)

         trial(Xp, Yp, mlp, 'Multi layer perceptron')

Multi layer perceptron
             precision    recall  f1-score   support

          0       0.67      1.00      0.80      3343
          1       0.43      0.00      0.00      1657

avg / total       0.59      0.67      0.54      5000

        Done in 0.60 s
Cross validation...
        Accuracy: 0.67 (+/- 0.00)
        F1 score: 0.19 (+/- 0.34)
        Done in 12.21 s
```

```
In [27]: parameters = {'n_neighbors': [1, 2, 4, 10, 20],
                       'weights': ['uniform', 'distance']}

         ncc = neighbors.KNeighborsClassifier()
         # cv = StratifiedShuffleSplit(n_splits=10)
         skf = StratifiedKFold(n_splits=10, shuffle=True)
         clf = GridSearchCV(ncc, parameters, cv=skf, n_jobs=-1, scoring='f1')
         start_time = time()
         print ("start")
         clf.fit(Xp, Yp)
         print("The best parameters are %s with a score of %0.2f"
               % (clf.best_params_, clf.best_score_))
         print ("\tDone in %.2f s" % (time() - start_time))
```

```
start
The best parameters are {'weights': 'uniform', 'n_neighbors': 1} with a score of 0.
        Done in 40.58 s
```

```
In [41]: parameters = {'n_estimators': [1024],
                       'min_samples_split': [256],
                       'class_weight': ['balanced', None],
                       'max_depth': [None],
                       'max_features': [5, None, 'sqrt']
                      }

         trees = ExtraTreesClassifier()
         skf = StratifiedKFold(n_splits=10, shuffle=True)
         clf = GridSearchCV(trees, parameters, cv=skf, n_jobs=-1, scoring='f1')
         start_time = time()
         '''start
         The best parameters are {'class_weight': 'balanced_subsample', 'max_depth
         Done in 1698.70 s
         '''

         print ("start")
         clf.fit(Xp, Yp)
         print("The best parameters are %s with a score of %0.2f"
               % (clf.best_params_, clf.best_score_))
         print ("\tDone in %.2f s" % (time() - start_time))
```

```
start
The best parameters are {'class_weight': 'balanced', 'max_features': None, 'max_dep
        Done in 293.03 s
```

```
In [36]: trees = ExtraTreesClassifier(class_weight='balanced',
                                      n_estimators=1024,
                                      min_samples_split=256)

         trial(Xp, Yp, trees, 'Extra tree classifier optimized')
```

```
Extra tree classifier optimized
             precision    recall  f1-score   support

          0       0.81      0.81      0.81      3343
          1       0.62      0.62      0.62      1657

avg / total       0.75      0.75      0.75      5000
```

```
        Done in 9.27 s
Cross validation...
        Accuracy: 0.68 (+/- 0.05)
        F1 score: 0.51 (+/- 0.05)
        Done in 72.17 s
```