Project Report

December 12, 2018

Group name: VDC group | Members: Chu-Ching Ho, Di Wu, Van Ly Nguyen

1 Preparation

We first load the data into memory and split data into training data and validation data as following:

```
In [1]: import numpy as np
        np.random.seed(0)
        import mltools as ml
        import matplotlib.pyplot as plt
        import pandas as pd
        from sklearn.model_selection import train_test_split
        CSV_FILE_PATH = 'data/diabetic_data.csv'
        data = pd.read_csv(CSV_FILE_PATH)

        print(data.shape)
        print(data.info())
```

```
(101766, 50)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 50 columns):
encounter_id                101766 non-null int64
patient_nbr                 101766 non-null int64
race                        101766 non-null object
gender                      101766 non-null object
age                         101766 non-null object
weight                      101766 non-null object
admission_type_id           101766 non-null int64
discharge_disposition_id    101766 non-null int64
admission_source_id         101766 non-null int64
time_in_hospital            101766 non-null int64
payer_code                  101766 non-null object
medical_specialty           101766 non-null object
num_lab_procedures          101766 non-null int64
num_procedures              101766 non-null int64
```

```
num_medications             101766 non-null int64
number_outpatient           101766 non-null int64
number_emergency            101766 non-null int64
number_inpatient            101766 non-null int64
diag_1                      101766 non-null object
diag_2                      101766 non-null object
diag_3                      101766 non-null object
number_diagnoses            101766 non-null int64
max_glu_serum               101766 non-null object
A1Cresult                   101766 non-null object
metformin                   101766 non-null object
repaglinide                 101766 non-null object
nateglinide                 101766 non-null object
chlorpropamide              101766 non-null object
glimepiride                 101766 non-null object
acetohexamide               101766 non-null object
glipizide                   101766 non-null object
glyburide                   101766 non-null object
tolbutamide                 101766 non-null object
pioglitazone                101766 non-null object
rosiglitazone               101766 non-null object
acarbose                    101766 non-null object
miglitol                    101766 non-null object
troglitazone                101766 non-null object
tolazamide                  101766 non-null object
examide                     101766 non-null object
citoglipton                 101766 non-null object
insulin                     101766 non-null object
glyburide-metformin         101766 non-null object
glipizide-metformin         101766 non-null object
glimepiride-pioglitazone    101766 non-null object
metformin-rosiglitazone     101766 non-null object
metformin-pioglitazone      101766 non-null object
change                      101766 non-null object
diabetesMed                 101766 non-null object
readmitted                  101766 non-null object
dtypes: int64(13), object(37)
memory usage: 38.8+ MB
None
```

Because not all features are useful, we could do a feature selection before training our model. Our features include much personal information about patients such as age, weight, race and many other medical treatment results.

We can see there are 101766 data and 49 features in the original data. And Y is the class label of each specific data point. For some very obvious features (like "encounter_id", "patient_nbr"), we can say they are just idenitities and are not related to our prediction (readmitted info) so we just ignore them. For feature examide and citoglipton, they only have 1 value and cannot be used in

prediction. Further, among our data there are many missing value as mentioned in the essay. And we can just drop those features with missing rate higher than certain value (eg. 40%).

```
In [2]: for col in data.columns:
            if data[col].dtype == object:
                count = data[col][data[col] == '?'].count()
                if count != 0:
                    print(col, count)
        print('gender', data['gender'][data['gender'] == 'Unknown/Invalid'].count())
```

```
('race', 2273)
('weight', 98569)
('payer_code', 40256)
('medical_specialty', 49949)
('diag_1', 21)
('diag_2', 358)
('diag_3', 1423)
('gender', 3)
```

```
In [3]: # identities
        data = data.drop('encounter_id', axis=1)
        data = data.drop('patient_nbr', axis=1)
        # only contains one value
        data = data.drop('examide', axis=1)
        data = data.drop('citoglipton', axis=1)
        # missing rate is high
        data = data.drop('weight', axis=1)
        data = data.drop('payer_code', axis=1)
        data = data.drop('medical_specialty', axis=1)
```

We should also remove some records: for those records with diag1, diag2 and diag3 all missing, we should get rid of them. And we also remove records with gender = 'Unknown/Invalid'.

```
In [4]: diag_Idx = set(data[(data['diag_1'] == '?') & (data['diag_2'] == '?') & (data['diag_3']
        gender_Idx = set(data['gender'][data['gender'] == 'Unknown/Invalid'].index)
        data = data.drop(data.index[list(diag_Idx.union(gender_Idx))])
        print(data.shape)
```

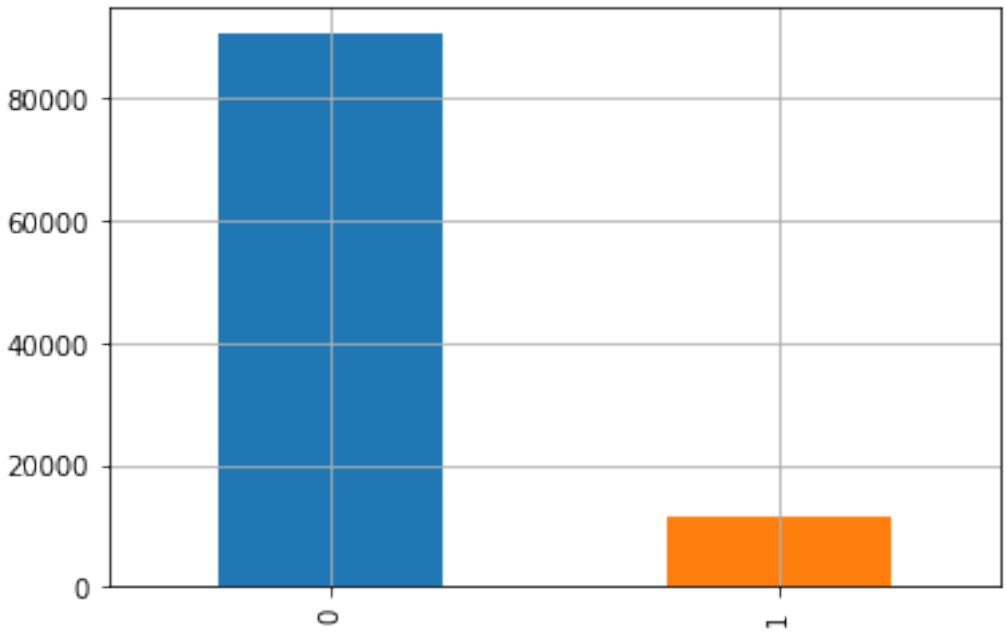
```
(101762, 43)
```

For the remaining data missing, we should replace them with np.nan.

```
In [5]: data = data.replace('?',np.nan)
```

We are predicting whether a patient will occur readmission after treatment. And for patients with >30 days' readmitted, we replace them with 0. For readmitted in more than 30 days has very little correlated factor with first admitted. The distribution of our classes is like the following:

```
In [6]: data['readmitted'] = data['readmitted'].replace('>30', 0)
        data['readmitted'] = data['readmitted'].replace('<30', 1)
        data['readmitted'] = data['readmitted'].replace('NO', 0)
        data['readmitted'].value_counts().plot(kind='bar', grid=True)
        plt.figure(figsize=(5,3))
        plt.show()
        print(data.groupby('readmitted').size())
        print(data.head().T)
```




```
<Figure size 360x216 with 0 Axes>


readmitted
0    90405
1    11357
dtype: int64
                                0          1                2          3  \
race                      Caucasian  Caucasian  AfricanAmerican  Caucasian
gender                       Female     Female           Female       Male
age                          [0-10)     [10-20)          [20-30)    [30-40)
admission_type_id                 6          1                1          1
discharge_disposition_id         25          1                1          1
admission_source_id               1          7                7          7
time_in_hospital                  1          3                2          2
num_lab_procedures               41         59               11         44
num_procedures                    0          0                5          1
```

num_medications	1	18	13	16
number_outpatient	0	0	2	0
number_emergency	0	0	0	0
number_inpatient	0	0	1	0
diag_1	250.83	276	648	8
diag_2	NaN	250.01	250	250.43
diag_3	NaN	255	V27	403
number_diagnoses	1	9	6	7
max_glu_serum	None	None	None	None
A1Cresult	None	None	None	None
metformin	No	No	No	No
repaglinide	No	No	No	No
nateglinide	No	No	No	No
chlorpropamide	No	No	No	No
glimepiride	No	No	No	No
acetohexamide	No	No	No	No
glipizide	No	No	Steady	No
glyburide	No	No	No	No
tolbutamide	No	No	No	No
pioglitazone	No	No	No	No
rosiglitazone	No	No	No	No
acarbose	No	No	No	No
miglitol	No	No	No	No
troglitazone	No	No	No	No
tolazamide	No	No	No	No
insulin	No	Up	No	Up
glyburide-metformin	No	No	No	No
glipizide-metformin	No	No	No	No
glimepiride-pioglitazone	No	No	No	No
metformin-rosiglitazone	No	No	No	No
metformin-pioglitazone	No	No	No	No
change	No	Ch	No	Ch
diabetesMed	No	Yes	Yes	Yes
readmitted	0	0	0	0

	4
race	Caucasian
gender	Male
age	[40-50)
admission_type_id	1
discharge_disposition_id	1
admission_source_id	7
time_in_hospital	1
num_lab_procedures	51
num_procedures	0
num_medications	8
number_outpatient	0
number_emergency	0

```
number_inpatient                0
diag_1                        197
diag_2                        157
diag_3                        250
number_diagnoses                5
max_glu_serum                None
A1Cresult                    None
metformin                      No
repaglinide                    No
nateglinide                    No
chlorpropamide                 No
glimepiride                    No
acetohexamide                  No
glipizide                  Steady
glyburide                      No
tolbutamide                    No
pioglitazone                   No
rosiglitazone                  No
acarbose                       No
miglitol                       No
troglitazone                   No
tolazamide                     No
insulin                    Steady
glyburide-metformin            No
glipizide-metformin            No
glimepiride-pioglitazone       No
metformin-rosiglitazone        No
metformin-pioglitazone         No
change                         Ch
diabetesMed                   Yes
readmitted                      0
```

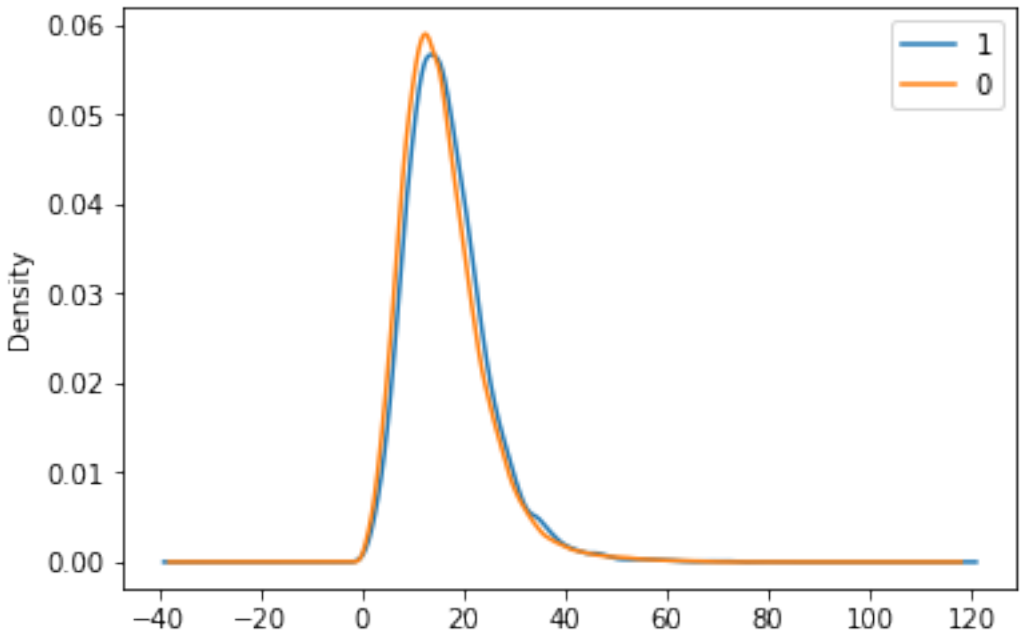
From bar chart above among 101,767 data, we could see that nearly 90,405 data point with "readmitted", 11,357 labeled "no readmitted". That means patients without readmitted are more frequent and patients with readmitted much smaller which is consistent with common sense. After knowing about our classes, we want to know more about our features which is vital to train our model and make corresponding prediction in the following.

2 Feature selection

For some numeric features, we could plot the density distribution of each feature and get rid of unuseful features in the following (classification density is same over different values).

```
In [7]: # for feature num_medications
        data['num_medications'][data['readmitted'] == 1].plot(kind='kde')
        data['num_medications'][data['readmitted'] == 0].plot(kind='kde')
```

```
plt.legend(('1', '0'),loc='best')
plt.show()
```



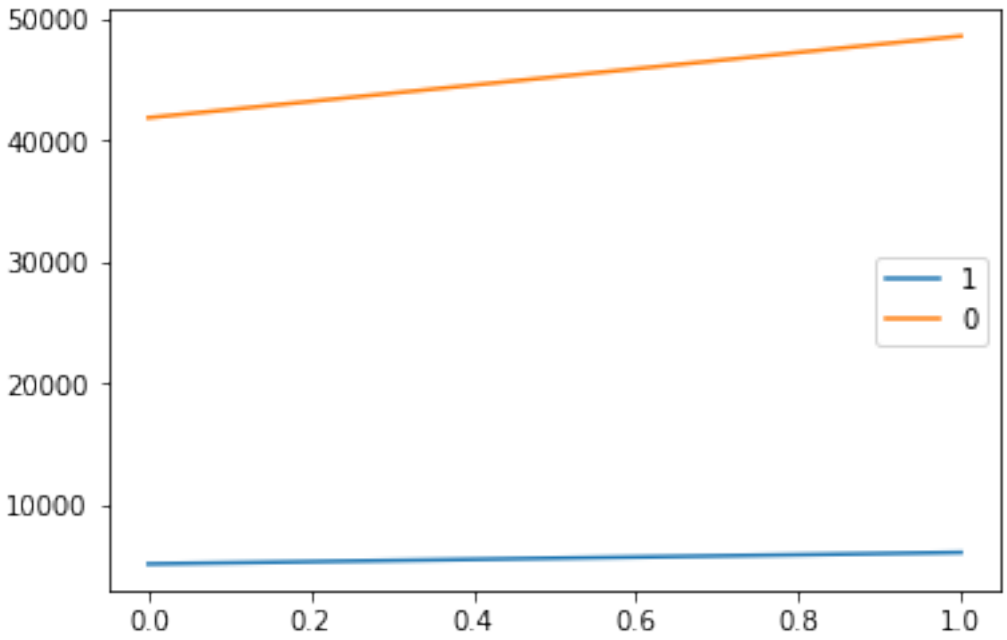
For this 'num_medications' feature, we find the density of 0, 1 and 2 almost overlapped. It means it nearly makes no contribution to class decision. We should get rid of it.

```
In [8]: data = data.drop('num_medications', axis=1)
```

After that, we should do some encoding for some nominal features as following:

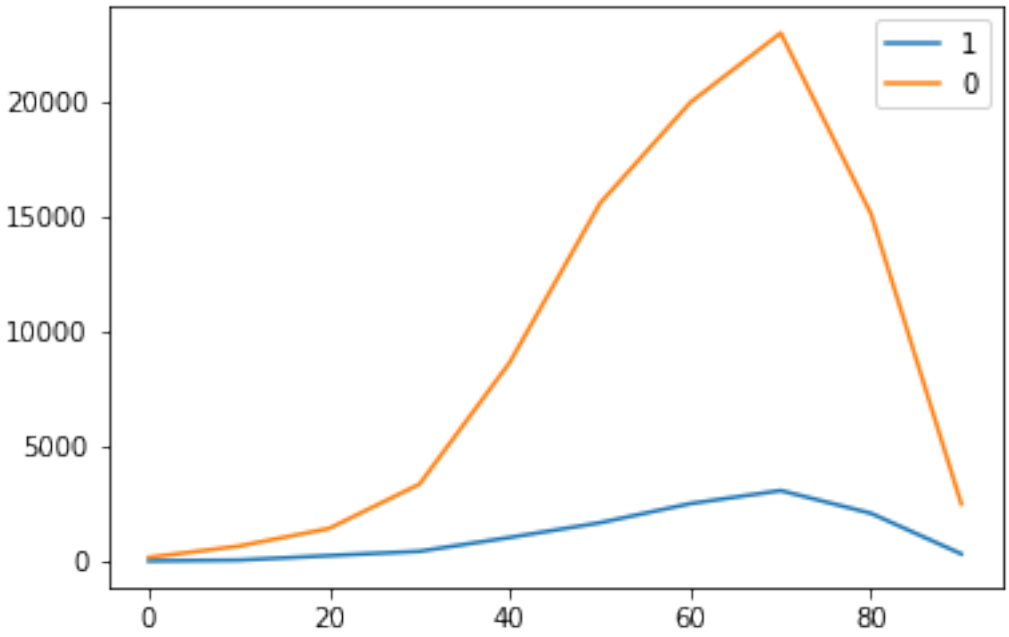
```
In [9]: from collections import OrderedDict as odict

        # For feature gender, we should code 'male' and 'female' using 0 and 1
        data['gender'] = data['gender'].replace('Male',0)
        data['gender'] = data['gender'].replace('Female',1)

        feat2 = pd.concat([data.iloc[:,1], data['readmitted']], axis=1)
        feat2_lt30 = feat2[feat2['readmitted'] == 1]
        feat2_no = feat2[feat2['readmitted'] == 0]

        feat2_1 = dict(feat2_lt30['gender'].value_counts())
        feat2_1 = odict(sorted(feat2_1.items()))

        feat2_2 = dict(feat2_no['gender'].value_counts())
        feat2_2 = odict(sorted(feat2_2.items()))

        plt.plot(list(feat2_1.keys()), list(feat2_1.values()), label="class_1")
```

```
plt.plot(list(feat2_2.keys()), list(feat2_2.values()), label="class_2")
plt.legend(('1', '0'),loc='best')
plt.show()
```



In [10]:
```
# for feature age

# for age feature with '[i - i+10)' representation,
# we should replace them with the begining number to turn this feature into numeric
data['age'] = data['age'].replace('[0-10)',0)
data['age'] = data['age'].replace('[10-20)',10)
data['age'] = data['age'].replace('[20-30)',20)
data['age'] = data['age'].replace('[30-40)',30)
data['age'] = data['age'].replace('[40-50)',40)
data['age'] = data['age'].replace('[50-60)',50)
data['age'] = data['age'].replace('[60-70)',60)
data['age'] = data['age'].replace('[70-80)',70)
data['age'] = data['age'].replace('[80-90)',80)
data['age'] = data['age'].replace('[90-100)',90)

feat4 = pd.concat([data.iloc[:,2], data['readmitted']], axis=1)
feat4_lt30 = feat4[feat4['readmitted'] == 1]
feat4_no = feat4[feat4['readmitted'] == 0]

feat4_1 = dict(feat4_lt30['age'].value_counts())
feat4_1 = odict(sorted(feat4_1.items()))
```

```
feat4_2 = dict(feat4_no['age'].value_counts())
feat4_2 = odict(sorted(feat4_2.items()))

plt.plot(list(feat4_1.keys()), list(feat4_1.values()), label="class_1")
plt.plot(list(feat4_2.keys()), list(feat4_2.values()), label="class_2")
plt.legend(('1', '0'),loc='best')
plt.show()
```




```
In [11]: # For other discrete features, we should also replace them with numeric value
         data['change'] = data['change'].replace('Ch', 1)
         data['change'] = data['change'].replace('No', 0)

         data['diabetesMed'] = data['diabetesMed'].replace('Yes', 1)
         data['diabetesMed'] = data['diabetesMed'].replace('No', 0)

         data['max_glu_serum'] = data['max_glu_serum'].replace('>300', 3)
         data['max_glu_serum'] = data['max_glu_serum'].replace('>200', 2)
         data['max_glu_serum'] = data['max_glu_serum'].replace('Normal', 1)
         data['max_glu_serum'] = data['max_glu_serum'].replace('None', 0)

         data['A1Cresult'] = data['A1Cresult'].replace('>8', 3)
         data['A1Cresult'] = data['A1Cresult'].replace('>7', 2)
         data['A1Cresult'] = data['A1Cresult'].replace('Normal', 1)
         data['A1Cresult'] = data['A1Cresult'].replace('None', 0)
```

For feature race, it is a discrete Nominal feature with values: Caucasian, Asian, African American, Hispanic, and other. We would like to transform it into a numeric feature. Here we use

one-hot encoding. In ont-hot encoding, we create 5 additional columns for every values in 'race' and each has value 1 if record has that value or 0 otherwise.

```
In [12]: def create_dummies(data,column_name):
             dummies = pd.get_dummies(data[column_name],prefix=column_name, drop_first = True)
             print(dummies.head().T)
             data = pd.concat([data,dummies],axis=1)
             data = data.drop(column_name, axis=1)
             return data

         data = create_dummies(data, 'race')
         # data = create_dummies(data, 'max_glu_serum')
         # data = create_dummies(data, 'A1Cresult')
```

```
                 0  1  2  3  4
race_Asian       0  0  0  0  0
race_Caucasian   1  1  0  1  1
race_Hispanic    0  0  0  0  0
race_Other       0  0  0  0  0
```

There are 24 kinds of drug related features, we didn't know the details of each drugs and their efficacy. They indicate whether the dosage of each drug has been changed during treatment. "up" indicates increasing during the encounter, "down" indicates decreasing, "steady" indicates no change, and "no" indicates no prescribed (which also means no change). In order to simplify our model, we only use the number of dosage change during encounter instead of using them all which means we don't distinguish changes between drugs and will combine the changes together among all 24 changes.

```
In [13]: drugs = ['metformin', 'repaglinide', 'nateglinide', 'chlorpropamide',
                  'glimepiride', 'glipizide', 'glyburide', 'pioglitazone',
                  'rosiglitazone', 'acarbose', 'miglitol', 'insulin',
                  'glyburide-metformin', 'tolazamide',
                  'metformin-pioglitazone','metformin-rosiglitazone',
                  'glimepiride-pioglitazone', 'glipizide-metformin',
                  'troglitazone', 'tolbutamide', 'acetohexamide']
         data['drug_change_num'] = 0
         for drug in drugs:
             data['drug_change_num'] += data[drug].apply(lambda x: 0 if (x == 'No' or x == 'St
             data = data.drop(drug, axis=1)

         print(data['drug_change_num'].value_counts())
```

```
0    74059
1    26272
2     1318
3      108
4        5
Name: drug_change_num, dtype: int64
```

After all data preprocessing is done, we should shuffle our data and split them into training data (80%) and testing data (20%).

```
In [14]: data = data.sample(n = len(data)).reset_index(drop = True)
         print(data.shape)
```

```
(101762, 25)
```

```
In [15]: Y = data.readmitted
         X = data.drop('readmitted', axis=1)
         print(X.shape)
         print(Y.shape)
         X = X.apply(pd.to_numeric, errors='coerce')
         Y = Y.apply(pd.to_numeric, errors='coerce')
         X.fillna(0, inplace=True)
         Y.fillna(0, inplace=True)
         X_train, X_test, y_train, y_test = train_test_split(X, Y,test_size=0.2)
```

```
(101762, 24)
(101762,)
```

3 Model build

3.1 Logistic regression

We first choose logistic regression model. We use our training data to train the model and make prediction for both training data and testing data and finally calculate ROC AUC and Accuracy for our prediction.

```
In [16]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import roc_curve, auc, accuracy_score

         lr = LogisticRegression()
         lr.fit(X_train, y_train)

         logregression_y_train_predict = lr.predict_proba(X_train)[:,1]
         logregression_y_test_predict = lr.predict_proba(X_test)[:,1]

         threshold = 0.5

         print('Training result:')
         fpr_train, tpr_train, thresholds_train = roc_curve(y_train, logregression_y_train_pred
         print('AUC=',auc(fpr_train, tpr_train))
         print('Accuracy=',accuracy_score(y_train, (logregression_y_train_predict > threshold))

         print('Testing result:')
         fpr_test, tpr_test, thresholds_test = roc_curve(y_test, logregression_y_test_predict)
```

```
        print('AUC=',auc(fpr_test, tpr_test))
        print('Accuracy=',accuracy_score(y_test, (logregression_y_test_predict > threshold)))
```

```
Training result:
('AUC=', 0.6416803958636719)
('Accuracy=', 0.8882433146212335)
Testing result:
('AUC=', 0.6381560490621228)
('Accuracy=', 0.8883211320198496)
```

It can get 0.638 score in test data

3.2 Random forest

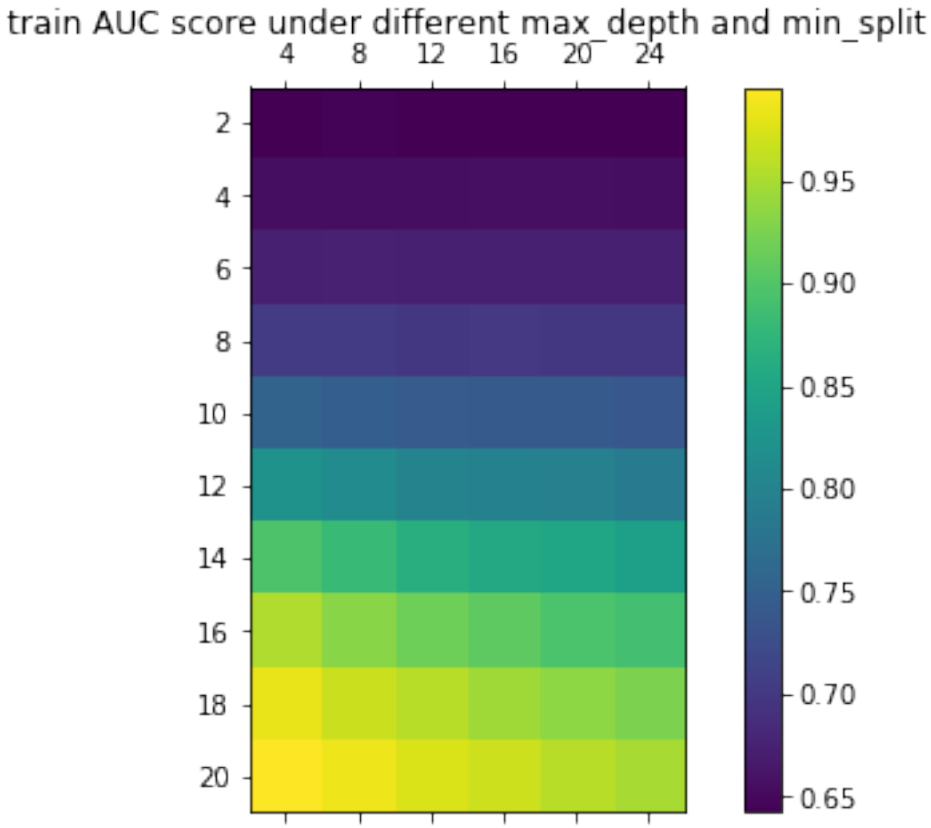
3.2.1 max_depth and min_sample_split

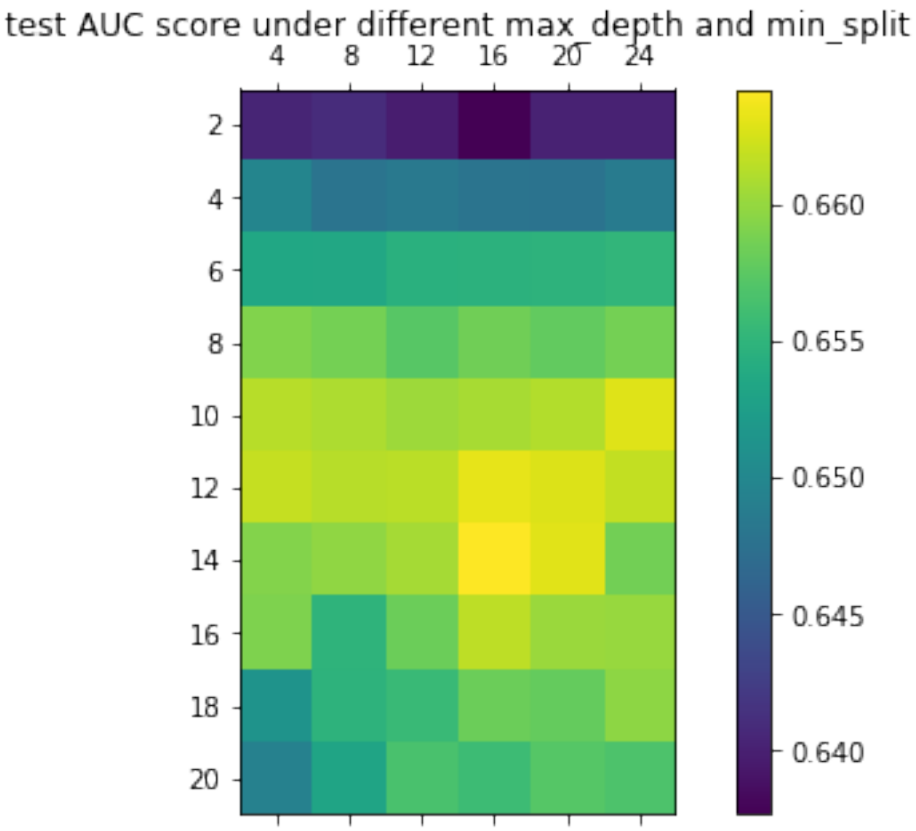
```
In [17]: from sklearn.ensemble import RandomForestClassifier

        max_depth_arr = [2,4,6,8,10,12,14,16,18,20]
        min_sample_split=[4,8,12,16,20,24]
        tr_auc = np.zeros((len(max_depth_arr),len(min_sample_split)))
        te_auc = np.zeros((len(max_depth_arr),len(min_sample_split)))
        for i,d in enumerate(max_depth_arr):
            for j,s in enumerate(min_sample_split):
                clf = RandomForestClassifier(n_estimators=100, n_jobs=4,
                                             max_depth=d, min_samples_split=s)
                clf.fit(X_train, y_train)
                randomforest_y_train_predict = clf.predict_proba(X_train)[:,1]
                randomforest_y_test_predict = clf.predict_proba(X_test)[:,1]
                fpr, tpr, thresholds = roc_curve(y_train, randomforest_y_train_predict)
                tr_auc[i][j]=auc(fpr, tpr)
                fpr, tpr, thresholds = roc_curve(y_test, randomforest_y_test_predict)
                te_auc[i][j]=auc(fpr, tpr)
        f, ax = plt.subplots(1, 1, figsize=(8, 5))
        cax = ax.matshow(tr_auc, interpolation='nearest')
        f.colorbar(cax)
        ax.set_yticklabels(list(max_depth_arr))
        ax.set_xticklabels(list(min_sample_split))
        ax.set_yticks(np.arange(len(max_depth_arr)))
        ax.set_xticks(np.arange(len(min_sample_split)))
        ax.set_title('train AUC score under different max_depth and min_split')

        f, ax = plt.subplots(1, 1, figsize=(8, 5))
        cax = ax.matshow(te_auc, interpolation='nearest')
        f.colorbar(cax)
        ax.set_yticklabels(list(max_depth_arr))
        ax.set_xticklabels(list(min_sample_split))
        ax.set_yticks(np.arange(len(max_depth_arr)))
```

```
        ax.set_xticks(np.arange(len(min_sample_split)))
        ax.set_title('test AUC score under different max_depth and min_split')
```

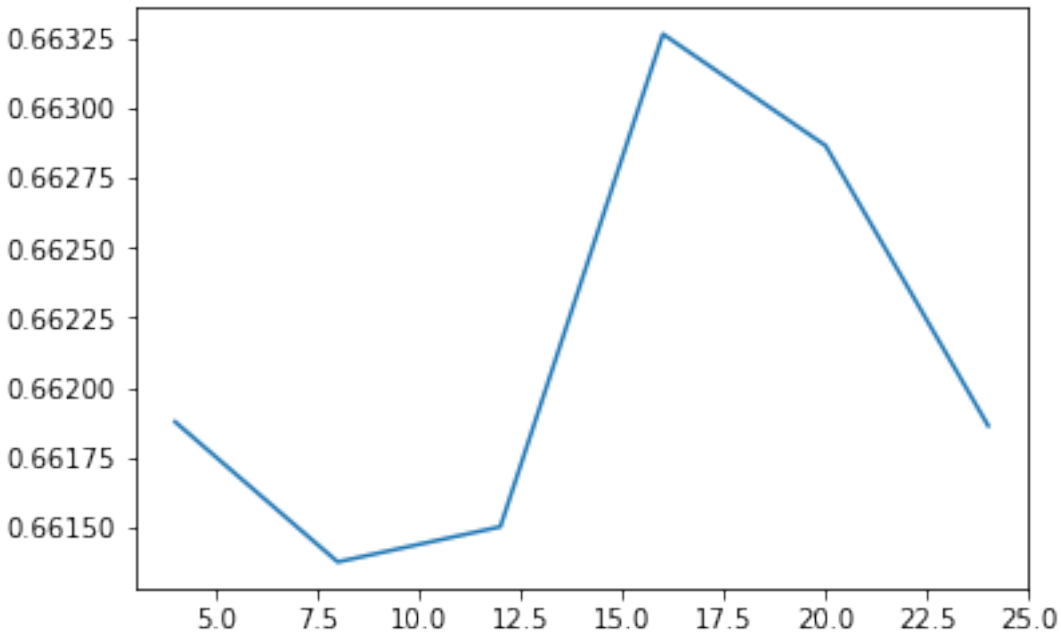
```
/anaconda2/lib/python2.7/site-packages/sklearn/ensemble/weight_boosting.py:29: DeprecationWarni
  from numpy.core.umath_tests import inner1d
```

Out[17]: Text(0.5,1.05,'test AUC score under different max_depth and min_split')





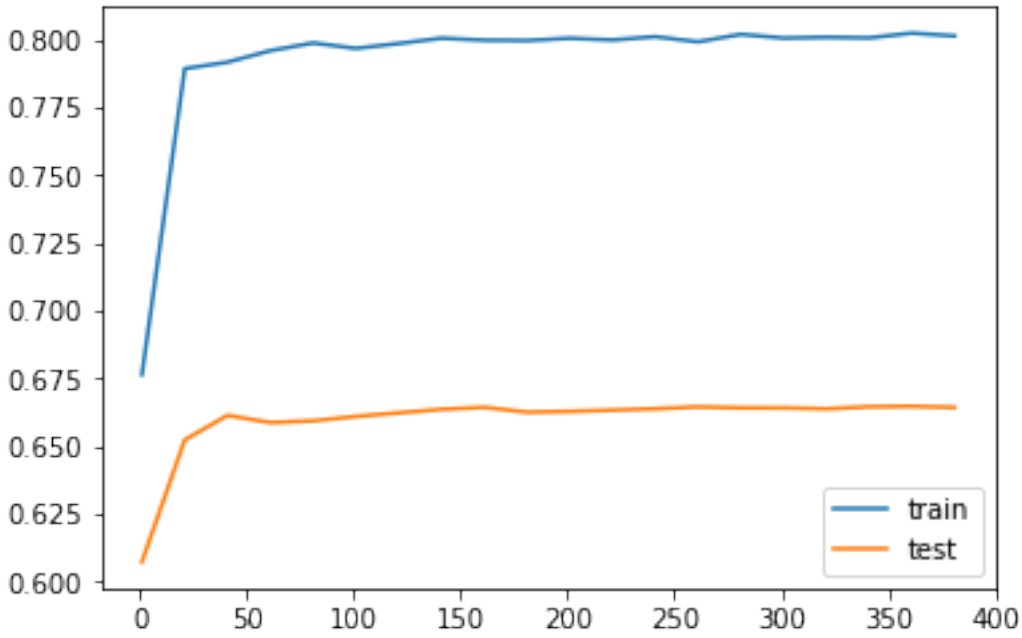
In [19]:
```
plt.plot(min_sample_split, te_auc[5,:])
plt.show()
```



From plot above, we could know min sample split produce the best result when it equals to 16 and max depth produce best result when it equals to 12

3.2.2 n_estimators

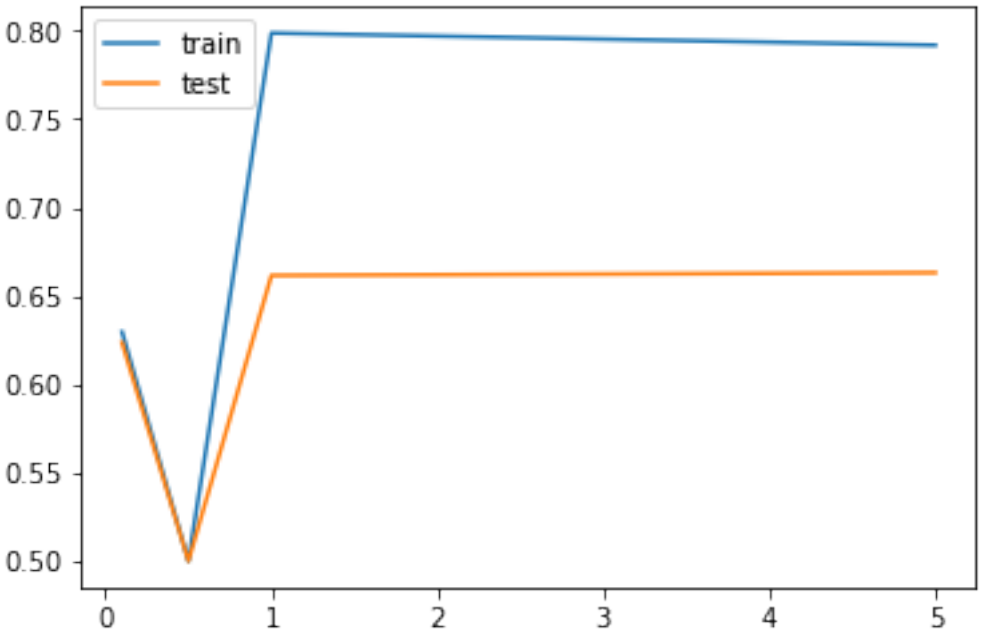
```
In [21]: train_results=np.zeros(20)
         test_results=np.zeros(20)
         x=range(0,20)
         for i in range(0,20):
             clf = RandomForestClassifier(n_estimators=20*i+1, n_jobs=4, max_depth=12,
                                          min_samples_split=16, min_samples_leaf=1)
             clf.fit(X_train, y_train)
             randomforest_y_train_predict1=clf.predict_proba(X_train)[:,1]
             randomforest_y_test_predict1=clf.predict_proba(X_test)[:,1]
             fpr, tpr, thresholds = roc_curve(y_train, randomforest_y_train_predict1)
             train_results[i]=auc(fpr, tpr)
             fpr, tpr, thresholds = roc_curve(y_test, randomforest_y_test_predict1)
             test_results[i]=auc(fpr, tpr)
             x[i] = 20*i+1
         plt.plot(x,train_results)
         plt.plot(x,test_results)
         plt.legend(['train','test'])
         plt.show()
```



When n_estimators=75 around, it will produce best result as shown in above plot. Then we can use our best attribute to train random forest model:

3.2.3 min_samples_leafs

```
In [22]: train_results=np.zeros(4)
         test_results=np.zeros(4)
         min_samples_leafs = [0.1, 0.5, 1, 5]
         for i,s in enumerate(min_samples_leafs):
             clf = RandomForestClassifier(n_estimators=75, n_jobs=4, max_depth=12,
                                          min_samples_split=16, min_samples_leaf=s)
             clf.fit(X_train, y_train)
             randomforest_y_train_predict2=clf.predict_proba(X_train)[:,1]
             randomforest_y_test_predict2=clf.predict_proba(X_test)[:,1]
             fpr, tpr, thresholds = roc_curve(y_train, randomforest_y_train_predict2)
             train_results[i]=auc(fpr, tpr)
             fpr, tpr, thresholds = roc_curve(y_test, randomforest_y_test_predict2)
             test_results[i]=auc(fpr, tpr)
         plt.plot(min_samples_leafs,train_results)
         plt.plot(min_samples_leafs,test_results)
         plt.legend(['train','test'])
         plt.show()
```



When min_samples_leaf=1, it gets its best score

```
In [23]: clf = RandomForestClassifier(n_estimators=75, n_jobs=4, max_depth=12,
                                      min_samples_split=16, min_samples_leaf=1)
```

```
        clf.fit(X_train, y_train)
        randomforest_y_train_predict3=clf.predict_proba(X_train)[:,1]
        randomforest_y_test_predict3=clf.predict_proba(X_test)[:,1]

        print('Training result:')
        fpr_train, tpr_train, thresholds_train = roc_curve(y_train, randomforest_y_train_predi
        print('AUC=',auc(fpr_train, tpr_train))
        print('Accuracy=',accuracy_score(y_train, (randomforest_y_train_predict3 > threshold))

        print('Testing result:')
        fpr_test, tpr_test, thresholds_test = roc_curve(y_test, randomforest_y_test_predict3)
        print('AUC=',auc(fpr_test, tpr_test))
        print('Accuracy=',accuracy_score(y_test, (randomforest_y_test_predict3 > threshold)))
```

```
Training result:
('AUC=', 0.7984144166234035)
('Accuracy=', 0.8897542040806299)
Testing result:
('AUC=', 0.6612977707103164)
('Accuracy=', 0.8884193976317988)
```

It finally can get 0.6624 score on the test data.

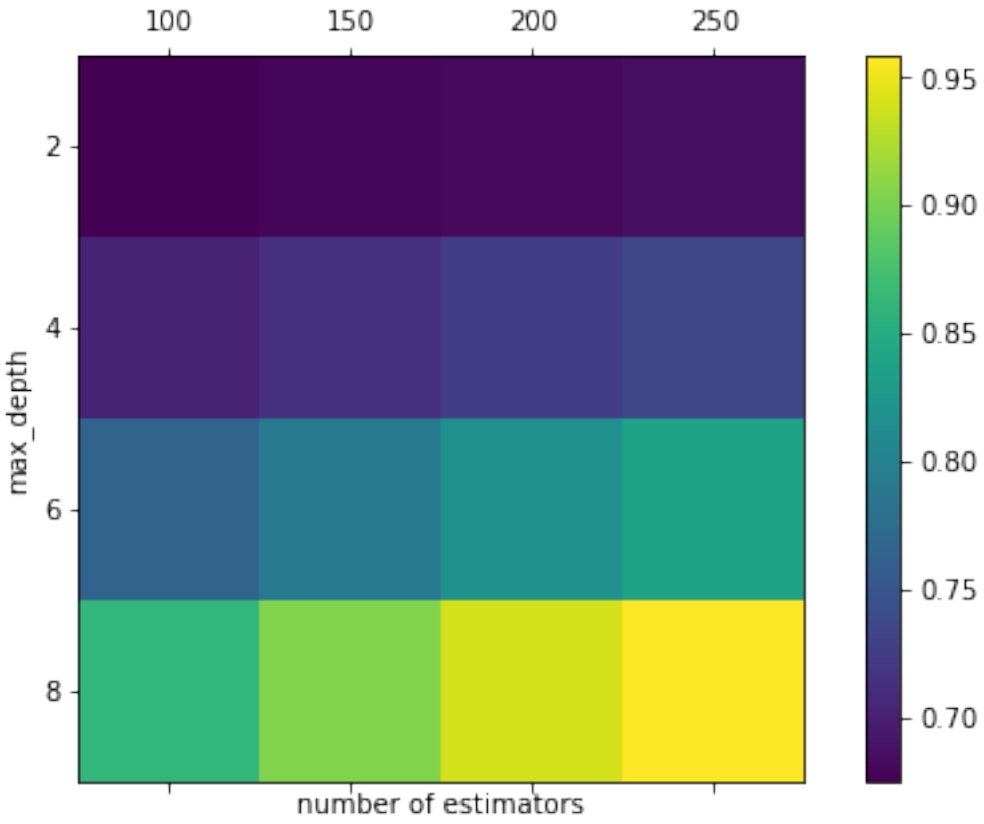
3.3 Gradient boosting

This time we use a gradient boosting classifier model.

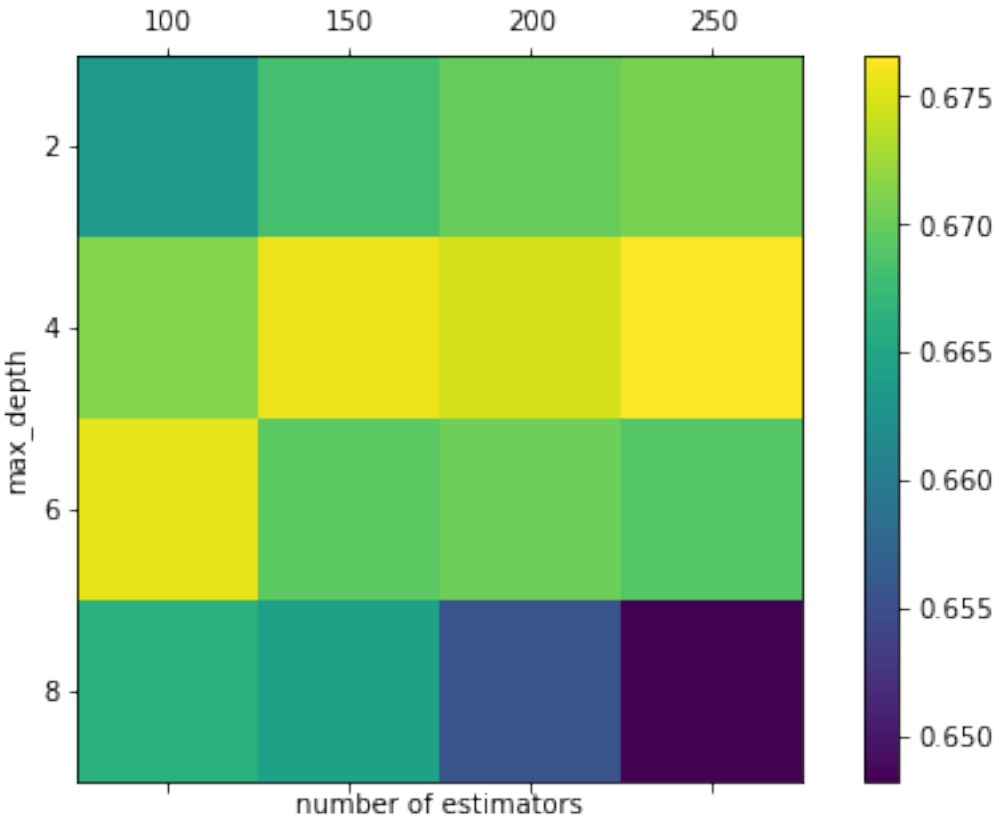
```
In [28]: from sklearn.ensemble import GradientBoostingClassifier

        threshold = 0.5

        numEstimators = range(100,300,50) # Or something else
        max_depth = range(2,10,2) # Or something else
        tr_auc = np.zeros((len(max_depth),len(numEstimators)))
        va_auc = np.zeros((len(max_depth),len(numEstimators)))
        for i,dpth in enumerate(max_depth):
            for j,numEst in enumerate(numEstimators):
                gboost = GradientBoostingClassifier(learning_rate=0.1, n_estimators=numEst,
                        subsample=0.8, min_samples_split=2,
                        min_samples_leaf=1, max_depth=dpth)

                gboost.fit(X_train, y_train)
                gboost_y_train_predict = gboost.predict_proba(X_train)[:,1]
                gboost_y_test_predict = gboost.predict_proba(X_test)[:,1]

                fpr_train, tpr_train, thresholds_train = roc_curve(y_train, gboost_y_train_pre
                tr_auc[i][j] = auc(fpr_train, tpr_train)
```

```
                fpr_test, tpr_test, thresholds_test = roc_curve(y_test, gboost_y_test_predict)
                va_auc[i][j] = auc(fpr_test, tpr_test)

        plt.figure(1)
        f, ax = plt.subplots(1, 1,figsize=(8, 5))
        cax = ax.matshow(tr_auc, interpolation='nearest')
        f.colorbar(cax)
        ax.set_xticklabels(['']+list(numEstimators))
        ax.set_yticklabels(['']+list(max_depth))
        plt.xlabel('number of estimators')
        plt.ylabel('max_depth')
        plt.show()

        plt.figure(2)
        f, ax = plt.subplots(1, 1,figsize=(8, 5))
        cax = ax.matshow(va_auc, interpolation='nearest')
        f.colorbar(cax)
        ax.set_xticklabels(['']+list(numEstimators))
        ax.set_yticklabels(['']+list(max_depth))
        plt.xlabel('number of estimators')
        plt.ylabel('max_depth')
        plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```



According to the above images, we can figure out it got max value when max_depth=4 and numbers of estimators=250

```
In [26]: from sklearn.ensemble import GradientBoostingClassifier

         gboost = GradientBoostingClassifier(learning_rate=0.1, n_estimators=250,
                                             subsample=0.8, min_samples_split=2,
                                             min_samples_leaf=1, max_depth=4)
         gboost.fit(X_train, y_train)

         gboost_y_train_predict = gboost.predict_proba(X_train)[:,1]
         gboost_y_test_predict = gboost.predict_proba(X_test)[:,1]

         print('Training result:')
         fpr_train, tpr_train, thresholds_train = roc_curve(y_train, gboost_y_train_predict)
         print('AUC=',auc(fpr_train, tpr_train))
         print('Accuracy=',accuracy_score(y_train, (gboost_y_train_predict > threshold)))

         print('Testing result:')
         fpr_test, tpr_test, thresholds_test = roc_curve(y_test, gboost_y_test_predict)
         print('AUC=',auc(fpr_test, tpr_test))
         print('Accuracy=',accuracy_score(y_test, (gboost_y_test_predict > threshold)))
```

```
Training result:
('AUC=', 0.7354316578069588)
('Accuracy=', 0.8922723531796238)
Testing result:
('AUC=', 0.6749433772334535)
('Accuracy=', 0.8877315383481551)
```

The gradient boosting method is used widely in industrial, it can get 0.6749 score on the test data

4 Conclusion

In this project, we have studied the Diabetes 130-US hospitals data to perform classification on the readmitted status. The original data has 49 features, but some of them are not useful or missing information a lot. By examining the characteristic of all features, we dropped 25 unuseful features and kept 24 useful features for the classification task. We split the data into 80% for training and 20% for testing.

We have carried out several methods such as logistic regression, random forest and gradient boosting. The testing AUC of logistic regression, random forest and gradient boosting are 0.638, 0.662, and 0.6749, respectively. Hence, among the 3 mentioned methods, gradient boosting gives the best performance. For the random forest, AUC of 0.662 is obtained with setting min_samples_plit = 16, max_depth = 12, n_estimators = 75, and min_samples_leaf = 1. For the gradient boosting method, AUC of 0.6749 is obtained with setting n_estimators = 250, max_depth = 4, min_samples_split = 2, and min_samples_leaf = 1.