Random Forest - Credit Default Prediction

In this case study, we will build a random forest model to predict whether a given customer defaults or not. Credit default is one of the most important problems in the banking and risk analytics industry. There are various attributes which can be used to predict default, such as demographic data (age, income, employment status, etc.), (credit) behavioural data (past loans, payment, number of times a credit payment has been delayed by the customer etc.).

We'll start the process with data cleaning and preparation and then tune the model to find optimal hyperparameters.

ID: ID of each client
LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
SEX: Gender (1=male, 2=female)
EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
MARRIAGE: Marital status (1=married, 2=single, 3=others)
AGE: Age in years
PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months, 9=payment delay for nine months and above)
PAY_2: Repayment status in August, 2005 (scale same as above)
PAY_3: Repayment status in July, 2005 (scale same as above)
PAY_4: Repayment status in June, 2005 (scale same as above)
PAY_5: Repayment status in May, 2005 (scale same as above)
PAY_6: Repayment status in April, 2005 (scale same as above)
BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
default.payment.next.month: Default payment (1=yes, 0=no)

Data Understanding and Cleaning

In [1]:
```
# Importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# To ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Reading the csv file and putting it into 'df' object.
df = pd.read_csv('credit-card-default.csv')
df.head()
```

Out[2]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_A
0	1	20000	2	2	1	24	2	2	-1	-1	...	
1	2	120000	2	2	2	26	-1	2	0	0	...	
2	3	90000	2	2	2	34	0	0	0	0	...	14
3	4	50000	2	2	1	37	0	0	0	0	...	2
4	5	50000	1	2	1	57	-1	0	-1	0	...	2

5 rows × 25 columns

In [3]:
```
# Let's understand the type of columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
ID           30000 non-null int64
LIMIT_BAL    30000 non-null int64
SEX          30000 non-null int64
EDUCATION    30000 non-null int64
MARRIAGE     30000 non-null int64
AGE          30000 non-null int64
PAY_0        30000 non-null int64
PAY_2        30000 non-null int64
PAY_3        30000 non-null int64
PAY_4        30000 non-null int64
PAY_5        30000 non-null int64
PAY_6        30000 non-null int64
BILL_AMT1    30000 non-null int64
BILL_AMT2    30000 non-null int64
BILL_AMT3    30000 non-null int64
BILL_AMT4    30000 non-null int64
```

In this case, we know that there are no major data quality issues, so we'll go ahead and build the model.

Data Preparation and Model Building

In [4]:
```
# Importing test_train_split from sklearn library
from sklearn.model_selection import train_test_split
```

In [16]:
```
# Putting feature variable to X
X = df.drop('defaulted',axis=1)

# Putting response variable to y
y = df['defaulted']

# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=101)
```

Default Hyperparameters

Let's first fit a random forest model with default hyperparameters.

In [17]:
```
# Importing random forest classifier from sklearn library
from sklearn.ensemble import RandomForestClassifier

# Running the random forest with default parameters.
rfc = RandomForestClassifier()
```

In [19]:
```
# fit
rfc.fit(X_train,y_train)
```

Out[19]:
```
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=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [20]:
```
# Making predictions
predictions = rfc.predict(X_test)
```

In [21]:
```
# Importing classification report and confusion matrix from sklearn metrics
from sklearn.metrics import classification_report,confusion_matrix,
accuracy_score
```

In [22]:
```python
# Let's check the report of our default model
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.83      0.94      0.88      7058
          1       0.59      0.32      0.41      1942

avg / total       0.78      0.81      0.78      9000
```

In [23]:
```python
# Printing confusion matrix
print(confusion_matrix(y_test,predictions))
```

```
[[6639  419]
 [1330  612]]
```

In [24]:
```python
print(accuracy_score(y_test,predictions))
```

```
0.8056666666666666
```

So far so good, let's now look at the list of hyperparameters which we can tune to improve model performance.

Hyperparameter Tuning

Tuning max_depth

Let's try to find the optimum values for `max_depth` and understand how the value of max_depth impacts the overall accuracy of the ensemble.

In [25]:
```
# GridSearchCV to find optimal n_estimators
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'max_depth': range(2, 20, 5)}

# instantiate the model
rf = RandomForestClassifier()


# fit tree on training data
rf = GridSearchCV(rf, parameters,
                    cv=n_folds,
                   scoring="accuracy")
rf.fit(X_train, y_train)
```

Out[25]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, crit
erion='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=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_depth': range(2, 20, 5)}, pre_dispatch='2*n_jobs',
       refit=True, return_train_score='warn', scoring='accuracy',
       verbose=0)
```

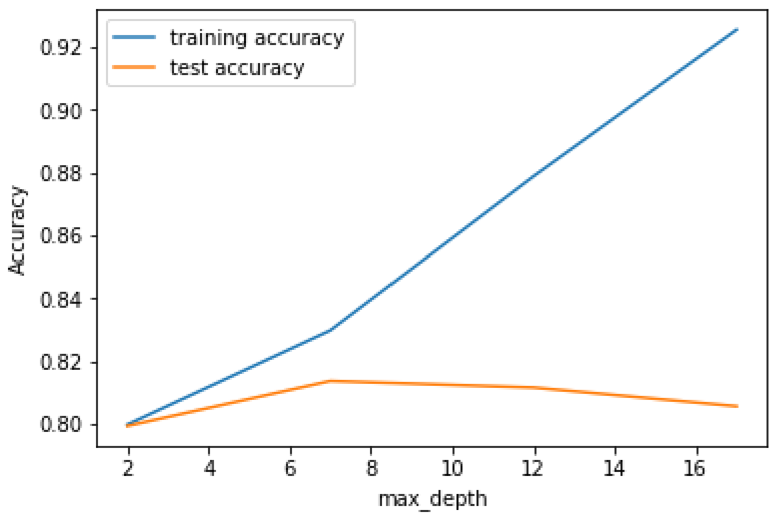
In [27]:
```
# scores of GridSearch CV
scores = rf.cv_results_
pd.DataFrame(scores).head()
```

Out[27]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params
0	0.159305	0.030066	0.007798	0.001166	2	{'max_depth': 2}
1	0.371566	0.030511	0.010917	0.002334	7	{'max_depth': 7}
2	0.588663	0.052848	0.014192	0.000750	12	{'max_depth': 12}
3	0.786404	0.131485	0.016392	0.002798	17	{'max_depth': 17}

4 rows × 21 columns

In [28]:
```
# plotting accuracies with max_depth
plt.figure()
plt.plot(scores["param_max_depth"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_max_depth"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.legend()
```



In [15]:
```
rf.best_params_
```

Out[15]:
```
{'max_depth': 7}
```

You can see that as we increase the value of max_depth, both train and test scores increase till a point, but after that test score starts to decrease. The ensemble tries to overfit as we increase the max_depth.

Tuning n_estimators

Let's try to find the optimum values for n_estimators and understand how the value of n_estimators impacts the overall accuracy. Notice that we'll specify an appropriately low value of max_depth, so that the trees do not overfit.

In [16]:
```
# GridSearchCV to find optimal n_estimators
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'n_estimators': range(100, 1500, 400)}

# instantiate the model (note we are specifying a max_depth)
rf = RandomForestClassifier(max_depth=4)


# fit tree on training data
rf = GridSearchCV(rf, parameters,
                    cv=n_folds,
                   scoring="accuracy")
rf.fit(X_train, y_train)
```

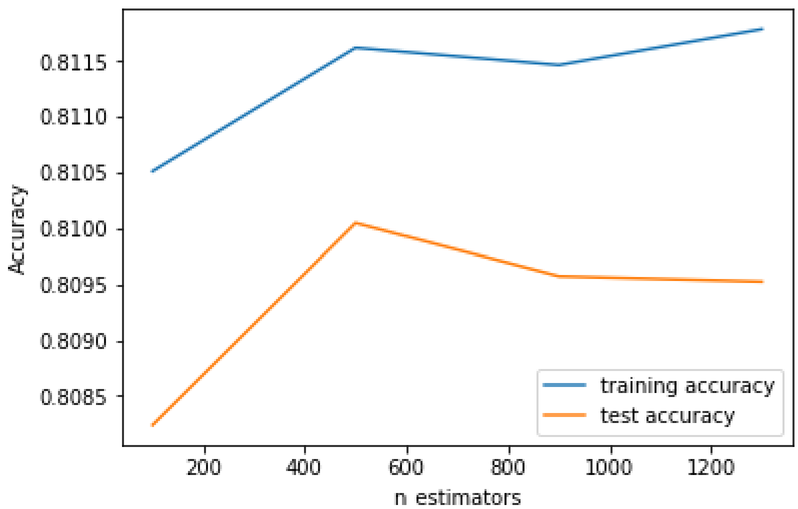
Out[16]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, crit
erion='gini',
            max_depth=4, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_estimators': range(100, 1500, 400)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

In [17]:
```
# scores of GridSearch CV
scores = rf.cv_results_
pd.DataFrame(scores).head()
```

Out[17]:

t_score	split2_train_score	split3_test_score	split3_train_score	split4_test_score	split4_train_score	s
.810000	0.811786	0.802143	0.810476	0.805192	0.810011	
).811190	0.811071	0.805476	0.812738	0.807811	0.812392	
.811429	0.811726	0.804524	0.812381	0.806859	0.812154	
).811190	0.812202	0.805952	0.812560	0.806382	0.812333	

In [18]:
```
# plotting accuracies with n_estimators
plt.figure()
plt.plot(scores["param_n_estimators"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_n_estimators"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("n_estimators")
plt.ylabel("Accuracy")
plt.legend()
```



Tuning max_features

Let's see how the model performance varies with `max_features` , which is the maximum numbre of features considered for splitting at a node.

In [19]:
```
# GridSearchCV to find optimal max_features
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'max_features': [4, 8, 14, 20, 24]}

# instantiate the model
rf = RandomForestClassifier(max_depth=4)


# fit tree on training data
rf = GridSearchCV(rf, parameters,
                    cv=n_folds,
                   scoring="accuracy")
rf.fit(X_train, y_train)
```

Out[19]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, crit
erion='gini',
            max_depth=4, 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=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_features': [4, 8, 14, 20, 24]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

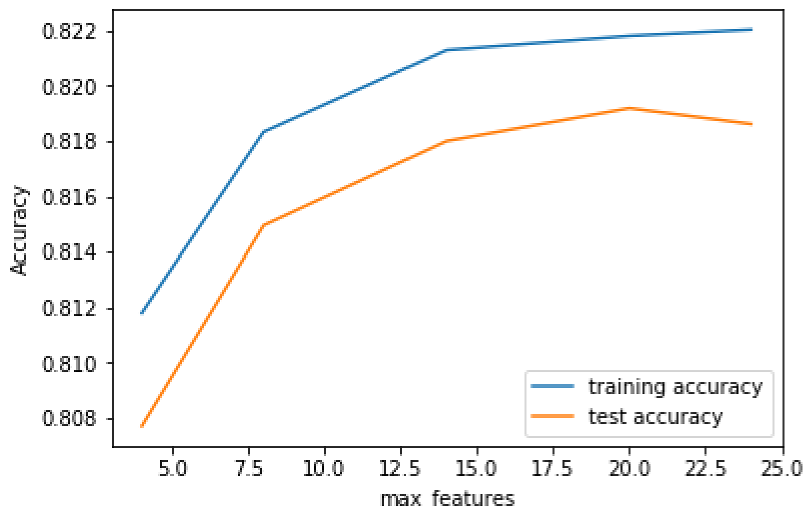
In [20]:
```
# scores of GridSearch CV
scores = rf.cv_results_
pd.DataFrame(scores).head()
```

Out[20]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_features	
0	0.143250	0.004957	0.807714	0.811810	4	{'max
1	0.238311	0.004913	0.814952	0.818333	8	{'max
2	0.388783	0.004893	0.818000	0.821298	14	{'max
3	0.517523	0.004974	0.819190	0.821810	20	{'max
4	0.634080	0.004955	0.818619	0.822036	24	{'max

5 rows × 21 columns

In [21]:
```
# plotting accuracies with max_features
plt.figure()
plt.plot(scores["param_max_features"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_max_features"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("max_features")
plt.ylabel("Accuracy")
plt.legend()
```



Apparently, the training and test scores *both* seem to increase as we increase max_features, and the model doesn't seem to overfit more with increasing max_features. Think about why that might be the case.

Tuning min_samples_leaf

The hyperparameter **min_samples_leaf** is the minimum number of samples required to be at a leaf node:

- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

Let's now check the optimum value for min samples leaf in our case.

In [22]:
```
# GridSearchCV to find optimal min_samples_leaf
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'min_samples_leaf': range(100, 400, 50)}

# instantiate the model
rf = RandomForestClassifier()


# fit tree on training data
rf = GridSearchCV(rf, parameters,
                    cv=n_folds,
                   scoring="accuracy")
rf.fit(X_train, y_train)
```

Out[22]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, crit
erion='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=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'min_samples_leaf': range(100, 400, 50)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

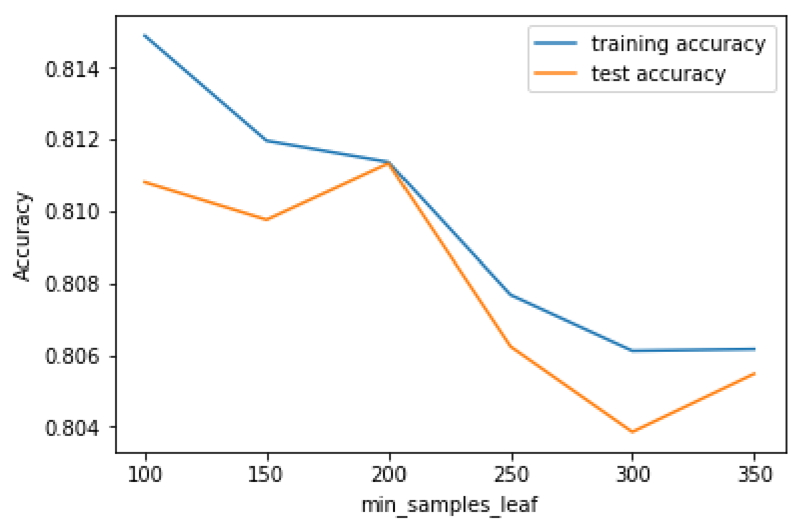
In [23]:
```
# scores of GridSearch CV
scores = rf.cv_results_
pd.DataFrame(scores).head()
```

Out[23]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_min_samples_leaf	
0	0.241674	0.006666	0.810810	0.814893	100	{
1	0.220668	0.015714	0.809762	0.811964	150	{
2	0.211578	0.006002	0.811333	0.811369	200	{
3	0.206866	0.006105	0.806238	0.807679	250	{
4	0.186352	0.005877	0.803857	0.806119	300	{

5 rows × 21 columns

In [24]:
```
# plotting accuracies with min_samples_leaf
plt.figure()
plt.plot(scores["param_min_samples_leaf"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_min_samples_leaf"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("min_samples_leaf")
plt.ylabel("Accuracy")
plt.legend()
```



You can see that the model starts of overfit as you decrease the value of min_samples_leaf.

Tuning min_samples_split

Let's now look at the performance of the ensemble as we vary min_samples_split.

In [25]:
```
# GridSearchCV to find optimal min_samples_split
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'min_samples_split': range(200, 500, 50)}

# instantiate the model
rf = RandomForestClassifier()


# fit tree on training data
rf = GridSearchCV(rf, parameters,
                    cv=n_folds,
                   scoring="accuracy")
rf.fit(X_train, y_train)
```

Out[25]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, crit
erion='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=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'min_samples_split': range(200, 500, 50)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

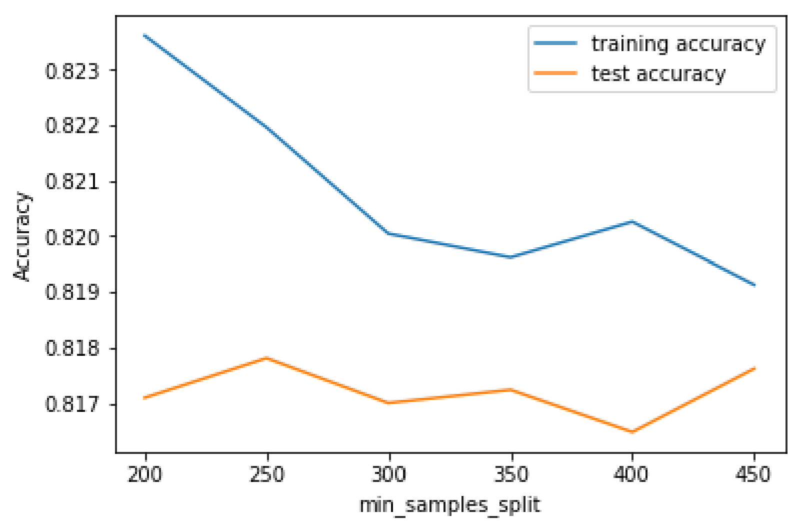
In [26]:
```
# scores of GridSearch CV
scores = rf.cv_results_
pd.DataFrame(scores).head()
```

Out[26]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_min_samples_split
0	0.356812	0.007173	0.817095	0.823619	200
1	0.328454	0.007153	0.817810	0.821964	250
2	0.313334	0.007008	0.817000	0.820048	300
3	0.295372	0.006546	0.817238	0.819619	350
4	0.288697	0.006540	0.816476	0.820262	400

5 rows × 21 columns

In [27]:
```
# plotting accuracies with min_samples_split
plt.figure()
plt.plot(scores["param_min_samples_split"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_min_samples_split"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("min_samples_split")
plt.ylabel("Accuracy")
plt.legend()
```



Grid Search to Find Optimal Hyperparameters

We can now find the optimal hyperparameters using GridSearchCV.

In [28]:
```
# Create the parameter grid based on the results of random search
param_grid = {
    'max_depth': [4,8,10],
    'min_samples_leaf': range(100, 400, 200),
    'min_samples_split': range(200, 500, 200),
    'n_estimators': [100,200, 300],
    'max_features': [5, 10]
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1,verbose = 1)
```

In [29]:
```
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   29.2s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done 216 out of 216 | elapsed:  4.7min finished
```

Out[29]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, crit
erion='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=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'max_depth': [4, 8, 10], 'n_estimators': [100, 200, 300], 'm
in_samples_leaf': range(100, 400, 200), 'min_samples_split': range(200, 500, 20
0), 'max_features': [5, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=1)
```

In [30]:
```
# printing the optimal accuracy score and hyperparameters
print('We can get accuracy
of',grid_search.best_score_,'using',grid_search.best_params_)
```

```
We can get accuracy of 0.818523809524 using {'max_depth': 4, 'n_estimators': 20
0, 'min_samples_leaf': 100, 'min_samples_split': 400, 'max_features': 10}
```

Fitting the final model with the best parameters obtained from grid search.

In [31]:
```
# model with the best hyperparameters
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(bootstrap=True,
                             max_depth=10,
                             min_samples_leaf=100,
                             min_samples_split=200,
                             max_features=10,
                             n_estimators=100)
```

In [32]:
```
# fit
rfc.fit(X_train,y_train)
```

Out[32]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=10, max_features=10, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=100, min_samples_split=200,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [33]:
```
# predict
predictions = rfc.predict(X_test)
```

In [34]:
```
# evaluation metrics
from sklearn.metrics import classification_report,confusion_matrix
```

In [35]:
```
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.84      0.96      0.90      7058
          1       0.69      0.36      0.47      1942

avg / total       0.81      0.83      0.80      9000
```

In [37]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[6753  305]
 [1250  692]]
```

In [38]:
```
(6753+692)/(6753+692+305+1250)
```

Out[38]:
```
0.8272222222222222
```

In [ ]:
```

```