Project 2

Project Description:

- Use same datasets as Project 1.
- Preprocess data: Explore data and apply data scaling.

Regression Task:

- Apply any two models with bagging and any two models with pasting.
- Apply any two models with adaboost boosting
- Apply one model with gradient boosting
- Apply PCA on data and then apply all the models in project 1 again on data you get from PCA. Compare your results with results in project 2. You don't need to apply all the models twice. Just copy the result table from project 1, prepare similar table for all the models after PCA and compare both tables. Does PCA help in getting better results?
- Apply deep learning models covered in class

Classification Task:

- Apply two voting classifiers - one with hard voting and one with soft voting
- Apply any two models with bagging and any two models with pasting.
- Apply any two models with adaboost boosting
- Apply one model with gradient boosting
- Apply PCA on data and then apply all the models in project 1 again on data you get from PCA. Compare your results with results in project 1. You don't need to apply all the models twice. Just copy the result table from project 1, prepare similar table for all the models after PCA and compare both tables. Does PCA help in getting better results?
- Apply deep learning models covered in class

Deliverables:

- Use markdown to provide inline comments for this project.
- Your outputs should be clearly executed in the notebook i.e. we should not need to rerun the code to obtain the outputs.
- Visualization encouraged.
- If you are submitting two different files, then please only one group member submit both the files. If you submit two files separately from different accounts, it will be submitted as two different attempts.
- If you are submitting two different files, then please follow below naming convetion:
  Project2_Regression_GroupXX_Firstname1_Firstname2.ipynb
  Project2_Classification_GroupXX_Firstname1_Firstname2.ipynb
- If you are submitting single file, then please follow below naming convetion:
  Project2_Both_GroupXX_Firstname1_Firstname2.ipynb

Questions regarding the project:

- We have created a discussion board under Projects folder on e-learning. Create threads over there and post your queries related to project there.

- We will also answer queries there. We will not be answering any project related queries through the mail.

libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Using pandas reading the audit_risk and trial.csv.

In [2]:
```
audit_risk_df = pd.read_csv("audit_risk.csv")
trial_df = pd.read_csv("trial.csv")
audit_risk_df.T
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9
Sector_score	3.89	3.89	3.89	3.89	3.89	3.89	3.89	3.89	3.89	3.89
LOCATION_ID	23	6	6	6	6	6	7	8	8	8
PARA_A	4.18	0	0.51	0	0	0	1.1	8.5	8.4	3.98
Score_A	0.6	0.2	0.2	0.2	0.2	0.2	0.4	0.6	0.6	0.6
Risk_A	2.508	0	0.102	0	0	0	0.44	5.1	5.04	2.388
PARA_B	2.5	4.83	0.23	10.8	0.08	0.83	7.41	12.03	11.05	0.99
Score_B	0.2	0.2	0.2	0.6	0.2	0.2	0.4	0.6	0.6	0.2
Risk_B	0.5	0.966	0.046	6.48	0.016	0.166	2.964	7.218	6.63	0.198
TOTAL	6.68	4.83	0.74	10.8	0.08	0.83	8.51	20.53	19.45	4.97
numbers	5	5	5	6	5	5	5	5.5	5.5	5
Score_B.1	0.2	0.2	0.2	0.6	0.2	0.2	0.2	0.4	0.4	0.2
Risk_C	1	1	1	3.6	1	1	1	2.2	2.2	1
Money_Value	3.38	0.94	0	11.75	0	2.95	44.95	7.79	7.34	1.93
Score_MV	0.2	0.2	0.2	0.6	0.2	0.2	0.6	0.4	0.4	0.2
Risk_D	0.676	0.188	0	7.05	0	0.59	26.97	3.116	2.936	0.386
District_Loss	2	2	2	2	2	2	2	2	2	2
PROB	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
RiSk_E	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
History	0	0	0	0	0	0	0	0	0	0
Prob	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Risk_F	0	0	0	0	0	0	0	0	0	0
Score	2.4	2	2	4.4	2	2	3.2	4.2	4.2	2.4
Inherent_Risk	8.574	2.554	1.548	17.53	1.416	2.156	31.774	18.034	17.206	4.372
CONTROL_RISK	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
Detection_Risk	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Audit_Risk	1.7148	0.5108	0.3096	3.506	0.2832	0.4312	6.3548	3.6068	3.4412	0.8744
Risk	1	0	0	1	0	0	1	1	1	0

27 rows × 776 columns

In [3]:
```
audit_risk_df.rename(columns={'PROB': 'PROB1'}, inplace=True)
```

In [4]:
```
print(audit_risk_df.columns)
print(trial_df.columns)
```

```
Index(['Sector_score', 'LOCATION_ID', 'PARA_A', 'Score_A', 'Risk_A', 'PARA_
B',
       'Score_B', 'Risk_B', 'TOTAL', 'numbers', 'Score_B.1', 'Risk_C',
       'Money_Value', 'Score_MV', 'Risk_D', 'District_Loss', 'PROB1', 'RiSk
_E',
       'History', 'Prob', 'Risk_F', 'Score', 'Inherent_Risk', 'CONTROL_RIS
K',
       'Detection_Risk', 'Audit_Risk', 'Risk'],
      dtype='object')
Index(['Sector_score', 'LOCATION_ID', 'PARA_A', 'SCORE_A', 'PARA_B', 'SCORE
_B',
       'TOTAL', 'numbers', 'Marks', 'Money_Value', 'MONEY_Marks', 'Distric
t',
       'Loss', 'LOSS_SCORE', 'History', 'History_score', 'Score', 'Risk'],
      dtype='object')
```

After reading the two datasets the following observations were made :

Detection_Risk is a constant value

In [5]:
```
audit_risk_df = audit_risk_df.drop("Detection_Risk", axis = 1)
```

In [6]:
```
1 trial_df
```

Out[6]:

	Sector_score	LOCATION_ID	PARA_A	SCORE_A	PARA_B	SCORE_B	TOTAL	numbers
0	3.89	23	4.18	6	2.5000	2	6.6800	5.0
1	3.89	6	0.00	2	4.8300	2	4.8300	5.0
2	3.89	6	0.51	2	0.2300	2	0.7400	5.0
3	3.89	6	0.00	2	10.8000	6	10.8000	6.0
4	3.89	6	0.00	2	0.0800	2	0.0800	5.0
5	3.89	6	0.00	2	0.8300	2	0.8300	5.0
6	3.89	7	1.10	4	7.4100	4	8.5100	5.0
7	3.89	8	8.50	6	12.0300	6	20.5300	5.5
8	3.89	8	8.40	6	11.0500	6	19.4500	5.5
9	3.89	8	3.98	6	0.9900	2	4.9700	5.0
10	3.89	8	5.43	6	10.7700	6	16.2000	5.0
11	3.89	8	15.38	6	40.1400	6	55.5200	5.0
12	3.89	8	5.47	6	7.6300	4	13.1000	5.0
13	3.89	8	1.09	4	0.3500	2	1.4400	5.0
14	3.89	8	0.00	2	0.8400	2	0.8400	5.0
15	3.89	13	1.95	4	9.0100	4	10.9600	5.0
16	3.89	37	8.54	6	31.6300	6	40.1700	5.0
17	3.89	37	4.18	6	4.8300	2	9.0100	5.5
18	3.89	37	1.81	4	1.0300	2	2.8400	5.0
19	3.89	37	4.86	6	46.7800	6	51.6400	5.5
20	3.89	24	6.26	6	14.1000	6	20.3600	5.0
21	3.89	3	0.02	2	5.9400	4	5.9600	5.0
22	3.89	3	5.31	6	22.7900	6	28.1000	5.0
23	3.89	3	0.94	2	0.0100	2	0.9500	5.0
24	3.89	4	5.78	6	57.9200	6	63.7000	5.0
25	3.89	4	7.42	6	2.2400	2	9.6600	5.0
26	3.89	4	0.00	2	1.1000	2	1.1000	5.0
27	3.89	14	6.85	6	31.7600	6	38.6100	5.0
28	3.89	14	0.00	2	1.0300	2	1.0300	5.0
29	3.89	37	0.00	2	0.7500	2	0.7500	5.0
...	...	...	...	...	...	...	...	...
746	55.57	13	0.25	2	0.0017	2	0.2517	5.0
747	55.57	13	0.31	2	0.0015	2	0.3115	5.0
748	55.57	13	0.00	2	0.0000	2	0.0000	5.0

	Sector_score	LOCATION_ID	PARA_A	SCORE_A	PARA_B	SCORE_B	TOTAL	numbers
749	55.57	13	0.84	2	0.0000	2	0.8400	5.0
750	55.57	13	1.09	4	0.0000	2	1.0900	5.0
751	55.57	13	1.29	4	0.0000	2	1.2900	5.0
752	55.57	13	0.51	2	0.3700	2	0.8800	5.0
753	55.57	21	0.09	2	0.0000	2	0.0900	5.0
754	55.57	18	0.39	2	0.9100	2	1.3000	5.0
755	55.57	21	1.07	4	0.0000	2	1.0700	5.0
756	55.57	25	0.00	2	0.0000	2	0.0000	5.0
757	55.57	32	0.50	2	2.9700	6	3.4700	5.0
758	55.57	22	0.49	2	0.5500	2	1.0400	5.0
759	55.57	14	0.84	2	0.6500	2	1.4900	5.0
760	55.57	12	0.90	2	1.1100	4	2.0100	5.0
761	55.57	12	0.00	2	0.0000	2	0.0000	5.0
762	55.57	14	0.59	2	0.0000	2	0.5900	5.0
763	55.57	36	0.02	2	0.0000	2	0.0200	5.0
764	55.57	14	1.48	4	4.4800	6	5.9600	5.0
765	55.57	22	0.00	2	3.3000	6	3.3000	5.0
766	55.57	8	0.80	2	0.5700	2	1.3700	5.0
767	55.57	18	0.36	2	0.5400	2	0.9000	5.0
768	55.57	9	0.44	2	0.5300	2	0.9700	5.0
769	55.57	16	0.51	2	0.5000	2	1.0100	5.0
770	55.57	18	0.75	2	0.4500	2	1.2000	5.0
771	55.57	9	0.49	2	0.4000	2	0.8900	5.0
772	55.57	16	0.47	2	0.3700	2	0.8400	5.0
773	55.57	14	0.24	2	0.0400	2	0.2800	5.0
774	55.57	18	0.20	2	0.0000	2	0.2000	5.0
775	55.57	15	0.00	2	0.0000	2	0.0000	5.0

776 rows × 18 columns

SCORE_A and SCORE_B in trial_df and audit_risk_df are on different scales so we will make them on same scale.

In [7]:

```
audit_risk_df["Score_A"] = audit_risk_df["Score_A"]*10
audit_risk_df["Score_B"] = audit_risk_df["Score_B"]*10
```

In [8]:
```
c_with_risk_cols = ['Sector_score', 'LOCATION_ID', 'PARA_A', 'Score_A',
c_without_risk_cols = ['Sector_score', 'LOCATION_ID', 'PARA_A', 'Score_A
c_with_risk_cols_upper = [x.upper() for x in c_with_risk_cols]
c_without_risk_cols_upper = [x.upper() for x in c_without_risk_cols]

audit_names = audit_risk_df.columns
audit_names_upper =  [x.upper() for x in audit_names]
audit_risk_df.columns = audit_names_upper

trial_names = trial_df.columns
trial_names_upper =  [x.upper() for x in trial_names]
trial_df.columns = trial_names_upper
```

In [9]:
```
L= ['SECTOR_SCORE', 'LOCATION_ID', 'PARA_A', 'SCORE_A', 'PARA_B', 'SCORE
```

In [10]:
```
audit_risk = audit_risk_df.merge(trial_df, on=L)
```

In [11]:
```
audit_risk['RISK'].unique()
```

Out[11]:
```
array([1, 0])
```

In [12]:
```
audit_risk = audit_risk.drop(["MONEY_MARKS","DISTRICT"], axis=1)
```

In [13]:
```
1 audit_risk.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 629 entries, 0 to 628
Data columns (total 30 columns):
SECTOR_SCORE     629 non-null float64
LOCATION_ID      629 non-null object
PARA_A           629 non-null float64
SCORE_A          629 non-null float64
RISK_A           629 non-null float64
PARA_B           629 non-null float64
SCORE_B          629 non-null float64
RISK_B           629 non-null float64
TOTAL            629 non-null float64
NUMBERS          629 non-null float64
SCORE_B.1        629 non-null float64
RISK_C           629 non-null float64
MONEY_VALUE      628 non-null float64
SCORE_MV         629 non-null float64
RISK_D           629 non-null float64
DISTRICT_LOSS    629 non-null int64
PROB1            629 non-null float64
RISK_E           629 non-null float64
HISTORY          629 non-null int64
PROB             629 non-null float64
RISK_F           629 non-null float64
SCORE            629 non-null float64
INHERENT_RISK    629 non-null float64
CONTROL_RISK     629 non-null float64
AUDIT_RISK       629 non-null float64
RISK             629 non-null int64
MARKS            629 non-null int64
LOSS             629 non-null int64
LOSS_SCORE       629 non-null int64
HISTORY_SCORE    629 non-null int64
dtypes: float64(22), int64(7), object(1)
memory usage: 152.3+ KB
```

In [14]:
```
audit_risk['MONEY_VALUE'] = audit_risk["MONEY_VALUE"].fillna(audit_risk[
# merged_data_sans_dup = merged_data_sans_dup["Money_Value"].fillna(merg

audit_risk.isnull().sum()
```

Out[14]:
```
SECTOR_SCORE     0
LOCATION_ID      0
PARA_A           0
SCORE_A          0
RISK_A           0
PARA_B           0
SCORE_B          0
RISK_B           0
TOTAL            0
NUMBERS          0
SCORE_B.1        0
RISK_C           0
MONEY_VALUE      0
SCORE_MV         0
RISK_D           0
DISTRICT_LOSS    0
PROB1            0
RISK_E           0
HISTORY          0
PROB             0
RISK_F           0
SCORE            0
INHERENT_RISK    0
CONTROL_RISK     0
AUDIT_RISK       0
RISK             0
MARKS            0
LOSS             0
LOSS_SCORE       0
HISTORY_SCORE    0
dtype: int64
```

We observe that LOCATION_ID for three rows has different values so we fix them

In [15]:
```
audit_risk["LOCATION_ID"]= audit_risk["LOCATION_ID"].replace("LOHARU", 4
audit_risk["LOCATION_ID"]= audit_risk["LOCATION_ID"].replace("NUH", 46)
audit_risk["LOCATION_ID"]= audit_risk["LOCATION_ID"].replace("SAFIDON",
```

In [16]:
```
sans_out = audit_risk[audit_risk.PARA_B != 1264.630000]
audit_risk.shape
```

Out[16]: (629, 30)

In [17]:
```
1 sans_out[['MONEY_VALUE','RISK_D']].describe()
```

Out[17]:

	MONEY_VALUE	RISK_D
count	628.000000	628.000000
mean	17.235104	10.114415
std	73.623456	44.213178
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.140000	0.027000
75%	9.107500	3.585000
max	935.030000	561.018000

In [18]:
```
1 sans_out[(sans_out['INHERENT_RISK'] == 622.838000) | (sans_out['TOTAL']
```

Out[18]:

	SECTOR_SCORE	LOCATION_ID	PARA_A	SCORE_A	RISK_A	PARA_B	SCORE_B	RISK
34	3.89	19	7.97	6.0	4.782	17.18	6.0	10.
288	1.99	2	57.03	6.0	34.218	134.33	6.0	80.

2 rows × 30 columns

In [19]:
```
1 final_df = sans_out[(sans_out['INHERENT_RISK'] != 622.838000) & (sans_ou
```

In [20]:
```
final_df.shape
final_df.isnull().any()
```

Out[20]:
```
SECTOR_SCORE     False
LOCATION_ID      False
PARA_A           False
SCORE_A          False
RISK_A           False
PARA_B           False
SCORE_B          False
RISK_B           False
TOTAL            False
NUMBERS          False
SCORE_B.1        False
RISK_C           False
MONEY_VALUE      False
SCORE_MV         False
RISK_D           False
DISTRICT_LOSS    False
PROB1            False
RISK_E           False
HISTORY          False
PROB             False
RISK_F           False
SCORE            False
INHERENT_RISK    False
CONTROL_RISK     False
AUDIT_RISK       False
RISK             False
MARKS            False
LOSS             False
LOSS_SCORE       False
HISTORY_SCORE    False
dtype: bool
```

In [21]:
```
final_df['RISK'].unique()
final_df.columns
```

Out[21]:
```
Index(['SECTOR_SCORE', 'LOCATION_ID', 'PARA_A', 'SCORE_A', 'RISK_A', 'PARA_
B',
       'SCORE_B', 'RISK_B', 'TOTAL', 'NUMBERS', 'SCORE_B.1', 'RISK_C',
       'MONEY_VALUE', 'SCORE_MV', 'RISK_D', 'DISTRICT_LOSS', 'PROB1', 'RISK
_E',
       'HISTORY', 'PROB', 'RISK_F', 'SCORE', 'INHERENT_RISK', 'CONTROL_RIS
K',
       'AUDIT_RISK', 'RISK', 'MARKS', 'LOSS', 'LOSS_SCORE', 'HISTORY_SCOR
E'],
      dtype='object')
```

In [22]:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

Audit_risk = final_df.copy()
mm_scaler = MinMaxScaler()
std_scaler = StandardScaler()

y_final_reg = final_df['AUDIT_RISK']# Regression y

y_final_clf = final_df['RISK'] # Classification y
to_scale_x_df = Audit_risk.drop(["AUDIT_RISK","RISK"], axis =1)

mm_x_df = to_scale_x_df.copy()
std_x_df = to_scale_x_df.copy()

num_cols = ['SECTOR_SCORE', 'LOCATION_ID','PARA_A', 'SCORE_A', 'RISK_A',
       'SCORE_B', 'RISK_B', 'TOTAL', 'NUMBERS', 'SCORE_B.1', 'RISK_C',
       'MONEY_VALUE', 'SCORE_MV', 'RISK_D', 'DISTRICT_LOSS', 'PROB1', 'R
       'HISTORY', 'PROB', 'RISK_F', 'SCORE', 'INHERENT_RISK', 'CONTROL_R
        'MARKS', 'LOSS', 'LOSS_SCORE', 'HISTORY_SCORE']
num_cols = [x.upper() for x in num_cols]

mm_x_df[num_cols] = mm_scaler.fit_transform(mm_x_df[num_cols])       # M
std_x_df[num_cols] = std_scaler.fit_transform(std_x_df[num_cols])    # S
X=mm_x_df[num_cols]
y=y_final_clf
```

In [23]:
```
1 X
2
```

Out[23]:

	SECTOR_SCORE	LOCATION_ID	PARA_A	SCORE_A	RISK_A	PARA_B	SCORE_B	R
0	0.035172	0.478261	0.049176	1.0	0.049176	0.017314	0.0	0.0
1	0.035172	0.108696	0.000000	0.0	0.000000	0.033451	0.0	0.0
2	0.035172	0.108696	0.006000	0.0	0.002000	0.001593	0.0	0.0
3	0.035172	0.108696	0.000000	0.0	0.000000	0.074797	1.0	0.0
4	0.035172	0.108696	0.000000	0.0	0.000000	0.000554	0.0	0.0
5	0.035172	0.108696	0.000000	0.0	0.000000	0.005748	0.0	0.0
6	0.035172	0.130435	0.012941	0.5	0.008627	0.051319	0.5	0.0
7	0.035172	0.152174	0.100000	1.0	0.100000	0.083316	1.0	0.0
8	0.035172	0.152174	0.098824	1.0	0.098824	0.076529	1.0	0.0
9	0.035172	0.152174	0.063882	1.0	0.063882	0.074590	1.0	0.0
10	0.035172	0.152174	0.180941	1.0	0.180941	0.277997	1.0	0.2
11	0.035172	0.152174	0.064353	1.0	0.064353	0.052843	0.5	0.0
12	0.035172	0.152174	0.000000	0.0	0.000000	0.005818	0.0	0.0
13	0.035172	0.260870	0.022941	0.5	0.015294	0.062400	0.5	0.0
14	0.035172	0.782609	0.100471	1.0	0.100471	0.219059	1.0	0.2
15	0.035172	0.782609	0.049176	1.0	0.049176	0.033451	0.0	0.0
16	0.035172	0.782609	0.057176	1.0	0.057176	0.323984	1.0	0.3
17	0.035172	0.500000	0.073647	1.0	0.073647	0.097652	1.0	0.0
18	0.035172	0.043478	0.062471	1.0	0.062471	0.157836	1.0	0.1
19	0.035172	0.043478	0.011059	0.0	0.003686	0.000069	0.0	0.0
20	0.035172	0.065217	0.068000	1.0	0.068000	0.401136	1.0	0.4
21	0.035172	0.065217	0.087294	1.0	0.087294	0.015514	0.0	0.0
22	0.035172	0.065217	0.000000	0.0	0.000000	0.007618	0.0	0.0
23	0.035172	0.282609	0.080588	1.0	0.080588	0.219960	1.0	0.2
24	0.035172	0.282609	0.000000	0.0	0.000000	0.007133	0.0	0.0
25	0.035172	0.782609	0.028235	1.0	0.028235	0.115174	1.0	0.1
26	0.035172	0.086957	0.000000	0.0	0.000000	0.000346	0.0	0.0
27	0.035172	0.086957	0.000000	0.0	0.000000	0.012189	0.0	0.0
28	0.035172	0.086957	0.000000	0.0	0.000000	0.020569	0.0	0.0
29	0.035172	0.086957	0.000000	0.0	0.000000	0.002978	0.0	0.0
...	...	...	...	...	...	...	...	
599	0.926207	0.326087	0.002353	0.0	0.000784	0.000000	0.0	0.0
600	0.926207	0.304348	0.004706	0.0	0.001569	0.000017	0.0	0.0
601	0.926207	0.304348	0.000000	0.0	0.000000	0.006095	0.0	0.0

	SECTOR_SCORE	LOCATION_ID	PARA_A	SCORE_A	RISK_A	PARA_B	SCORE_B	R
602	0.926207	0.304348	0.000000	0.0	0.000000	0.000000	0.0	0.0
603	0.926207	0.304348	0.000000	0.0	0.000000	0.000000	0.0	0.0
604	0.926207	0.304348	0.000000	0.0	0.000000	0.000000	0.0	0.0
605	0.926207	0.304348	0.000000	0.0	0.000000	0.000000	0.0	0.0
606	0.926207	0.260870	0.001176	0.0	0.000392	0.000000	0.0	0.0
607	0.926207	0.260870	0.012471	0.5	0.008314	0.011081	0.5	0.0
608	0.926207	0.260870	0.002941	0.0	0.000980	0.000012	0.0	0.0
609	0.926207	0.260870	0.003647	0.0	0.001216	0.000010	0.0	0.0
610	0.926207	0.260870	0.000000	0.0	0.000000	0.000000	0.0	0.0
611	0.926207	0.260870	0.009882	0.0	0.003294	0.000000	0.0	0.0
612	0.926207	0.260870	0.006000	0.0	0.002000	0.002563	0.0	0.0
613	0.926207	0.434783	0.001059	0.0	0.000353	0.000000	0.0	0.0
614	0.926207	0.369565	0.004588	0.0	0.001529	0.006302	0.0	0.0
615	0.926207	0.521739	0.000000	0.0	0.000000	0.000000	0.0	0.0
616	0.926207	0.456522	0.005765	0.0	0.001922	0.003809	0.0	0.0
617	0.926207	0.282609	0.009882	0.0	0.003294	0.004502	0.0	0.0
618	0.926207	0.282609	0.006941	0.0	0.002314	0.000000	0.0	0.0
619	0.926207	0.760870	0.000235	0.0	0.000078	0.000000	0.0	0.0
620	0.926207	0.152174	0.009412	0.0	0.003137	0.003948	0.0	0.0
621	0.926207	0.369565	0.004235	0.0	0.001412	0.003740	0.0	0.0
622	0.926207	0.173913	0.005176	0.0	0.001725	0.003671	0.0	0.0
623	0.926207	0.326087	0.006000	0.0	0.002000	0.003463	0.0	0.0
624	0.926207	0.369565	0.008824	0.0	0.002941	0.003117	0.0	0.0
625	0.926207	0.326087	0.005529	0.0	0.001843	0.002563	0.0	0.0
626	0.926207	0.282609	0.002824	0.0	0.000941	0.000277	0.0	0.0
627	0.926207	0.369565	0.002353	0.0	0.000784	0.000000	0.0	0.0
628	0.926207	0.304348	0.000000	0.0	0.000000	0.000000	0.0	0.0

626 rows × 28 columns

SOFT VOTING

In [24]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


# split data into train+validation set and test set
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, random_s

# split train+validation set into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(X_trainval, y_trai

print("Size of training set: {}   size of validation set: {}   size of t
      " {}\n".format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]

best_score = 0
```

```
Size of training set: 351   size of validation set: 118   size of test set:
157
```

In [25]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons


```

In [26]:
```
log_clf = LogisticRegression(penalty = 'l1', C = 1)
log_clf.fit(X_trainval, y_trainval)
knn_clf = KNeighborsClassifier(3)
knn_clf.fit(X_trainval, y_trainval)


voting_clf = VotingClassifier(estimators=[('lr', log_clf), ('knn', knn_c
voting_clf.fit(X_trainval, y_trainval) # to find majority voting


from sklearn.metrics import accuracy_score
for clf in (log_clf, knn_clf, voting_clf):
    clf.fit(X_trainval, y_trainval)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))


from sklearn.metrics import precision_score
for clf in (log_clf, knn_clf, voting_clf):
    clf.fit(X_trainval, y_trainval)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, precision_score(y_test, y_pred))



from sklearn.metrics import recall_score
for clf in (log_clf, knn_clf, voting_clf):
    clf.fit(X_trainval, y_trainval)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, recall_score(y_test, y_pred))
```

```
LogisticRegression 1.0
KNeighborsClassifier 1.0
VotingClassifier 1.0
LogisticRegression 1.0
KNeighborsClassifier 1.0
VotingClassifier 1.0
LogisticRegression 1.0
KNeighborsClassifier 1.0
VotingClassifier 1.0
```

In [27]:
```
log_reg = LogisticRegression(penalty = 'l1', C = 1)
log_reg.fit(X_trainval, y_trainval)

print(log_reg.score(X_trainval, y_trainval))
print(log_reg.score(X_test, y_test))


logreg_tr_pred = log_reg.predict(X_trainval)
logreg_test_pred = log_reg.predict(X_test)
```

```
0.9957356076759062
1.0
```

HARD VOTING

In [28]:

```
svc_clf = SVC(C=1.0, gamma='auto')
log_clf.fit(X_trainval, y_trainval)
knn_clf = KNeighborsClassifier(3)
knn_clf.fit(X_trainval, y_trainval)


voting_clf = VotingClassifier(estimators=[('svc', svc_clf), ('knn', knn_
voting_clf.fit(X_trainval, y_trainval) # to find majority voting


from sklearn.metrics import accuracy_score
for clf in (svc_clf, knn_clf, voting_clf):
    clf.fit(X_trainval, y_trainval)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))


from sklearn.metrics import precision_score
for clf in (svc_clf, knn_clf, voting_clf):
    clf.fit(X_trainval, y_trainval)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, precision_score(y_test, y_pred))



from sklearn.metrics import recall_score
for clf in (svc_clf, knn_clf, voting_clf):
    clf.fit(X_trainval, y_trainval)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, recall_score(y_test, y_pred))
```

```
SVC 1.0
KNeighborsClassifier 1.0
VotingClassifier 1.0
SVC 1.0
KNeighborsClassifier 1.0
VotingClassifier 1.0
SVC 1.0
KNeighborsClassifier 1.0
VotingClassifier 1.0
```

BAGGING - LOGISTIC REGRESSION

In [30]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV


log_reg = LogisticRegression(penalty = 'l1', C = 1)
n_estimators_vals = [100, 200, 300, 400, 500]
max_samples_vals = [10, 50, 70, 100, 120, 150, 170, 200]


param_grid = dict(n_estimators=n_estimators_vals, max_samples = max_samp

log_bag = BaggingClassifier(log_reg,bootstrap = True, random_state=0)

grid_search = GridSearchCV(log_bag, param_grid = dict(n_estimators=n_est
grid_search.fit(X_trainval, y_trainval)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_scor
```

```
Best score on validation set: 0.00
Best parameters: {'max_samples': 170, 'n_estimators': 400}
Best cross-validation score: 0.99
```

In [31]:
```
log_reg = LogisticRegression(penalty = 'l1', C = 1)
bag_clf = BaggingClassifier(log_reg, n_estimators=400, max_samples=170,

bag_clf.fit(X_trainval, y_trainval)
y_pred = bag_clf.predict(X_test)
```

In [32]:
```

from sklearn.metrics import accuracy_score, precision_score, recall_scor
print(accuracy_score(y_test, y_pred))


print(precision_score(y_test, y_pred))


print(recall_score(y_test, y_pred))
```

```
0.9872611464968153
1.0
0.9736842105263158
```

In [33]:
```
bag_clf.fit(X_trainval, y_trainval)
print('Train score: {:.2f}'.format(bag_clf.score(X_trainval, y_trainval)
print('Test score: {:.2f}'.format(bag_clf.score(X_test, y_test)))
```

```
Train score: 0.99
Test score: 0.99
```

In [34]:
```
log_reg = LogisticRegression(penalty = 'l1', C = 1)
log_reg.fit(X_trainval, y_trainval)
y_pred_tree = log_reg.predict(X_test)

print(accuracy_score(y_test, y_pred_tree))

print(precision_score(y_test, y_pred_tree))

print(recall_score(y_test, y_pred_tree))
```

```
1.0
1.0
1.0
```

In [35]:
```
pd.crosstab(y_trainval, logreg_tr_pred)
```

Out[35]:

col_0	0	1
RISK		
0	241	0
1	2	226

In [36]:
```
print(log_reg.score(X_trainval, y_trainval))
```

```
0.9957356076759062
```

In [37]:
```
pd.crosstab(y_test, logreg_test_pred)
```

Out[37]:

col_0	0	1
RISK		
0	81	0
1	0	76

In [38]:
```
print(log_reg.score(X_test, y_test))
```

```
1.0
```

In [39]:

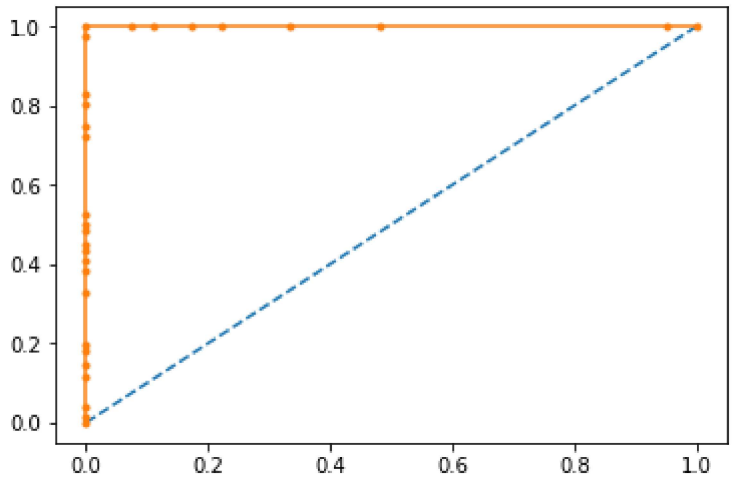
```
from sklearn.metrics import classification_report
report = classification_report(y_test, logreg_test_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        76

   micro avg       1.00      1.00      1.00       157
   macro avg       1.00      1.00      1.00       157
weighted avg       1.00      1.00      1.00       157
```

In [40]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

# predict probabilities
probs = log_reg.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
print( thresholds )
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

```
AUC: 1.000
[1.99999996 0.99999996 0.9999999  0.99999725 0.99999723 0.99999716
 0.99999687 0.99999682 0.99999677 0.99996298 0.99995922 0.99995718
 0.99995614 0.99994919 0.99994846 0.9982855  0.99816322 0.99274926
 0.99223491 0.88502964 0.70762461 0.04250934 0.04239167 0.04207386
 0.04121566 0.04082524 0.02866025 0.01440958 0.01320529]
```



K-NN

In [41]:
```
from sklearn.neighbors import KNeighborsClassifier

train_score_array = []
test_score_array = []


knn = KNeighborsClassifier(3)
knn.fit(X_train, y_train)
train_score_array.append(knn.score(X_trainval, y_trainval))
test_score_array.append(knn.score(X_test, y_test))
```

In [42]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV


knn_clf = KNeighborsClassifier(3)
n_estimators_vals = [100, 200, 300, 400, 500]
max_samples_vals = [10, 50, 70, 100, 120, 150, 170, 200]


param_grid = dict(n_estimators=n_estimators_vals, max_samples = max_samp

knn_bag = BaggingClassifier(knn_clf,bootstrap = True, random_state=0)

grid_search = GridSearchCV(knn_bag, param_grid = dict(n_estimators=n_est
grid_search.fit(X_trainval, y_trainval)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_scor
```

```
Best score on validation set: 0.00
Best parameters: {'max_samples': 170, 'n_estimators': 100}
Best cross-validation score: 1.00

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packag
es/sklearn/model_selection/_search.py:841: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and w
ill be removed in 0.24. This will change numeric results when test-set size
s are unequal.
  DeprecationWarning)
```

In [43]:
```
from sklearn.ensemble import BaggingClassifier


knn_clf = KNeighborsClassifier(3)
knn_bag_clf = BaggingClassifier(knn_clf, n_estimators=100, max_samples=1

knn_bag_clf.fit(X_trainval, y_trainval)
y_pred = bag_clf.predict(X_test)
```

In [44]:
```
from  sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))

from  sklearn.metrics import precision_score
print(precision_score(y_test, y_pred))

from  sklearn.metrics import recall_score
print(recall_score(y_test, y_pred))
```

```
0.9872611464968153
1.0
0.9736842105263158
```

In [45]:
```
knn_bag_clf.fit(X_trainval, y_trainval)
print('Train score: {:.2f}'.format(bag_clf.score(X_trainval, y_trainval)
print('Test score: {:.2f}'.format(bag_clf.score(X_test, y_test)))
```

```
Train score: 0.99
Test score: 0.99
```

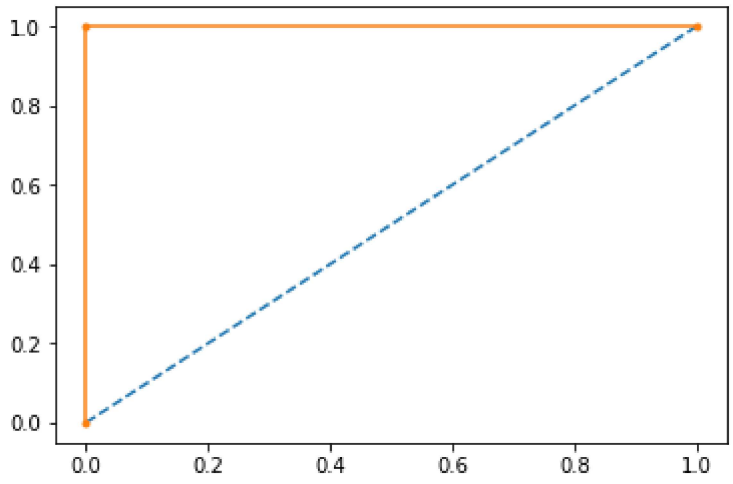
In [46]:
```


knnc_tr_pred = knn_bag_clf.predict(X_trainval)
knnc_test_pred = knn_bag_clf.predict(X_test)
print(knnc_tr_pred[4])

print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, knnc_tr_pred))
print("f1 score: ", f1_score(y_trainval, knnc_tr_pred))
print("recall score: ", recall_score(y_trainval, knnc_tr_pred))
print("precision: ", precision_score(y_trainval, knnc_tr_pred))
print("   ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, knnc_test_pred))
print("f1 score: ", f1_score(y_test, knnc_test_pred))
print("recall score: ", recall_score(y_test, knnc_test_pred))
print("precision: ", precision_score(y_test, knnc_test_pred))
```

```
1
Train data
Accuracy score:  0.997867803837953
f1 score:  0.9978021978021978
recall score:  0.9956140350877193
precision:  1.0

Test data
Accuracy score:  0.9936305732484076
f1 score:  0.9933774834437086
recall score:  0.9868421052631579
precision:  1.0
```

In [47]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

# fit a model
knn_clf.fit(X_trainval,y_trainval)
# predict probabilities
probs = knn_clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
print( thresholds )
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

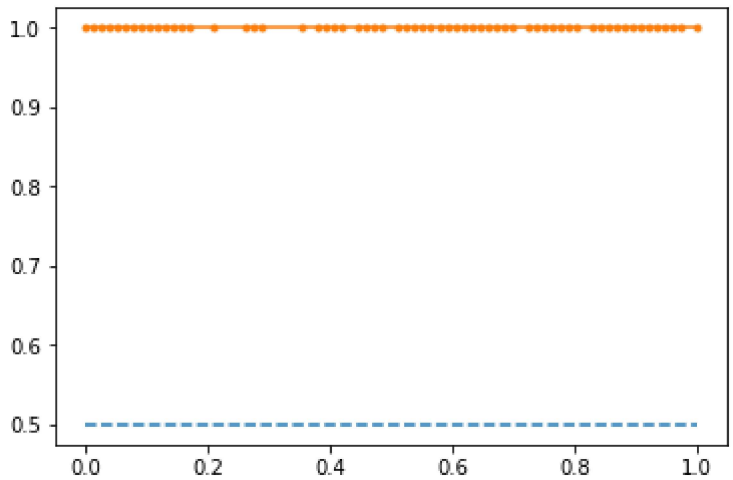
```
AUC: 1.000
[2. 1. 0.]
```



In [49]:

```
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve

# predict probabilities
probs = bag_clf.predict_proba(X_test)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# predict class values
yhat = bag_clf.predict(X_test)

# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)

# calculate F1 score
f1 = f1_score(y_test, yhat)

# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
pyplot.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(recall, precision, marker='.')
# show the plot
pyplot.show()
```

```
f1=0.987 auc=1.000 ap=1.000
```



PASTING - SVC LINEAR KERNEL

In [50]:
```
## svc linear kernel

from sklearn import svm
from sklearn.svm import SVC
```

In [51]:
```
svc_clf = SVC(C=1.0, gamma='auto')
n_estimators_vals = [100, 200, 300, 400, 500]
max_samples_vals = [10, 50, 70, 100, 120, 150, 170, 200]


param_grid = dict(n_estimators=n_estimators_vals, max_samples = max_samp

svc_bag = BaggingClassifier(svc_clf,bootstrap = False, random_state=0)

grid_search = GridSearchCV(svc_bag, param_grid = dict(n_estimators=n_est
grid_search.fit(X_trainval, y_trainval)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_scor
```

```
Best score on validation set: 0.00
Best parameters: {'max_samples': 200, 'n_estimators': 100}
Best cross-validation score: 0.98

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packag
es/sklearn/model_selection/_search.py:841: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and w
ill be removed in 0.24. This will change numeric results when test-set size
s are unequal.
  DeprecationWarning)
```

In [52]:
```
from sklearn.ensemble import BaggingClassifier


svc_clf = SVC(C=1.0, gamma='auto')
svc_bag_clf = BaggingClassifier(svc_clf, n_estimators=100, max_samples=2

svc_bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

In [53]:
```
from  sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))

from  sklearn.metrics import precision_score
print(precision_score(y_test, y_pred))

from  sklearn.metrics import recall_score
print(recall_score(y_test, y_pred))
```

```
0.9872611464968153
1.0
0.9736842105263158
```

In [54]:
```
svc_bag_clf.fit(X_trainval, y_trainval)
print('Train score: {:.2f}'.format(bag_clf.score(X_trainval, y_trainval)
print('Test score: {:.2f}'.format(bag_clf.score(X_test, y_test)))
```

```
Train score: 0.99
Test score: 0.99
```

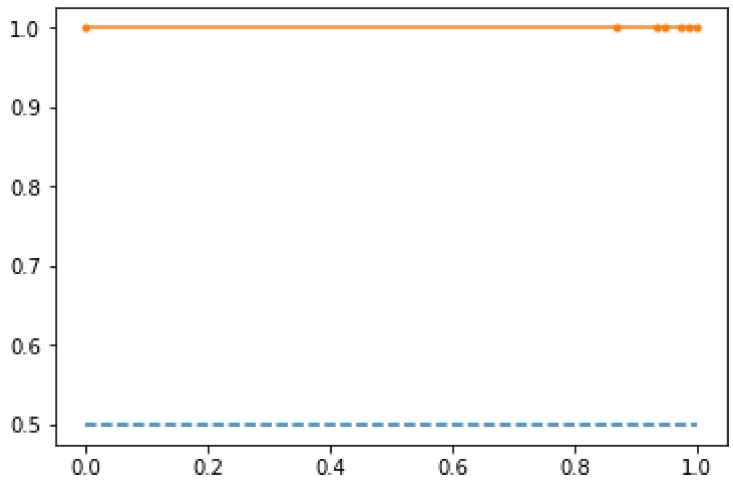
In [55]:
```

svc_tr_pred = svc_bag_clf.predict(X_trainval)
svc_test_pred = svc_bag_clf.predict(X_test)

print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, svc_tr_pred))
print("f1 score: ", f1_score(y_trainval, svc_tr_pred))
print("recall score: ", recall_score(y_trainval, svc_tr_pred))
print("precision: ", precision_score(y_trainval, svc_tr_pred))
print("   ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, svc_test_pred))
print("f1 score: ", f1_score(y_test, svc_test_pred))
print("recall score: ", recall_score(y_test, svc_test_pred))
print("precision: ", precision_score(y_test, svc_test_pred))
```

```
Train data
Accuracy score:  0.9829424307036247
f1 score:  0.9821428571428572
recall score:  0.9649122807017544
precision:  1.0

Test data
Accuracy score:  0.9745222929936306
f1 score:  0.972972972972973
recall score:  0.9473684210526315
precision:  1.0
```

In [56]:

```
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve

# predict probabilities
probs = svc_bag_clf.predict_proba(X_test)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# predict class values
yhat = svc_bag_clf.predict(X_test)

# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)

# calculate F1 score
f1 = f1_score(y_test, yhat)

# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
pyplot.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(recall, precision, marker='.')
# show the plot
pyplot.show()
```

```
f1=0.973 auc=1.000 ap=1.000
```



SVC - KERNEL RBF

In [57]:
```
## kernel rbf
svc_rbf_clf = SVC(kernel='rbf',C=1.0, gamma=0.5)
n_estimators_vals = [100, 200, 300, 400, 500]
max_samples_vals = [10, 50, 70, 100, 120, 150, 170, 200]


param_grid = dict(n_estimators=n_estimators_vals, max_samples = max_samp

svc_bag_clf = BaggingClassifier(svc_rbf_clf,bootstrap = False, random_st

grid_search = GridSearchCV(svc_bag_clf , param_grid = dict(n_estimators=
grid_search.fit(X_trainval, y_trainval)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_scor
```

```
Best score on validation set: 0.00
Best parameters: {'max_samples': 50, 'n_estimators': 100}
Best cross-validation score: 1.00
```

In [58]:
```
from sklearn.ensemble import BaggingClassifier


svc_rbf_clf = SVC(kernel='rbf',C=1.0, gamma=0.5)
svc_bag_clf = BaggingClassifier(svc_rbf_clf, n_estimators=100, max_sampl

svc_bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

In [59]:
```
from  sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))

from  sklearn.metrics import precision_score
print(precision_score(y_test, y_pred))

from  sklearn.metrics import recall_score
print(recall_score(y_test, y_pred))
```

```
0.9872611464968153
1.0
0.9736842105263158
```

In [60]:
```
svc_bag_clf.fit(X_trainval, y_trainval)
print('Train score: {:.2f}'.format(bag_clf.score(X_trainval, y_trainval)
print('Test score: {:.2f}'.format(bag_clf.score(X_test, y_test)))
```

```
Train score: 0.99
Test score: 0.99
```

In [61]:

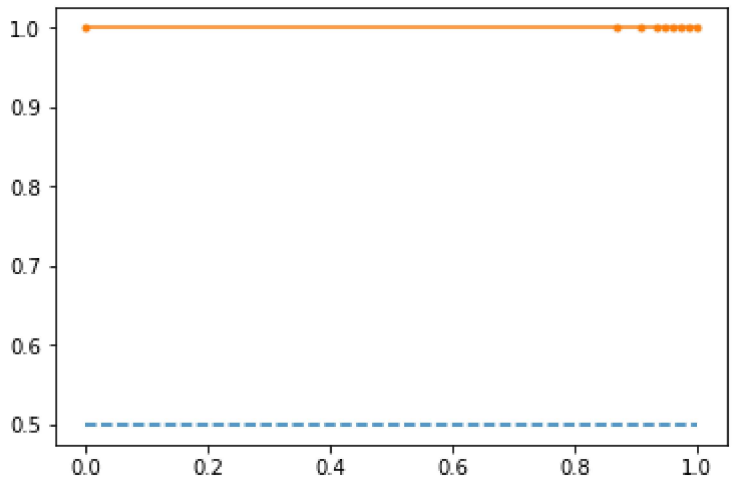
```
svc_tr_pred = svc_bag_clf.predict(X_trainval)
svc_test_pred = svc_bag_clf.predict(X_test)

print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, svc_tr_pred))
print("f1 score: ", f1_score(y_trainval, svc_tr_pred))
print("recall score: ", recall_score(y_trainval, svc_tr_pred))
print("precision: ", precision_score(y_trainval, svc_tr_pred))
print("   ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, svc_test_pred))
print("f1 score: ", f1_score(y_test, svc_test_pred))
print("recall score: ", recall_score(y_test, svc_test_pred))
print("precision: ", precision_score(y_test, svc_test_pred))
```

```
Train data
Accuracy score:  0.997867803837953
f1 score:  0.9978021978021978
recall score:  0.9956140350877193
precision:  1.0

Test data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0
```

In [62]:

```
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve

# predict probabilities
probs = svc_bag_clf.predict_proba(X_test)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# predict class values
yhat = svc_bag_clf.predict(X_test)

# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)

# calculate F1 score
f1 = f1_score(y_test, yhat)

# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
pyplot.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(recall, precision, marker='.')
# show the plot
pyplot.show()
```

```
f1=1.000 auc=1.000 ap=1.000
```



GRADIENT BOOSTING - DECISION TREE

In [63]:
```
## decision tree
from sklearn.tree import DecisionTreeClassifier
dt_clf = DecisionTreeClassifier(max_depth=4)
dt_clf.fit(X_trainval, y_trainval)
dt_pred = dt_clf.predict(X_trainval)
dt_test_pred = dt_clf.predict(X_test)
```

In [64]:
```
from sklearn.ensemble import GradientBoostingClassifier
gbrt = GradientBoostingClassifier(random_state=0, max_depth=4)
gbrt.fit(X_trainval, y_trainval)

print("Accuracy on training set: {:.3f}".format(gbrt.score(X_trainval, y
print("Accuracy on test set: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 1.000
```

In [65]:
```
gbrt.fit(X_trainval, y_trainval)

```

Out[65]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=4,
                           max_features=None, 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=100,
                           n_iter_no_change=None, presort='auto', random_state=0,
                           subsample=1.0, tol=0.0001, validation_fraction=0.1,
                           verbose=0, warm_start=False)
```

In [66]:
```
gbrt_tr_pred = gbrt.predict(X_trainval)
gbrt_test_pred = gbrt.predict(X_test)


```

In [68]:
```

gbrt_tr_pred = gbrt.predict(X_trainval)
gbrt_test_pred= gbrt.predict(X_test)

```

In [69]:
```
print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, dt_pred))
print("f1 score: ", f1_score(y_trainval, dt_pred))
print("recall score: ", recall_score(y_trainval, dt_pred))
print("precision: ", precision_score(y_trainval, dt_pred))
print("    ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, dt_test_pred))
print("f1 score: ", f1_score(y_test, dt_test_pred))
print("recall score: ", recall_score(y_test, dt_test_pred))
print("precision: ", precision_score(y_test, dt_test_pred))
```

```
Train data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0

Test data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0
```

ADABOOST CLASSIFIER - LINEAR SVC

In [70]:
```
## Linear svc
from sklearn.svm import LinearSVC

lsvc = LinearSVC(C=1)
```

In [71]:
```
from sklearn.ensemble import AdaBoostClassifier
lsvc = LinearSVC(C=1)

n_estimators_vals = [100, 200, 300, 400, 500]
learning_rate_vals = [0.01, 0.1, 0.3, 0.5, 1.0]


param_grid = dict(n_estimators=n_estimators_vals, learning_rate = learni
lsvc_ada = AdaBoostClassifier(lsvc, algorithm="SAMME",random_state=0)
grid_search = GridSearchCV(lsvc_ada, param_grid = dict(n_estimators=n_es
grid_search.fit(X_trainval, y_trainval)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_scor
```

```
Best score on validation set: 0.00
Best parameters: {'learning_rate': 0.01, 'n_estimators': 100}
Best cross-validation score: 1.00
```

In [72]:
```
ada_clf = AdaBoostClassifier(LinearSVC(C=1), n_estimators=100, algorithm
ada_clf.fit(X_trainval, y_trainval)


clf_tr_pred = ada_clf.predict(X_trainval)
clf_test_pred = ada_clf.predict(X_test)
```

In [73]:
```
print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, clf_tr_pred))
print("f1 score: ", f1_score(y_trainval, clf_tr_pred))
print("recall score: ", recall_score(y_trainval, clf_tr_pred))
print("precision: ", precision_score(y_trainval, clf_tr_pred))
print("   ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, clf_test_pred))
print("f1 score: ", f1_score(y_test, clf_test_pred))
print("recall score: ", recall_score(y_test, clf_test_pred))
print("precision: ", precision_score(y_test, clf_test_pred))
```

```
Train data
Accuracy score:  0.997867803837953
f1 score:  0.9978021978021978
recall score:  0.9956140350877193
precision:  1.0

Test data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0
```

In [74]:
```
pd.crosstab(y_trainval, clf_tr_pred)
```

Out[74]:

col_0	0	1
RISK		
0	241	0
1	1	227

In [75]:
```
pd.crosstab(y_test, clf_test_pred)
```

Out[75]:

col_0	0	1
RISK		
0	81	0
1	0	76

In [76]:
```
report = classification_report(y_test, clf_test_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        76

   micro avg       1.00      1.00      1.00       157
   macro avg       1.00      1.00      1.00       157
weighted avg       1.00      1.00      1.00       157
```

SVC KERNEL POLY- ADABOOST

In [78]:
```
## svc poly

poly_clf = SVC(kernel='poly',C=1.0, degree=1)
```

In [79]:
```
from sklearn.ensemble import AdaBoostClassifier
poly_clf = SVC(kernel='poly',C=1.0, degree=1)

n_estimators_vals = [100, 200, 300, 400, 500]
learning_rate_vals = [0.01, 0.1, 0.3, 0.5, 1.0]


param_grid = dict(n_estimators=n_estimators_vals, learning_rate = learni
poly_clf_ada = AdaBoostClassifier(poly_clf, algorithm="SAMME",random_sta
grid_search = GridSearchCV(poly_clf_ada , param_grid = dict(n_estimators
grid_search.fit(X_trainval, y_trainval)
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_scor
```

```
Best parameters: {'learning_rate': 0.5, 'n_estimators': 100}
Best cross-validation score: 0.58
```

In [80]:
```

poly_clf = SVC(kernel='poly',C=1.0, degree=1)
poly_ada_clf = AdaBoostClassifier(poly_clf, n_estimators=100, algorithm=
poly_ada_clf.fit(X_trainval, y_trainval)


clf_tr_pred = poly_ada_clf.predict(X_trainval)
clf_test_pred = poly_ada_clf.predict(X_test)
```

In [81]:
```
print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, clf_tr_pred))
print("f1 score: ", f1_score(y_trainval, clf_tr_pred))
print("recall score: ", recall_score(y_trainval, clf_tr_pred))
print("precision: ", precision_score(y_trainval, clf_tr_pred))
print("    ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, clf_test_pred))
print("f1 score: ", f1_score(y_test, clf_test_pred))
print("recall score: ", recall_score(y_test, clf_test_pred))
print("precision: ", precision_score(y_test, clf_test_pred))
```

```
Train data
Accuracy score:  0.5138592750533049
f1 score:  0.0
recall score:  0.0
precision:  0.0

Test data
Accuracy score:  0.5159235668789809
f1 score:  0.0
recall score:  0.0
precision:  0.0
```

DEEP NEURAL NETWORKS

In [82]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from keras.optimizers import SGD
```

```
Using TensorFlow backend.
```

In [83]:
```
#step 1: build model
model1 = Sequential()
#input layer
model1.add(Dense(10, input_dim = 28, activation = 'relu'))
#hidden layers
#output layer
model1.add(Dense(1, activation = 'sigmoid'))

#step 2: make computational graph - compile
opt = SGD(lr=0.01, momentum=0.9)
model1.compile(loss='binary_crossentropy', optimizer = 'adam', metrics=[

#step 3: train the model - fit
model1.fit(X_trainval, y_trainval, epochs = 10, batch_size = 400)
```

```
Epoch 1/10
469/469 [==============================] - 0s 405us/sample - loss: 0.7721 -
accuracy: 0.1620
Epoch 2/10
469/469 [==============================] - 0s 16us/sample - loss: 0.7652 -
accuracy: 0.1684
Epoch 3/10
469/469 [==============================] - 0s 16us/sample - loss: 0.7582 -
accuracy: 0.1791
Epoch 4/10
469/469 [==============================] - 0s 15us/sample - loss: 0.7516 -
accuracy: 0.1876
Epoch 5/10
469/469 [==============================] - 0s 14us/sample - loss: 0.7448 -
accuracy: 0.2068
Epoch 6/10
469/469 [==============================] - 0s 20us/sample - loss: 0.7383 -
accuracy: 0.2281
Epoch 7/10
469/469 [==============================] - 0s 17us/sample - loss: 0.7316 -
accuracy: 0.2495
Epoch 8/10
469/469 [==============================] - 0s 14us/sample - loss: 0.7250 -
accuracy: 0.2623
Epoch 9/10
469/469 [==============================] - 0s 14us/sample - loss: 0.7185 -
accuracy: 0.2857
Epoch 10/10
469/469 [==============================] - 0s 22us/sample - loss: 0.7120 -
accuracy: 0.3006
```

Out[83]: <tensorflow.python.keras.callbacks.History at 0x13bf3fe10>

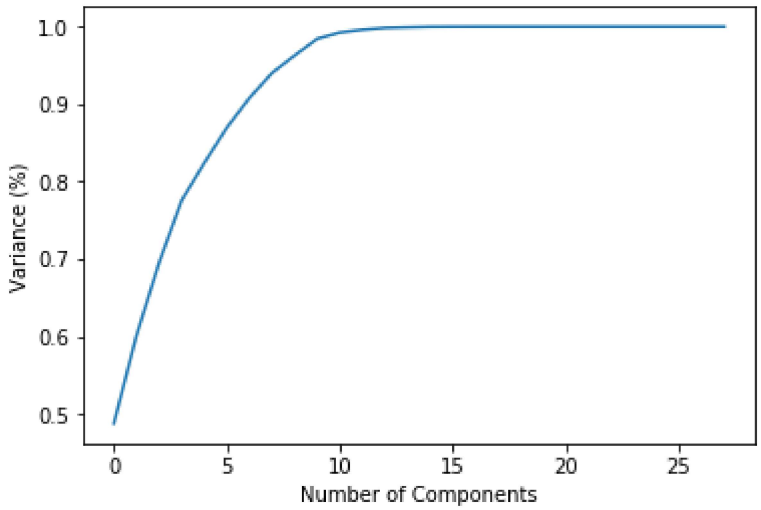
PCA

In [84]:
```
## PCA
from sklearn.decomposition import PCA



pca = PCA().fit(X)
```

In [85]:
```
# split data into train+validation set and test set
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV


# split data into train+validation set and test set
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, random_s

# split train+validation set into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(X_trainval, y_trai

print("Size of training set: {}   size of validation set: {}   size of t
      " {}\n".format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]

best_score = 0
```

```
Size of training set: 351   size of validation set: 118   size of test set:
157
```

In [86]:
```
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)')
plt.show()
```



In [87]:
```
pca = PCA(n_components=10)

X_trainval = pca.fit_transform(X_trainval)
X_test = pca.transform(X_test)
```

In [88]:
```
X_trainval.shape
```

Out[88]: (469, 10)

In [89]:
```
X_test.shape
```

Out[89]: (157, 10)

In [90]:
```
pca.explained_variance_
```

Out[90]:
```
array([0.65195423, 0.1617534 , 0.13172169, 0.115613  , 0.07155135,
       0.06024924, 0.05344001, 0.04211779, 0.03353733, 0.03023716])
```

In [91]:
```
pca.n_components_
```

Out[91]: 10

In [92]:
```
pca.explained_variance_ratio_
```

Out[92]:
```
array([0.47388799, 0.11757419, 0.09574495, 0.08403598, 0.05200875,
       0.04379355, 0.03884411, 0.03061429, 0.02437738, 0.02197858])
```

In [93]:
```
np.sum(pca.explained_variance_ratio_)
```

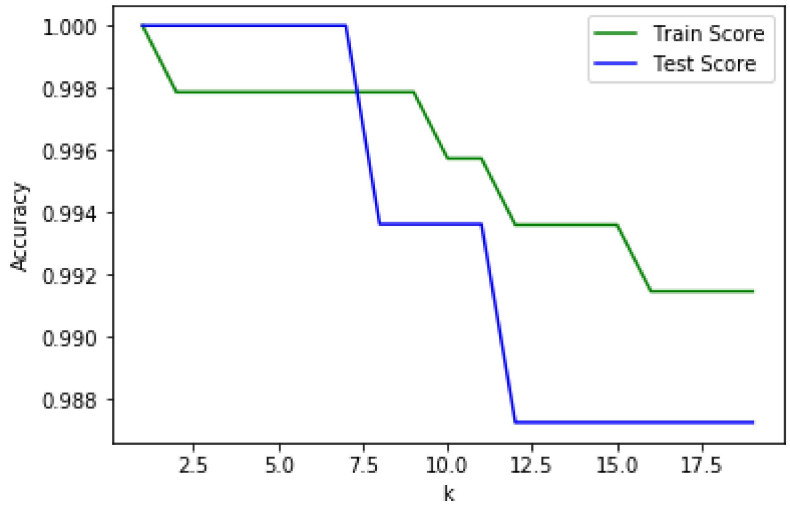
Out[93]: 0.982859760634868

K-NN AFTER PCA

In [94]:
```
from sklearn.neighbors import KNeighborsClassifier

train_score_array = []
test_score_array = []

for k in range(1,20):
    knn = KNeighborsClassifier(k)
    knn.fit(X_trainval, y_trainval)
    train_score_array.append(knn.score(X_trainval, y_trainval))
    test_score_array.append(knn.score(X_test, y_test))
```

In [95]:
```
x_axis = range(1,20)
%matplotlib inline
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend()
```

Out[95]: <matplotlib.legend.Legend at 0x13f6eec18>



In [96]:
```
from sklearn.model_selection import cross_val_score , GridSearchCV
from sklearn.neighbors import KNeighborsClassifier



knn = KNeighborsClassifier()


#param_grid = dict(k_range' : [1,3,5,7,9,12,15,17,20])
k_range = [1,3,5,7,9,12,15,17,20]
weights_range = ['uniform','distance']
param_grid = dict(n_neighbors=k_range, weights = weights_range)


#grid_search = GridSearchCV(knn, param_grid, cv=10, return_train_score=T
grid_search = GridSearchCV(knn, param_grid, cv=10, return_train_score=Tr
grid_search.fit(X_trainval, y_trainval)
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_scor
```

```
Best parameters: {'n_neighbors': 1, 'weights': 'uniform'}
Best cross-validation score: 1.00
```

In [97]:
```
from sklearn.neighbors import KNeighborsClassifier

train_score_array = []
test_score_array = []


knn = KNeighborsClassifier(3)
knn.fit(X_trainval, y_trainval)
train_score_array.append(knn.score(X_trainval, y_trainval))
test_score_array.append(knn.score(X_test, y_test))
```

In [98]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_scor
```

In [99]:
```
knn_c_bst_clf = KNeighborsClassifier(n_neighbors=1)

knn_c_bst_clf.fit(X_trainval,y_trainval)

knnc_tr_pred = knn_c_bst_clf.predict(X_trainval)
knnc_test_pred = knn_c_bst_clf.predict(X_test)
print(knnc_tr_pred[4])

print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, knnc_tr_pred))
print("f1 score: ", f1_score(y_trainval, knnc_tr_pred))
print("recall score: ", recall_score(y_trainval, knnc_tr_pred))
print("precision: ", precision_score(y_trainval, knnc_tr_pred))
print("   ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, knnc_test_pred))
print("f1 score: ", f1_score(y_test, knnc_test_pred))
print("recall score: ", recall_score(y_test, knnc_test_pred))
print("precision: ", precision_score(y_test, knnc_test_pred))
```

```
1
Train data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0

Test data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0
```

In [100]:
```
1 pd.crosstab(y_trainval, knnc_tr_pred)
```

Out[100]:

col_0	0	1
RISK		
0	241	0
1	0	228

In [101]:
```
1 pd.crosstab(y_test, knnc_test_pred)
```

Out[101]:

col_0	0	1
RISK		
0	81	0
1	0	76

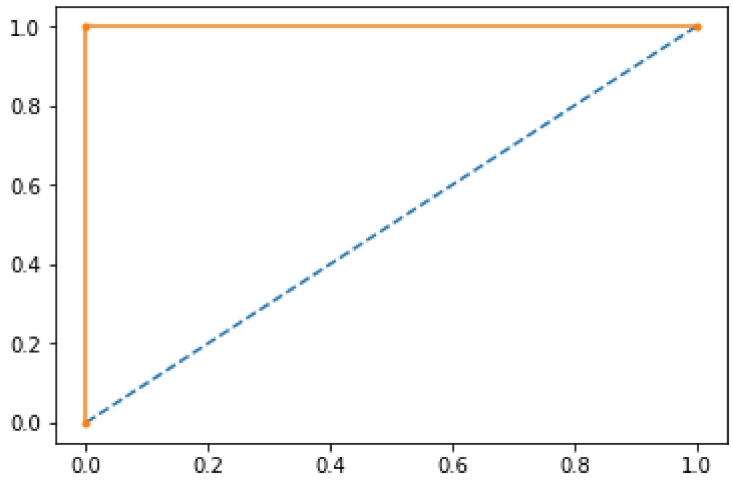
In [102]:
```
1 from sklearn.metrics import classification_report
2 report = classification_report(y_test, knnc_test_pred)
3 print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        76

   micro avg       1.00      1.00      1.00       157
   macro avg       1.00      1.00      1.00       157
weighted avg       1.00      1.00      1.00       157
```

In [103]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

# fit a model
knn_c_bst_clf.fit(X_trainval,y_trainval)
# predict probabilities
probs = knn_c_bst_clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
print( thresholds )
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

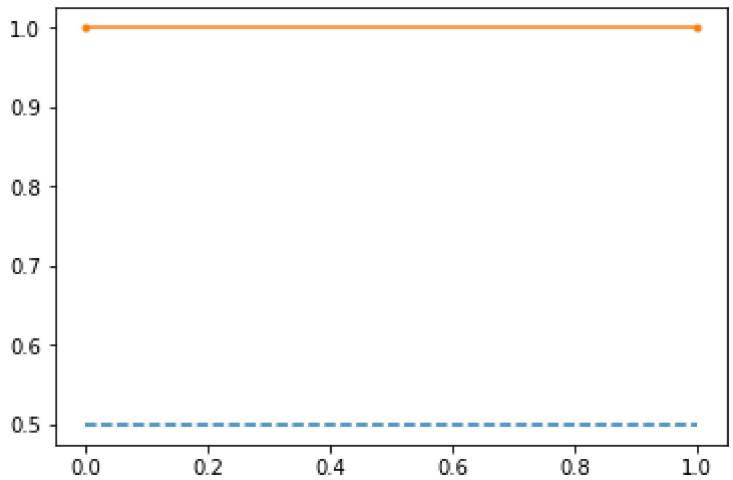
```
AUC: 1.000
[2. 1. 0.]
```



In [104]:

```
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve

# predict probabilities
probs = knn_c_bst_clf.predict_proba(X_test)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# predict class values
yhat = knn_c_bst_clf.predict(X_test)

# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)

# calculate F1 score
f1 = f1_score(y_test, yhat)

# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
pyplot.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(recall, precision, marker='.')
# show the plot
pyplot.show()
```

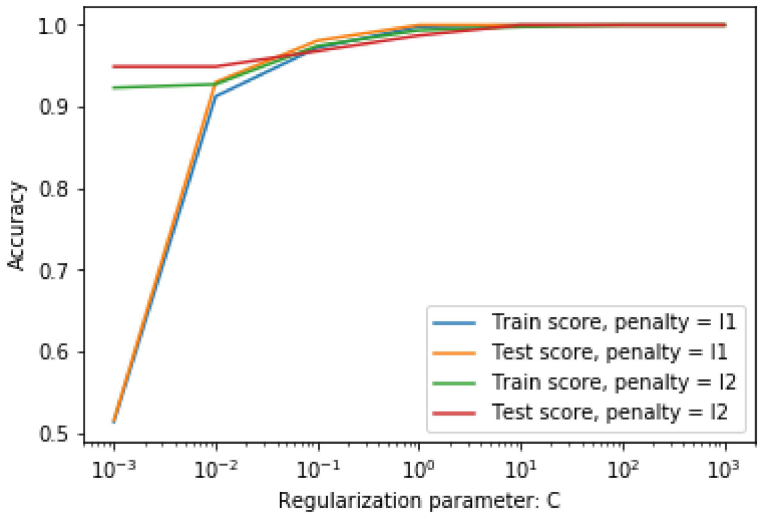
```
f1=1.000 auc=1.000 ap=1.000
```



LOGISTIC REGRESSION

In [105]:
```
from sklearn.linear_model import LogisticRegression

c_range = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
train_score_l1 = []
train_score_l2 = []
test_score_l1 = []
test_score_l2 = []

for c in c_range:
    log_l1 = LogisticRegression(penalty = 'l1', C = c)
    log_l2 = LogisticRegression(penalty = 'l2', C = c)
    log_l1.fit(X_trainval, y_trainval)
    log_l2.fit(X_trainval, y_trainval)
    train_score_l1.append(log_l1.score(X_trainval, y_trainval))
    train_score_l2.append(log_l2.score(X_trainval, y_trainval))
    test_score_l1.append(log_l1.score(X_test, y_test))
    test_score_l2.append(log_l2.score(X_test, y_test))
```

In [106]:
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(c_range, train_score_l1, label = 'Train score, penalty = l1')
plt.plot(c_range, test_score_l1, label = 'Test score, penalty = l1')
plt.plot(c_range, train_score_l2, label = 'Train score, penalty = l2')
plt.plot(c_range, test_score_l2, label = 'Test score, penalty = l2')
plt.legend()
plt.xlabel('Regularization parameter: C')
plt.ylabel('Accuracy')
plt.xscale('log')
```



In [107]:
```
from sklearn.linear_model import LogisticRegression

c_range = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
penalty_mod = ['l1','l2']

log_reg = LogisticRegression()

#create a parameter grid: map the parameter names to the values that sho
param_grid = dict(penalty=penalty_mod,C=c_range)
print(param_grid)

#instantiation of the grid
log_reg_grid = GridSearchCV(log_reg,param_grid, cv=10, scoring='accuracy

# fitting the grid
log_reg_grid.fit(X, y)
```

```
{'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packag
es/sklearn/model_selection/_search.py:841: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and w
ill be removed in 0.24. This will change numeric results when test-set size
s are unequal.
  DeprecationWarning)
```

Out[107]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, f
it_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10,
100, 1000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

In [108]:
```
scores = cross_val_score(log_reg, X, y,cv=10) # input arguments followed
print("Cross-validation scores: {}".format(scores))
```

```
Cross-validation scores: [1.         0.984375   1.         1.         1.
1.
 0.98387097 1.         1.         0.83870968]
```

In [109]:
```
log_reg = LogisticRegression(penalty = 'l2', C = 1)
log_reg.fit(X_trainval, y_trainval)

print(log_reg.score(X_trainval, y_trainval))
print(log_reg.score(X_test, y_test))


logreg_tr_pred = log_reg.predict(X_trainval)
logreg_test_pred = log_reg.predict(X_test)
```

```
0.9936034115138592
0.9872611464968153
```

In [110]:
```
pd.crosstab(y_trainval, logreg_tr_pred)
```

Out[110]:

col_0	0	1
RISK		
0	241	0
1	3	225

In [111]:
```
print(log_reg.score(X_trainval, y_trainval))
```

```
0.9936034115138592
```

In [112]:
```
pd.crosstab(y_test, logreg_test_pred)
```

Out[112]:

col_0	0	1
RISK		
0	81	0
1	2	74

In [113]:
```
print(log_reg.score(X_test, y_test))
```

```
0.9872611464968153
```

In [114]:
```
from sklearn.metrics import classification_report
report = classification_report(y_test, logreg_test_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        81
           1       1.00      0.97      0.99        76

   micro avg       0.99      0.99      0.99       157
   macro avg       0.99      0.99      0.99       157
weighted avg       0.99      0.99      0.99       157
```

In [115]:

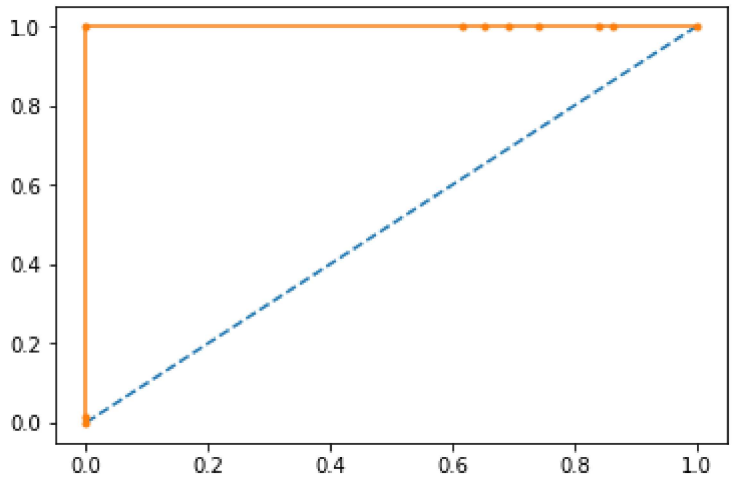
```
from sklearn.metrics import accuracy_score

print("Accuracy score: ", accuracy_score(y_trainval, logreg_tr_pred))
print("f1 score: ", f1_score(y_trainval, logreg_tr_pred))
print("recall score: ", recall_score(y_trainval, logreg_tr_pred))
print("precision: ", precision_score(y_trainval, logreg_tr_pred))
print("   ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, logreg_test_pred))
print("f1 score: ", f1_score(y_test, logreg_test_pred))
print("recall score: ", recall_score(y_test, logreg_test_pred))
print("precision: ", precision_score(y_test, logreg_test_pred))
```

```
Accuracy score:  0.9936034115138592
f1 score:  0.9933774834437086
recall score:  0.9868421052631579
precision:  1.0

Test data
Accuracy score:  0.9872611464968153
f1 score:  0.9866666666666666
recall score:  0.9736842105263158
precision:  1.0
```

In [116]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

# predict probabilities
probs = log_reg.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
print( thresholds )
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

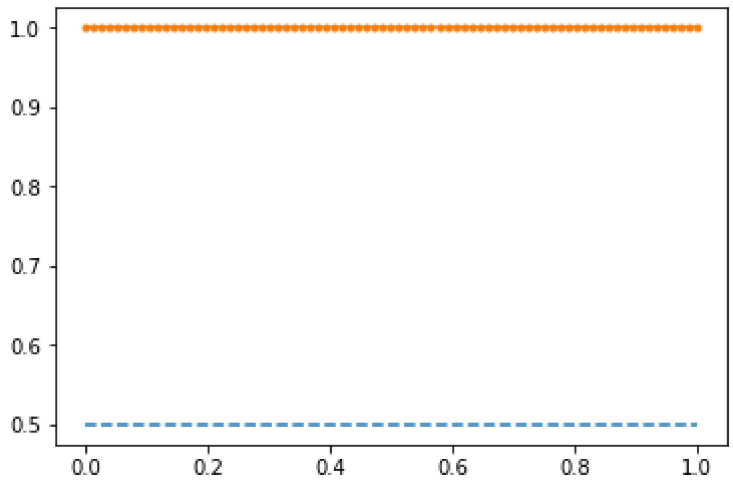
```
AUC: 1.000
[1.99999316 0.99999316 0.35667142 0.02871102 0.0286869  0.02832757
 0.02813376 0.02696169 0.02615027 0.02493556]
```



In [117]:

```
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve

# predict probabilities
probs = log_reg.predict_proba(X_test)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# predict class values
y_prd_class_val = log_reg.predict(X_test)

# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)

# calculate F1 score
f1 = f1_score(y_test, y_prd_class_val)

# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
pyplot.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(recall, precision, marker='.')
# show the plot
pyplot.show()
```

```
f1=0.987 auc=1.000 ap=1.000
```



LINEAR SVC

In [118]:

```
from sklearn.svm import LinearSVC
```

In [119]:
```
c_range= [0.001, 0.01, 0.1, 1, 10, 100]

param_grid = dict(C=c_range)
print("Parameter grid:\n{}".format(param_grid))
```

```
Parameter grid:
{'C': [0.001, 0.01, 0.1, 1, 10, 100]}
```

In [120]:
```
clf = LinearSVC()
linearsvc_grid_search = GridSearchCV(estimator=clf, param_grid = dict(C=
linearsvc_grid_search.fit(X, y)
```

Out[120]:
```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=LinearSVC(C=1.0, class_weight=None, dual=True, fit_interce
pt=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [121]:
```
linearsvc_grid_search.best_params_
```

Out[121]:
```
{'C': 1}
```

In [122]:
```
clf_best = LinearSVC(C=1)
```

In [123]:
```
clf_best.fit(X_trainval, y_trainval)

clf_tr_pred = clf_best.predict(X_trainval)
clf_test_pred = clf_best.predict(X_test)
```

In [124]:
```
print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, clf_tr_pred))
print("f1 score: ", f1_score(y_trainval, clf_tr_pred))
print("recall score: ", recall_score(y_trainval, clf_tr_pred))
print("precision: ", precision_score(y_trainval, clf_tr_pred))
print("    ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, clf_test_pred))
print("f1 score: ", f1_score(y_test, clf_test_pred))
print("recall score: ", recall_score(y_test, clf_test_pred))
print("precision: ", precision_score(y_test, clf_test_pred))
```

```
Train data
Accuracy score:  0.997867803837953
f1 score:  0.9978021978021978
recall score:  0.9956140350877193
precision:  1.0

Test data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0
```

In [125]:
```
pd.crosstab(y_trainval, clf_tr_pred)
```

Out[125]:

col_0	0	1
RISK		
0	241	0
1	1	227

In [126]:
```
pd.crosstab(y_test, clf_test_pred)
```

Out[126]:

col_0	0	1
RISK		
0	81	0
1	0	76

In [127]:
```
report = classification_report(y_test, clf_test_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        76

   micro avg       1.00      1.00      1.00       157
   macro avg       1.00      1.00      1.00       157
weighted avg       1.00      1.00      1.00       157
```

SVC LINEAR KERNEL

In [128]:
```
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score , GridSearchCV

c_range= [0.001, 0.01, 0.1, 1, 10, 100]

param_grid = dict(C=c_range)
print("Parameter grid:\n{}".format(param_grid))
```

```
Parameter grid:
{'C': [0.001, 0.01, 0.1, 1, 10, 100]}
```

In [129]:
```
svc = SVC(kernel='linear')
grid_search = GridSearchCV(estimator=svc, param_grid = dict(C=c_range) ,
grid_search.fit(X, y)
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packag
es/sklearn/model_selection/_search.py:841: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and w
ill be removed in 0.24. This will change numeric results when test-set size
s are unequal.
  DeprecationWarning)
```

Out[129]:
```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [130]:
```
grid_search.best_score_
```

Out[130]:
```
0.9984025559105432
```

In [131]:
```
grid_search.best_params_
```

Out[131]:
```
{'C': 10}
```

In [132]:
```
svc_best = SVC(C=10, gamma='auto',probability=True)
```

In [133]:
```
svc_best.fit(X_trainval, y_trainval)

svc_tr_pred = svc_best.predict(X_trainval)
svc_test_pred = svc_best.predict(X_test)
```

In [134]:
```
print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, svc_tr_pred))
print("f1 score: ", f1_score(y_trainval, svc_tr_pred))
print("recall score: ", recall_score(y_trainval, svc_tr_pred))
print("precision: ", precision_score(y_trainval, svc_tr_pred))
print("   ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, svc_test_pred))
print("f1 score: ", f1_score(y_test, svc_test_pred))
print("recall score: ", recall_score(y_test, svc_test_pred))
print("precision: ", precision_score(y_test, svc_test_pred))
```

```
Train data
Accuracy score:  0.997867803837953
f1 score:  0.9978021978021978
recall score:  0.9956140350877193
precision:  1.0

Test data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0
```

In [135]:
```
pd.crosstab(y_trainval, svc_tr_pred)
```

Out[135]:

col_0	0	1
RISK		
0	241	0
1	1	227

In [136]:
```
pd.crosstab(y_test, svc_test_pred)
```

Out[136]:

col_0	0	1
RISK		
0	81	0
1	0	76

In [137]:

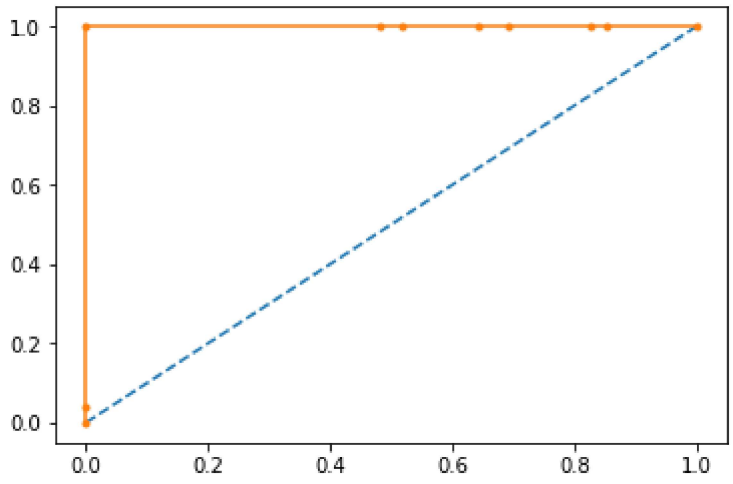
```
report = classification_report(y_test, svc_test_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        76

   micro avg       1.00      1.00      1.00       157
   macro avg       1.00      1.00      1.00       157
weighted avg       1.00      1.00      1.00       157
```

In [138]:

```
### ROC curve
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

# predict probabilities
probs = svc_best.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
print( thresholds )
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

```
AUC: 1.000
[2.         1.         0.94359095 0.01022384 0.01022288 0.00993117
 0.00992115 0.00968229 0.00967852 0.00834227]
```



SVC KERNEL RBF

In [139]:
```
#from mlxtend.plotting import plot_decision_regions
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score , GridSearchCV

c_range= [0.001, 0.01, 0.1, 1, 10, 100]
gamma_range=[0.001, 0.05,0.07,0.03,0.01,0.5,0.3, 0.1, 1, 10, 100]

param_grid = dict(C=c_range, gamma=gamma_range)
print("Parameter grid:\n{}".format(param_grid))
```

```
Parameter grid:
{'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.05, 0.07, 0.03, 0.
01, 0.5, 0.3, 0.1, 1, 10, 100]}
```

In [140]:
```
svc = SVC(kernel='rbf')
grid_search = GridSearchCV(estimator=svc, param_grid = dict(C=c_range,ga
grid_search.fit(X, y)
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packag
es/sklearn/model_selection/_search.py:841: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and w
ill be removed in 0.24. This will change numeric results when test-set size
s are unequal.
  DeprecationWarning)
```

Out[140]:
```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.
05, 0.07, 0.03, 0.01, 0.5, 0.3, 0.1, 1, 10, 100]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [141]:
```
grid_search.best_score_
```

Out[141]:
```
0.9984025559105432
```

In [142]:
```
grid_search.best_params_
```

Out[142]:
```
{'C': 0.1, 'gamma': 0.5}
```

In [143]:
```
svc_best_rbf = SVC(kernel='rbf',C=1.0, gamma=0.5)
```

In [144]:
```
svc_best_rbf.fit(X_trainval, y_trainval)

svc_rbf_tr_pred = svc_best_rbf.predict(X_trainval)
svc_rbf_test_pred = svc_best_rbf.predict(X_test)
```

In [145]:
```
print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, svc_rbf_tr_pred))
print("f1 score: ", f1_score(y_trainval, svc_rbf_tr_pred))
print("recall score: ", recall_score(y_trainval, svc_rbf_tr_pred))
print("precision: ", precision_score(y_trainval, svc_rbf_tr_pred))
print("    ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, svc_rbf_test_pred))
print("f1 score: ", f1_score(y_test, svc_rbf_test_pred))
print("recall score: ", recall_score(y_test, svc_rbf_test_pred))
print("precision: ", precision_score(y_test, svc_rbf_test_pred))
```

```
Train data
Accuracy score:  0.997867803837953
f1 score:  0.9978021978021978
recall score:  0.9956140350877193
precision:  1.0

Test data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0
```

In [146]:
```
pd.crosstab(y_trainval, svc_rbf_tr_pred)
```

Out[146]:

col_0	0	1
RISK		
0	241	0
1	1	227

In [147]:
```
pd.crosstab(y_test, svc_rbf_test_pred)
```

Out[147]:

col_0	0	1
RISK		
0	81	0
1	0	76

In [148]:
```
report = classification_report(y_test, svc_rbf_test_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        76

   micro avg       1.00      1.00      1.00       157
   macro avg       1.00      1.00      1.00       157
weighted avg       1.00      1.00      1.00       157
```

SVC KERNEL POLY

In [149]:
```
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score , GridSearchCV

c_range= [0.001, 0.01, 0.1, 1, 10, 100]
degree_range=[1,2,3,4]

param_grid = dict(C=c_range, degree = degree_range)
print("Parameter grid:\n{}".format(param_grid))
```

```
Parameter grid:
{'C': [0.001, 0.01, 0.1, 1, 10, 100], 'degree': [1, 2, 3, 4]}
```

In [150]:
```
svc = SVC(kernel='poly')
grid_search = GridSearchCV(estimator=svc, param_grid = dict(C=c_range,de
grid_search.fit(X, y)
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packag
es/sklearn/model_selection/_search.py:841: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and w
ill be removed in 0.24. This will change numeric results when test-set size
s are unequal.
  DeprecationWarning)
```

Out[150]:
```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='poly', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100], 'degree': [1, 2, 3,
4]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [151]:
```
grid_search.best_score_
```

Out[151]: 0.9984025559105432

In [152]:
```
grid_search.best_params_
```

Out[152]: {'C': 100, 'degree': 1}

In [153]:
```
svc_best_poly = SVC(kernel='poly',C=100, degree=1)
```

In [154]:
```
svc_best_poly.fit(X_trainval, y_trainval)

svc_poly_tr_pred = svc_best_poly.predict(X_trainval)
svc_poly_test_pred = svc_best_poly.predict(X_test)
```

In [155]:
```
print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, svc_poly_tr_pred))
print("f1 score: ", f1_score(y_trainval, svc_poly_tr_pred))
print("recall score: ", recall_score(y_trainval, svc_poly_tr_pred))
print("precision: ", precision_score(y_trainval, svc_poly_tr_pred))
print("    ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, svc_poly_test_pred))
print("f1 score: ", f1_score(y_test, svc_poly_test_pred))
print("recall score: ", recall_score(y_test, svc_poly_test_pred))
print("precision: ", precision_score(y_test, svc_poly_test_pred))
```

```
Train data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0

Test data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0
```

In [156]:
```
pd.crosstab(y_trainval, svc_poly_tr_pred)
```

Out[156]:

col_0	0	1
RISK		
0	241	0
1	0	228

In [157]:
```
1 pd.crosstab(y_test, svc_rbf_test_pred)
```

Out[157]:

col_0	0	1
RISK		
0	81	0
1	0	76

In [158]:
```
1 report = classification_report(y_test, svc_poly_test_pred)
2 print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        76

   micro avg       1.00      1.00      1.00       157
   macro avg       1.00      1.00      1.00       157
weighted avg       1.00      1.00      1.00       157
```

DECISION TREE

In [159]:
```
1 from sklearn.tree import DecisionTreeClassifier
2
3 dt = DecisionTreeClassifier()
4 param_grid = dict(max_depth=[4,6,8,10])
5
6 gs_dt = GridSearchCV(dt, param_grid=param_grid, cv=10, scoring='accuracy
7 gs_dt.fit(X_trainval, y_trainval)
```

Out[159]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gin
i', max_depth=None,
            max_features=None, 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, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': [4, 6, 8, 10]}, pre_dispatch='2*n_jobs',
       refit=True, return_train_score='warn', scoring='accuracy',
       verbose=0)
```

In [160]:
```
1 gs_dt.best_score_
```

Out[160]:
```
1.0
```

In [161]:
```
gs_dt.best_params_
```

Out[161]: {'max_depth': 4}

In [162]:
```
dt_best = DecisionTreeClassifier(max_depth=4)
dt_best.fit(X_trainval, y_trainval)
dt_pred = dt_best.predict(X_trainval)
dt_test_pred = dt_best.predict(X_test)
```

In [163]:
```
print("Train data")
print("Accuracy score: ", accuracy_score(y_trainval, dt_pred))
print("f1 score: ", f1_score(y_trainval, dt_pred))
print("recall score: ", recall_score(y_trainval, dt_pred))
print("precision: ", precision_score(y_trainval, dt_pred))
print("   ")
print("Test data")
print("Accuracy score: ", accuracy_score(y_test, dt_test_pred))
print("f1 score: ", f1_score(y_test, dt_test_pred))
print("recall score: ", recall_score(y_test, dt_test_pred))
print("precision: ", precision_score(y_test, dt_test_pred))
```

```
Train data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0

Test data
Accuracy score:  1.0
f1 score:  1.0
recall score:  1.0
precision:  1.0
```

In [164]:
```
pd.crosstab(y_trainval, dt_pred)
```

Out[164]:

col_0	0	1
RISK		
0	241	0
1	0	228

In [165]:
```
pd.crosstab(y_test, dt_test_pred)
```

Out[165]:

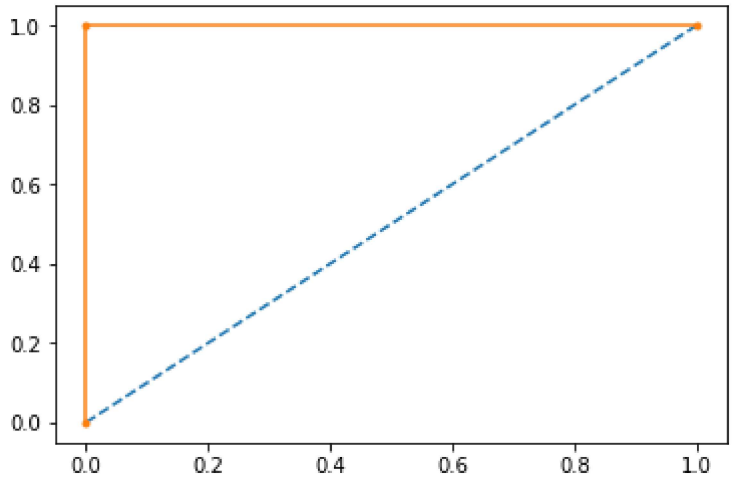
col_0	0	1
RISK		
0	81	0
1	0	76

In [166]:
```
report = classification_report(y_test, dt_test_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        76

   micro avg       1.00      1.00      1.00       157
   macro avg       1.00      1.00      1.00       157
weighted avg       1.00      1.00      1.00       157
```

In [167]:
```
### ROC curve
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

# predict probabilities
probs = dt_best.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % auc)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
print( thresholds )
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.')
# show the plot
pyplot.show()
```

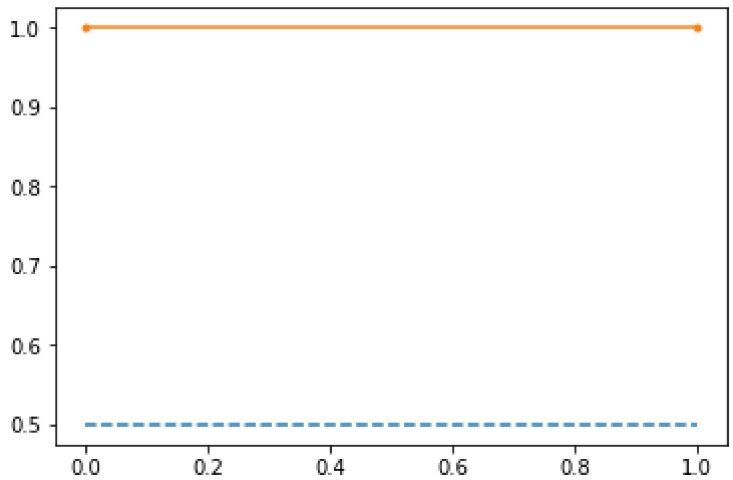
```
AUC: 1.000
[2. 1. 0.]
```



In [168]:

```
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve

# predict probabilities
probs = dt_best.predict_proba(X_test)

# keep probabilities for the positive outcome only
probs = probs[:, 1]

# predict class values
y_prd_class_val = dt_best.predict(X_test)

# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, probs)

# calculate F1 score
f1 = f1_score(y_test, y_prd_class_val)

# calculate precision-recall AUC
auc = auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y_test, probs)
print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))
# plot no skill
pyplot.plot([0, 1], [0.5, 0.5], linestyle='--')
# plot the precision-recall curve for the model
pyplot.plot(recall, precision, marker='.')
# show the plot
pyplot.show()
```

```
f1=1.000 auc=1.000 ap=1.000
```



Without PCA

In [8]:
```
# Without PCA
df={'MODEL':["Logistic","SVC Linear","SVC POLY","SVC RBF","KNN","Decisio
df1=pd.DataFrame(data=df)
df1
```

Out[8]:

	MODEL	train score	test score
0	Logistic	0.98	0.99
1	SVC Linear	0.96	0.94
2	SVC POLY	0.97	0.92
3	SVC RBF	0.95	0.95
4	KNN	0.96	0.90
5	Decision Tree	1.00	1.00

With PCA

In [9]:
```
# With PCA
df={'MODEL':["Logistic","SVC Linear","SVC POLY","SVC RBF","KNN","Decisio
df1=pd.DataFrame(data=df)
df1
```

Out[9]:

	MODEL	train score	test score
0	Logistic	1.00	1
1	SVC Linear	1.00	1
2	SVC POLY	1.00	1
3	SVC RBF	0.99	1
4	KNN	1.00	1
5	Decision Tree	1.00	1

We observe that there is an improvement using PCA but not neccesarily in every case

In [ ]:
```
1
```