資料探勘與機械學習理論報告

績優選股策略分析

國立高雄第一科技大學
財金學院金融系
指導老師：李宜熹 教授
學　　生：趙仁德 曾子瑋 蔡雍盛
日　　期：106年6月23日

大綱

壹、研究動機

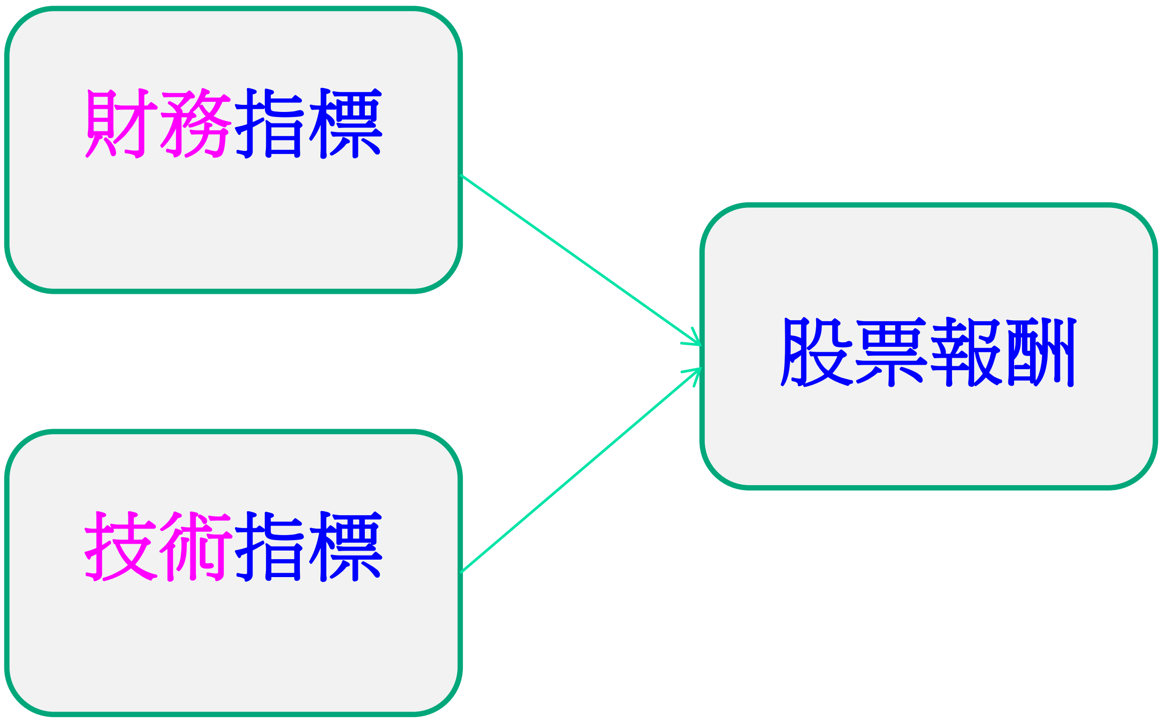
貳、研究架構

參、分析過程與結果

壹、研究動機

- 股票市場詭譎多變難預測，投資人對於各財務訊息與技術指標運用，有賴良好的預測模式。
- 本研究結合財務與技術指標，使用監督式分類分析演算法來建構預測模式，分析表現優於大盤與劣於大盤的特徵，做為選股趨吉避凶之依據。

貳、研究架構



財務指標

指標類別	指標名稱	指標代碼
獲利能力指標	ROE—綜合損益	A1
	營業利潤率	A2
	貝裡比率	A3
	營業資產報酬率	A4
成本費用率指標	研究發展費用率	B1
	現金流量比率	B2
	有息負債利率	B3
每股比率指標	每股淨值	C1
	常續性EPS	C2
成長率指標	營收成長率	D1
	總資產報酬成長率	D2
償債能力指標	速動比率	E1
	淨值/資產	E2
經營能力指標	總資產周轉天數	F1
相對價格指標	P/E	G1
	P/B	G2
	PSR	G3
	Tobins Q	G4

技術指標

常用技術指標	實質意義	備註
1. 均線(MA)	投資人的平均成本	5MA、10MA、20MA...
2. KD 指標	看出股價相對走勢	K>D，黃金交叉 →買進 K<D，死亡交叉 →賣出
3. RSI 指標	看出股價相對強弱	短週期>長週期 →買進 短週期<長週期 →賣出
4. MACD 指標	股價中長期波段走勢	快線 突破 慢線 →買進 快線 跌破 慢線 →賣出
5. 乖離率(BIAS)	投資人的平均報酬率	只能看短期波段訊號

參、分析過程與結果

（一）資料收集

（二）分析方法

（三）研究結果

資料收集

研究對象	研究期間	取得來源
財務指標	2015/4/3~ 2016/4/4 季資料	台灣經濟新報資料庫（**TEJ**）
股票價格		**YAHOO FINANCE**網站
技術指標		自行計算

資料來源：**2016**財管盃財務專題競賽 楊思達收集**(**只含財務指標**)**

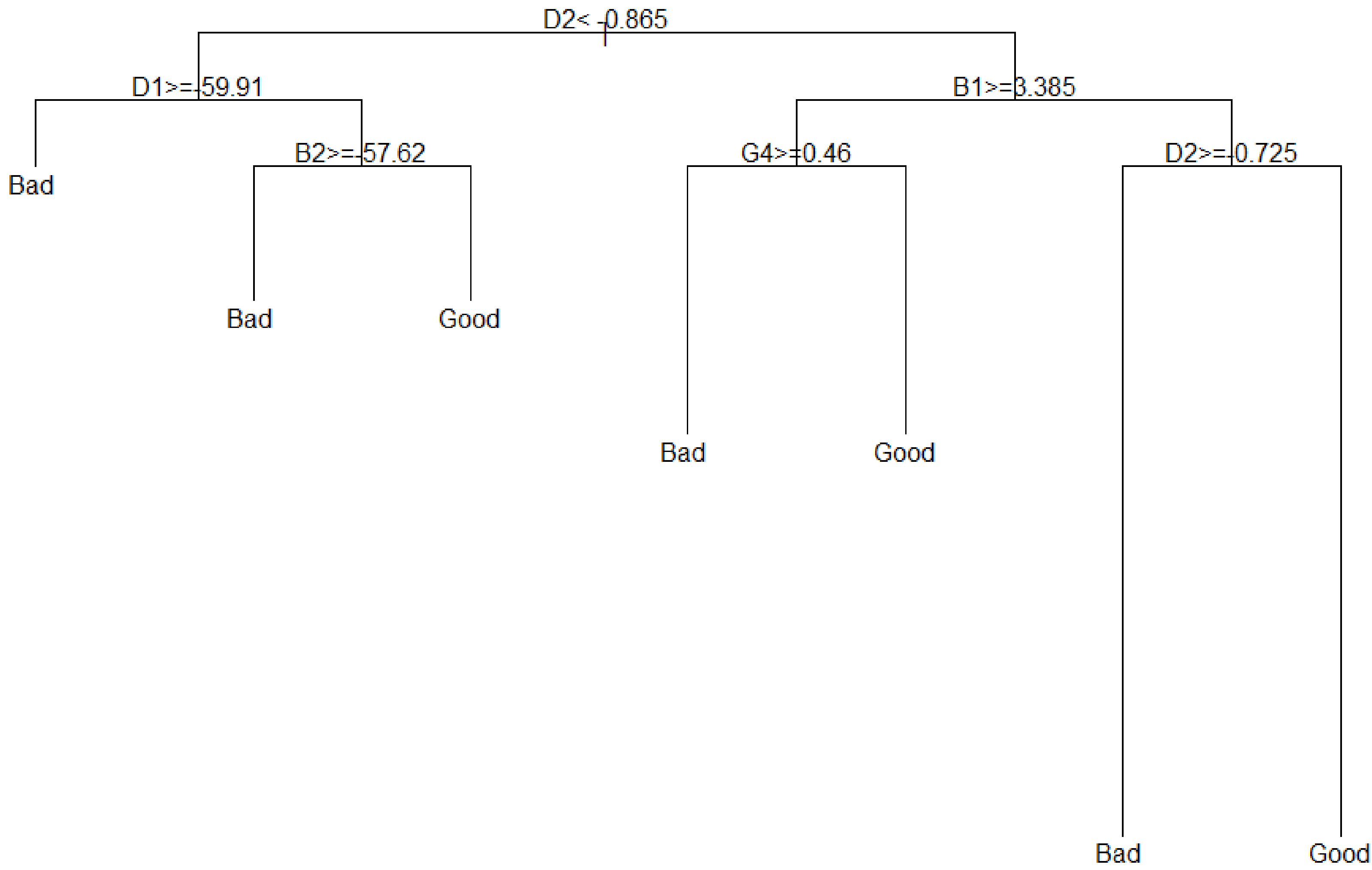
原始資料

id	A1	A2	A3	A4	B1	B2	B3	C1	C2	D1	D2	E1	E2	F1	G1	G2	G3	G4	X	winno	winrate	mark
3094	6.22	17.12	133.45	5.01	25.32	159.08	0.00	14.99	0.89	2.04	-0.09	1680.18	93.38	0.25	25.64	1.81	6.86	1.69	563	113	0.4593496	Bad
1338	10.96	11.16	187.68	11.30	2.46	49.67	3.20	77.15	9.19	-1.61	1.84	206.65	70.98	0.92	9.89	1.54	1.21	1.16	48	121	0.4918699	Bad
5234	13.86	8.99	152.89	9.47	9.33	37.81	1.77	20.96	2.90	1.42	-0.82	117.73	60.79	1.04	9.25	1.47	0.85	0.97	699	115	0.4674797	Bad
2328	-2.29	1.69	125.82	1.99	0.86	-3.66	1.46	19.33	0.65	10.20	0.70	158.80	60.27	0.99	8.66	0.69	0.36	0.46	271	106	0.4308943	Bad
1227	20.00	12.88	169.15	19.76	0.40	41.49	1.54	16.91	3.45	17.04	3.71	143.42	65.78	1.32	19.22	4.89	2.55	3.26	19	113	0.4593496	Bad
2027	1.31	0.47	104.46	0.55	0.00	12.07	2.26	14.69	-0.93	-0.26	-4.75	45.82	32.78	1.09	NA	1.00	0.22	0.85	221	118	0.4796748	Bad
1604	4.54	2.10	109.80	2.57	NA	9.94	1.60	14.98	1.07	-5.47	-1.25	91.72	61.95	0.80	10.45	0.85	0.64	0.62	141	121	0.4918699	Bad
2355	12.18	9.11	262.55	9.60	0.87	30.93	1.33	38.47	4.96	6.29	0.91	126.84	64.55	0.93	8.21	1.38	0.93	0.95	289	117	0.4756098	Bad
5871	18.02	24.11	166.48	4.42	NA	3.93	0.00	33.26	5.95	6.59	0.14	110.61	14.96	0.15	6.79	1.71	1.76	0.96	725	121	0.4918699	Bad
2436	-0.66	0.70	103.07	0.69	12.34	10.17	0.70	12.56	0.39	-5.22	-2.46	550.68	82.76	0.52	41.64	1.66	2.60	1.42	348	104	0.4227642	Bad
2880	10.04	NA	NA	NA	NA	NA	NA	15.45	1.42	0.70	0.04	NA	6.51	NA	9.33	0.99	3.21	0.10	482	112	0.4552846	Bad
1326	5.59	5.88	235.12	6.97	0.00	81.80	1.78	48.41	4.55	-17.96	3.21	199.88	64.93	0.63	12.40	1.53	1.32	1.10	46	130	0.5284553	Good
3231	5.56	0.38	108.84	0.87	2.15	7.88	2.20	28.45	0.76	5.22	-0.89	79.14	23.84	2.11	16.60	0.69	0.08	0.48	571	110	0.4471545	Bad
2373	12.93	5.36	116.90	6.87	NA	27.60	1.13	23.65	3.69	21.53	0.81	53.61	55.89	0.91	11.82	2.36	1.28	1.35	304	123	0.5000000	Good
1722	5.18	13.41	253.27	5.95	0.38	876.82	1.71	54.05	3.08	-0.13	-0.85	708.77	65.80	0.23	13.58	0.80	2.41	0.52	170	127	0.5162602	Good
6116	-7.33	-14.02	0.31	-7.12	4.56	29.42	2.37	10.09	-0.84	-27.43	-5.81	105.08	82.87	0.40	NA	0.39	0.73	0.38	733	105	0.4268293	Bad
6184	5.41	18.03	168.21	5.99	0.00	18.95	2.00	31.79	2.14	78.79	0.76	92.25	58.50	0.20	21.89	1.73	4.31	1.04	755	127	0.5162602	Good
6192	14.30	13.53	193.50	13.46	1.18	39.66	2.09	35.51	4.86	-9.90	-0.93	212.90	63.44	0.82	6.99	1.22	0.97	0.80	758	109	0.4430894	Bad
1521	24.55	7.72	182.51	14.31	2.78	30.63	2.30	23.45	5.90	15.50	3.12	86.51	48.89	1.62	12.86	3.83	1.20	1.87	112	110	0.4471545	Bad
4720	8.28	4.69	127.06	7.40	1.76	34.31	1.64	13.32	1.28	6.12	0.94	141.46	58.17	1.33	9.31	1.22	0.54	0.79	664	124	0.5040650	Good
4904	15.72	15.82	166.59	20.16	0.00	97.09	1.00	22.07	3.81	3.31	0.59	116.53	53.08	0.75	14.82	3.06	2.26	1.91	672	125	0.5081301	Good
8210	17.65	13.36	185.04	13.79	3.65	28.68	2.55	22.62	4.22	-0.75	-0.77	151.26	58.53	0.96	7.81	1.92	1.17	1.19	829	113	0.4593496	Bad
2489	13.96	-4.54	65.51	-7.55	3.38	16.74	2.34	17.90	-0.74	15.92	4.36	234.53	67.69	0.98	NA	0.91	0.64	0.60	391	129	0.5243902	Good
4934	6.16	3.65	183.41	4.68	1.04	34.52	3.68	16.86	0.98	19.40	2.76	112.97	66.76	1.03	22.25	1.29	0.88	0.98	680	123	0.5000000	Good
2852	2.33	5.00	122.50	NA	NA	NA	NA	16.29	0.65	3.06	-1.62	NA	34.87	0.37	13.76	0.71	0.69	0.25	478	125	0.5081301	Good

R Code 預測正確率75.6%

```
> #顯示訓練資料的正確率
> mark.traindata = iris$mark[-test.index]
> train.predict=factor(predict(iris.tree, iris.traindata,type='class'), le
vels=levels(mark.traindata))
>
> table.traindata =table(mark.traindata,train.predict)
> table.traindata
              train.predict
mark.traindata Bad Good
          Bad  408    0
          Good  61    8
> correct.traindata=sum(diag(table.traindata))/sum(table.traindata)*100
> correct.traindata
[1] 87.21174
>
> #顯示測試資料的正確率
> mark.testdata = iris$mark[test.index]
> test.predict=factor(predict(iris.tree, iris.testdata,
+                             type='class'), levels=levels(mark.testdata))

> table.testdata  =table(mark.testdata,test.predict)
> table.testdata
             test.predict
mark.testdata Bad Good
         Bad  154    4
         Good  46    1
> correct.testdata=sum(diag(table.testdata))/sum(table.testdata)*100
> correct.testdata
[1] 75.60976
```

R Code 產生之決策樹



分析方法

1. 類神經網路 (Neural Networks)
2. 邏輯斯迴歸 (Logistic regression)
3. 支持向量機器 (Support Vector Machine)
4. 決策樹 (Decision Trees)
5. 隨機森林 (Random Forest)
6. 最近鄰居 (K-Nearest Neighbors)

分析過程與結果

- 0.Prepare data for classification
- 1.PPN:Training a perceptron
- 2.LR:Logistic Regression Model
- 3.SVM:Support Vector Machines
- 4.TREE:Decision tree learning
  - Building a decision tree
  - Visualize a decision tree
- 5.FOREST:Random forests
  - Assessing feature importances with Random Forests
- 6.KNN:K nearest neighbors
  - Sequential feature selection algorithms:SBS
- 7.Model Evaluation and Hyperparameter Turning
  - Streamlining workflows with pipelines:P163
  - Using k fold cross validation to assess model performance:P170
  - Diagnosing bias and variance problems with learning curves:P171
  - Bagging:building an ensemble of classifiers from bootstrap samples
- 8.Pickle classification models
  - Pickle dump the classifiers
  - Pickle load the classifiers

0.Prepare data for classification

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from sklearn import datasets
import numpy as np

#iris = datasets.load_iris()
#X = iris.data[:, [2, 3]]
#y = iris.target

##############################################
### Step 1: Read data for stock forcasting ###
##############################################
import pandas as pd
mydata = pd.read_excel('D:/ipython/mystock.xls', head = 1)
mydata.columns = ['A1','A2','A3','A4','B1','B2','B3','C1','C2','D1','D2','E1','E2','F1','G1','G2','G3','G4','mark']
#mydata
mydata.shape
y_index = mydata.shape[1] - 1
y_index
#mydata.values
#X=mydata.values[:, [0,1,2,3,4,5,6,7,8,9,10,11,12]]
X=mydata.iloc[:, 0:y_index].values
X
y=mydata.iloc[:, y_index].values
#=mydata.values[:, [y_index]]
y
```

X and Y

```
array([[  -0.37,   10.33,  299.28, ...,    0.93,    1.08,    0.7 ],
       [   0.68,    6.09,  233.83, ...,    0.68,    1.39,    0.69],
       [ -29.74,   14.47,  -23.23, ...,    0.6 ,    2.61,    0.67],
       ...,
       [  18.71,   20.48,  331.4 , ...,    0.76,    0.71,    0.82],
       [  -5.67,   -5.4 ,  -12.57, ...,    0.58,    0.56,    0.71],
       [   7.16,    8.57,  311.59, ...,    0.79,    0.61,    0.74]])

array(['Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad',
       'Bad', 'Bad', 'Good', 'Bad', 'Good', 'Bad', 'Good', 'Bad', 'Good',
       'Bad', 'Good', 'Good', 'Bad', 'Good', 'Bad', 'Good', 'Good', 'Good',
       'Bad', 'Bad', 'Bad', 'Bad', 'Good', 'Bad', 'Bad', 'Bad', 'Bad',
       'Bad', 'Bad', 'Good', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad',
       'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad',
       'Bad', 'Good', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad',
       'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Good',
       'Bad', 'Good', 'Bad', 'Good', 'Bad', 'Bad', 'Bad', 'Bad', 'Good',
       'Bad', 'Bad', 'Bad', 'Bad', 'Bad', 'Good', 'Bad', 'Bad', 'Bad',
```

Splitting data into 70% training and 30% test data

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

if Version(sklearn_version) < '0.18':
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

#####################################################################
### Step 2: Splitting data into 70% training and 30% test data ###
#####################################################################
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0)
```

Standardizing the features

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

##################################################################
### Step 3: Standardizing the features                         ###
##################################################################
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

1.Perceptron test accuracy: 0.615

```
from sklearn.linear_model import Perceptron

ppn = Perceptron(n_iter=40, eta0=0.1, random_state=0)
ppn.fit(X_train_std, y_train)

y_train.shape
y_pred_train = ppn.predict(X_train_std)
print('Misclassified samples: %d' % (y_train != y_pred_train).sum())

y_test.shape
y_pred_test = ppn.predict(X_test_std)
print('Misclassified samples: %d' % (y_test != y_pred_test).sum())

print('==================================')
print('Training accuracy:', ppn.score(X_train, y_pred_train))
print('Test accuracy:', ppn.score(X_test, y_pred_test))
```

```
Perceptron(alpha=0.0001, class_weight=None, eta0=0.1, fit_intercept=True,
      n_iter=40, n_jobs=1, penalty=None, random_state=0, shuffle=True,
      verbose=0, warm_start=False)

(477,)

Misclassified samples: 149

(205,)

Misclassified samples: 72
==================================
Training accuracy: 0.626834381551
Test accuracy: 0.614634146341
```

2. Logistic regression accuracy: 0.8

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(C=1000.0, random_state=0)
lr.fit(X_train_std, y_train)

print('Training accuracy:', lr.score(X_train_std, y_train))
print('Test accuracy:', lr.score(X_test_std, y_test))
```

```
LogisticRegression(C=1000.0, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=0,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

Training accuracy: 0.842767295597
Test accuracy: 0.8
```

3.SVM test accuracy: 0.72

```
from sklearn.svm import SVC

svm = SVC(kernel='linear', C=1.0, random_state=0)
svm.fit(X_train_std, y_train)

print('Training accuracy:', svm.score(X_train_std, y_train))
print('Test accuracy:', svm.score(X_test, y_test))
```

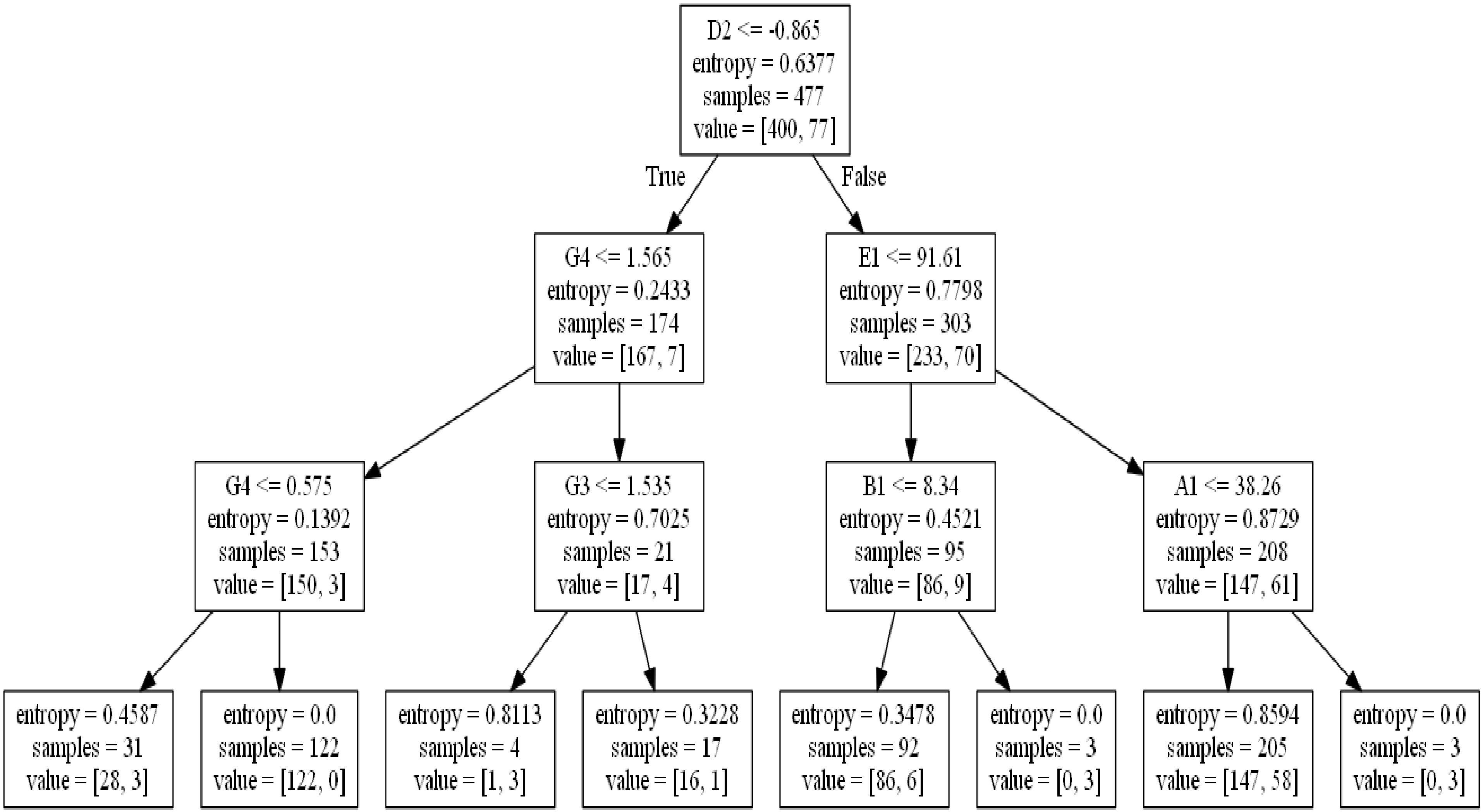
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)

Training accuracy: 0.840670859539
Test accuracy: 0.721951219512
```

4.Decision Tree test accuracy: 0.8

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
tree.fit(X_train, y_train)

print('Training accuracy:', tree.score(X_train, y_train))
print('Test accuracy:', tree.score(X_test, y_test))
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=0, splitter='best')

Training accuracy: 0.85534591195
Test accuracy: 0.8
```

Decision Tree

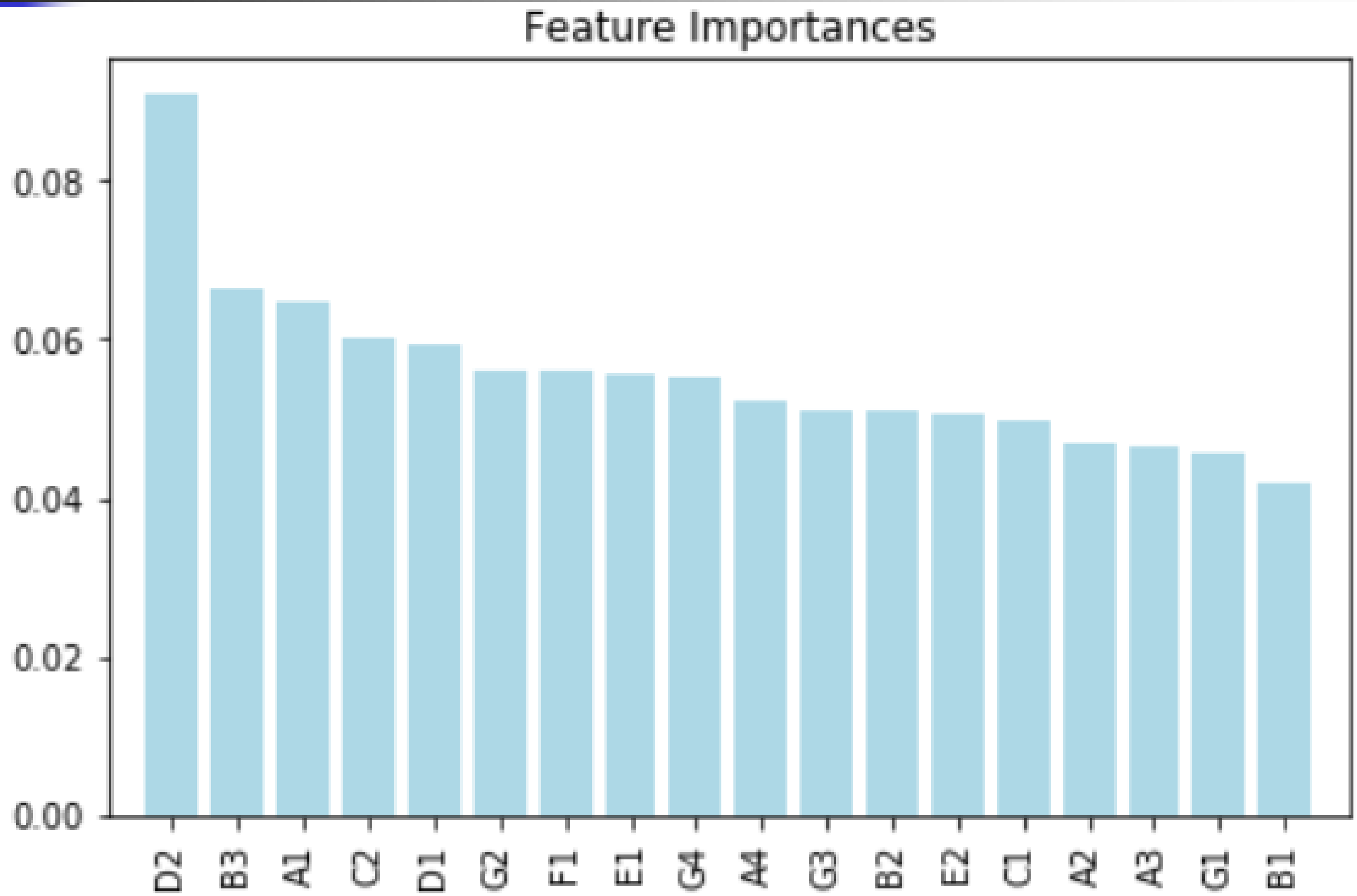


5.Random forests test accuracy: 0.809

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(criterion='entropy',
                                n_estimators=10,
                                random_state=1,
                                n_jobs=2)
forest.fit(X_train, y_train)
print('Training accuracy:', forest.score(X_train_std, y_train))
print('Test accuracy:', forest.score(X_test_std, y_test))
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=2, oob_score=False, random_state=1,
            verbose=0, warm_start=False)

Training accuracy: 0.830188679245
Test accuracy: 0.809756097561
```

Feature Importances with Random Forests

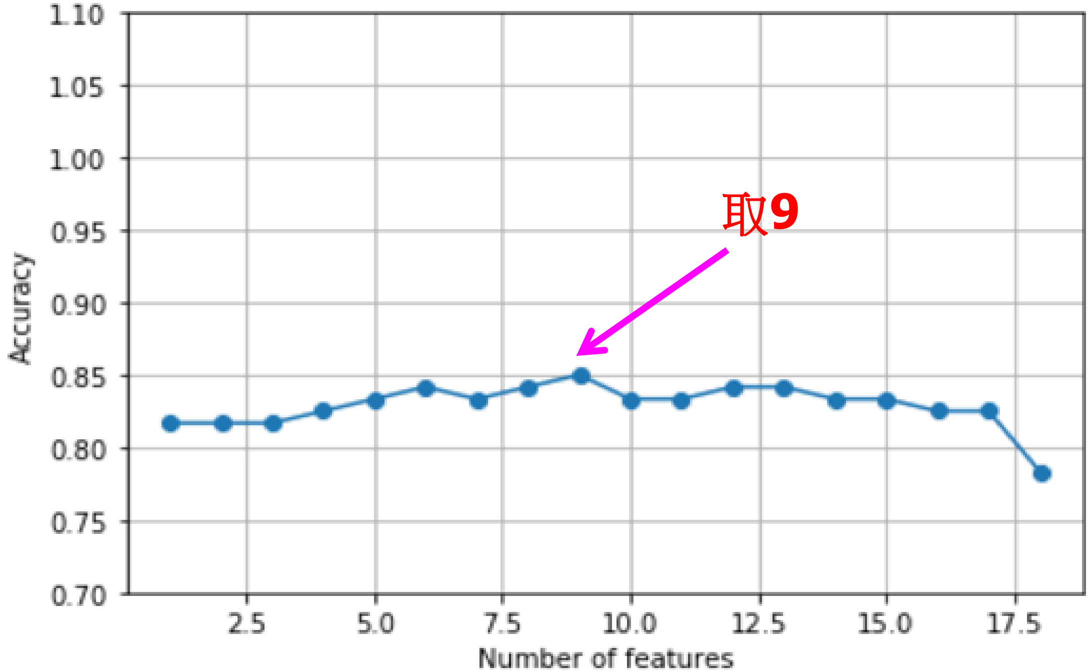


6.KNN test accuracy: 0.795

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=2, p=2, metric='minkowski')
knn.fit(X_train_std, y_train)
print('Training accuracy:', knn.score(X_train_std, y_train))
print('Test accuracy:', knn.score(X_test_std, y_test))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')

Training accuracy: 0.884696016771
Test accuracy: 0.79512195122
```

KNN：feature selection




```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

k9 = list(sbs.subsets_[9])
print(mydata.columns[0:][k9])
```

18 - 9

```
Index(['A1', 'A2', 'A3', 'A4', 'B2', 'B3', 'C1', 'D2', 'F1'], dtype='object')
```

Test accuracy: 0.795→0.8

```
knn = KNeighborsClassifier(n_neighbors=2, p=2, metric='minkowski')
knn.fit(X_train_std, y_train)
print('Training accuracy:', knn.score(X_train_std, y_train))
print('Test accuracy:', knn.score(X_test_std, y_test))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')

Training accuracy: 0.884696016771
Test accuracy: 0.79512195122
```

```
knn.fit(X_train_std[:, k9], y_train)
print('Training accuracy:', knn.score(X_train_std[:, k9], y_train))
print('Test accuracy:', knn.score(X_test_std[:, k9], y_test))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')

Training accuracy: 0.884696016771
Test accuracy: 0.8
```

7. Model Evaluation

- Streamlining workflows with pipelines:P163
- Using k fold cross validation to assess model performance:P170
- Diagnosing bias and variance problems with learning curves:P171
- Bagging: building an ensemble of classifiers from bootstrap samples

7.1 Streamlining workflows with pipelines

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline

pipe_tree = Pipeline([('scl', StandardScaler()),
                    ('clf', DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0))])

pipe_tree.fit(X_train, y_train)
print('Training Accuracy: %.3f' % pipe_tree.score(X_train, y_train))
print('Test accuracy:%.3f' % pipe_tree.score(X_test, y_test))
```

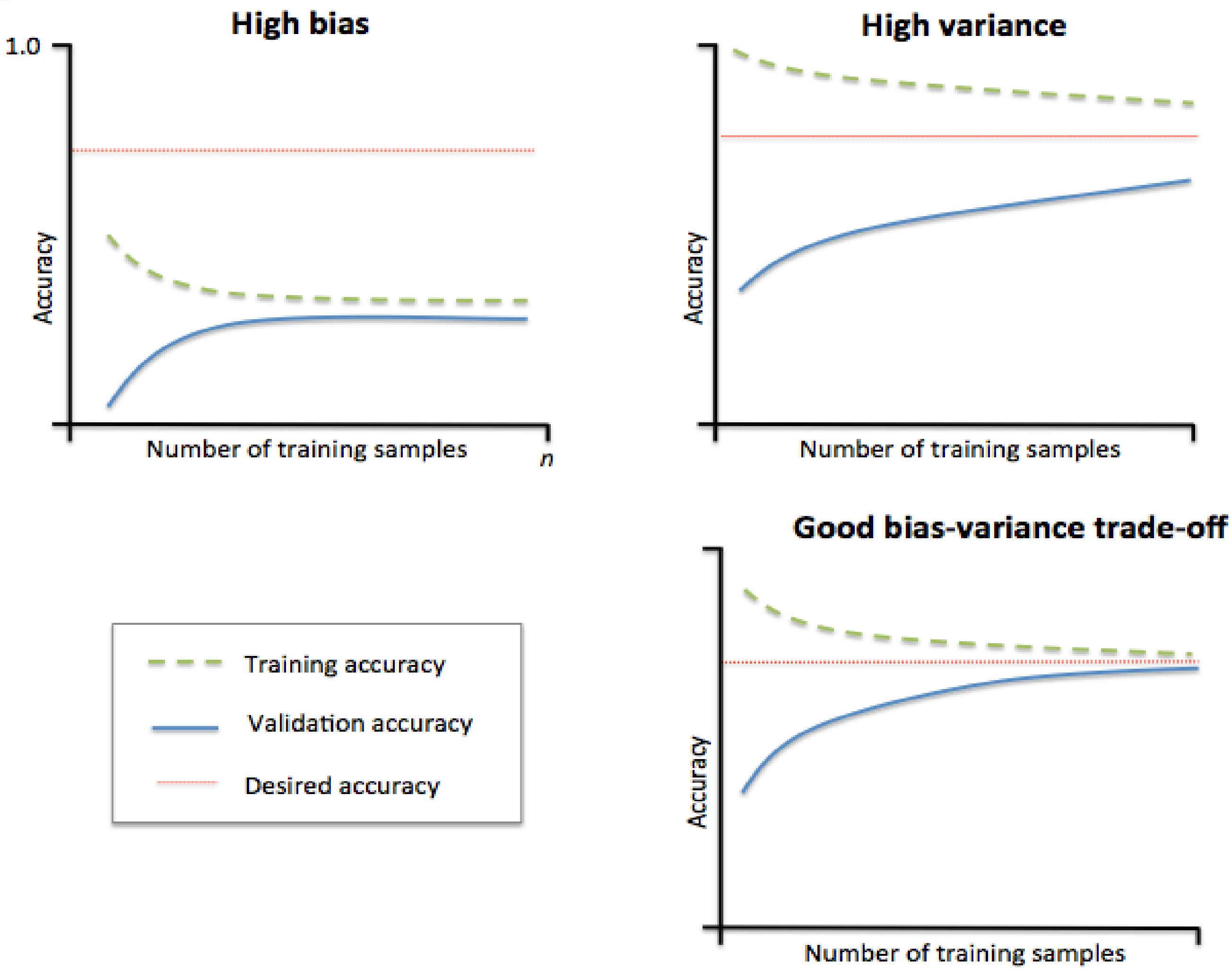
```
Pipeline(steps=[('scl', StandardScaler(copy=True, with_mean=True, with_std=True)), ('clf', DecisionTreeClassifier(class_weight=
None, criterion='entropy', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=0, splitter='best'))])

Training Accuracy: 0.855
Test accuracy:0.800
```

7.2 Using 10 fold cross validation

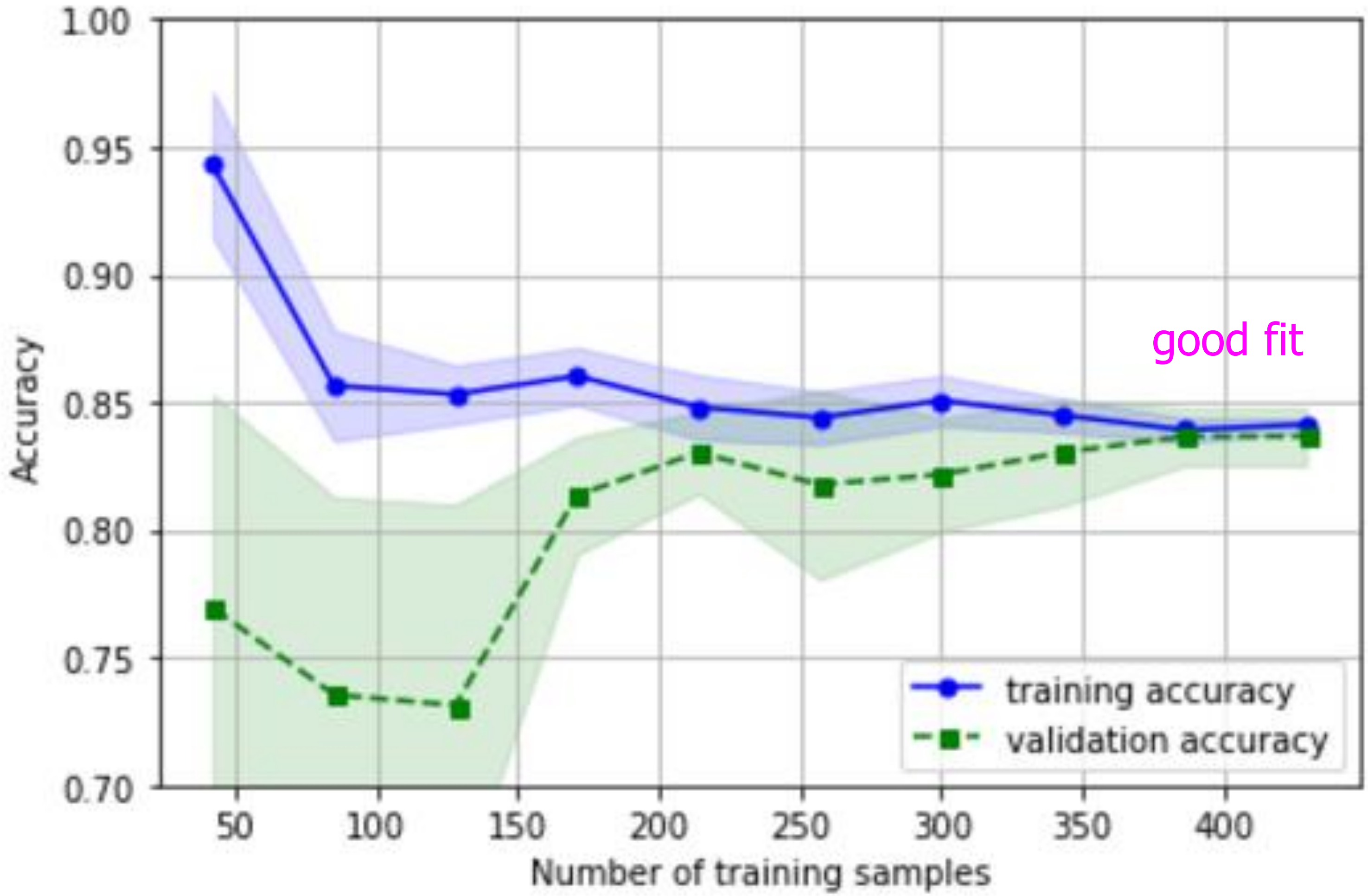
```
if Version(sklearn_version) < '0.18':
    from sklearn.cross_validation import cross_val_score
else:
    from sklearn.model_selection import cross_val_score

scores = cross_val_score(estimator=pipe_tree,
                         X=X_train,
                         y=y_train,
                         cv=10,
                         n_jobs=1)
print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

```
CV accuracy scores: [ 0.83333333  0.8125      0.83333333  0.83333333  0.83333333  0.83333333
  0.83333333  0.85106383  0.85106383  0.85106383]
CV accuracy: 0.837 +/- 0.011
```

7.3 Under/Over/Good fit ?



Diagnosing bias and variance problems with learning curves



7.4 Bagging: Building an ensemble of classifiers from bootstrap samples

```python
# max_depth=None
tree2 = DecisionTreeClassifier(criterion='entropy',
                               max_depth=None)

bag = BaggingClassifier(base_estimator=tree2,
                        n_estimators=500,
                        max_samples=1.0,
                        max_features=1.0,
                        bootstrap=True,
                        bootstrap_features=False,
                        n_jobs=1,
                        random_state=1)
```

```python
from sklearn.metrics import accuracy_score

tree2 = tree2.fit(X_train, y_train)
y_train_pred = tree2.predict(X_train)
y_test_pred = tree2.predict(X_test)

tree2_train = accuracy_score(y_train, y_train_pred)
tree2_test = accuracy_score(y_test, y_test_pred)
print('Decision tree2 train/test accuracies %.3f/%.3f'
      % (tree2_train, tree2_test))

bag = bag.fit(X_train, y_train)
y_train_pred = bag.predict(X_train)
y_test_pred = bag.predict(X_test)

bag_train = accuracy_score(y_train, y_train_pred)
bag_test = accuracy_score(y_test, y_test_pred)
print('Bagging train/test accuracies %.3f/%.3f'
      % (bag_train, bag_test))
```

```
Decision tree2 train/test accuracies 1.000/0.751
Bagging train/test accuracies 1.000/0.805
```

8. Pickle classification models

Pickle dump the classifiers

```
import pickle
import os
import re

dest = os.path.join('classifier', 'pkl_objects')
if not os.path.exists(dest):
    os.makedirs(dest)

pickle.dump(ppn, open(os.path.join(dest, 'ppn.pkl'), 'wb'), protocol=4)
pickle.dump(lr, open(os.path.join(dest, 'lr.pkl'), 'wb'), protocol=4)
pickle.dump(svm, open(os.path.join(dest, 'svm.pkl'), 'wb'), protocol=4)
pickle.dump(tree, open(os.path.join(dest, 'tree.pkl'), 'wb'), protocol=4)
pickle.dump(forest, open(os.path.join(dest, 'forest.pkl'), 'wb'), protocol=4)
pickle.dump(knn, open(os.path.join(dest, 'knn.pkl'), 'wb'), protocol=4)
```

Pickle load in the app.py

```
app = Flask(__name__)

###### Pickle load for Prediction
cur_dir = os.path.dirname(__file__)
ppn = pickle.load(open(os.path.join(cur_dir,
                 'pkl_objects',
                 'ppn.pkl'), 'rb'))
lr = pickle.load(open(os.path.join(cur_dir,
                 'pkl_objects',
                 'lr.pkl'), 'rb'))
svm = pickle.load(open(os.path.join(cur_dir,
                 'pkl_objects',
                 'svm.pkl'), 'rb'))
tree = pickle.load(open(os.path.join(cur_dir,
                 'pkl_objects',
                 'tree.pkl'), 'rb'))
forest = pickle.load(open(os.path.join(cur_dir,
                 'pkl_objects',
                'forest.pkl'), 'rb'))
knn = pickle.load(open(os.path.join(cur_dir,
                 'pkl_objects',
                 'knn.pkl'), 'rb'))
```

http://renderchao.pythonanywhere.com

輸入的參數：

A1:ROE—綜合損益: 38.26
A2:營業利潤率: 1.0
A3:貝裡比率: 1.0
A4:營業資產報酬率: 1.0
B1:研究發展費用率: 1.0
B2:現金流量比率: 1.0
B3:有息負債利率: 1.0
C1:每股淨值: 1.0
C2:常續性EPS: 1.0
D1:營收成長率: 1.0
D2:總資產報酬成長率: 1.0
E1:速動比率: 91.65
E2:淨值/資產: 1.0
F1:總資產周轉天數: 1.0
G1:P/E: 1.0
G2:P/B: 1.0
G3:PSR: 1.0
G4:Tobins Q: 1.0

股價勝大盤?

1.PPN: **Bad**
2.LR: **Bad**
3.SVM: **Bad**
4.TREE: **Bad**
5.FOREST: **Bad**
6.KNN: **Under construction**

重新查詢

輸入的參數：

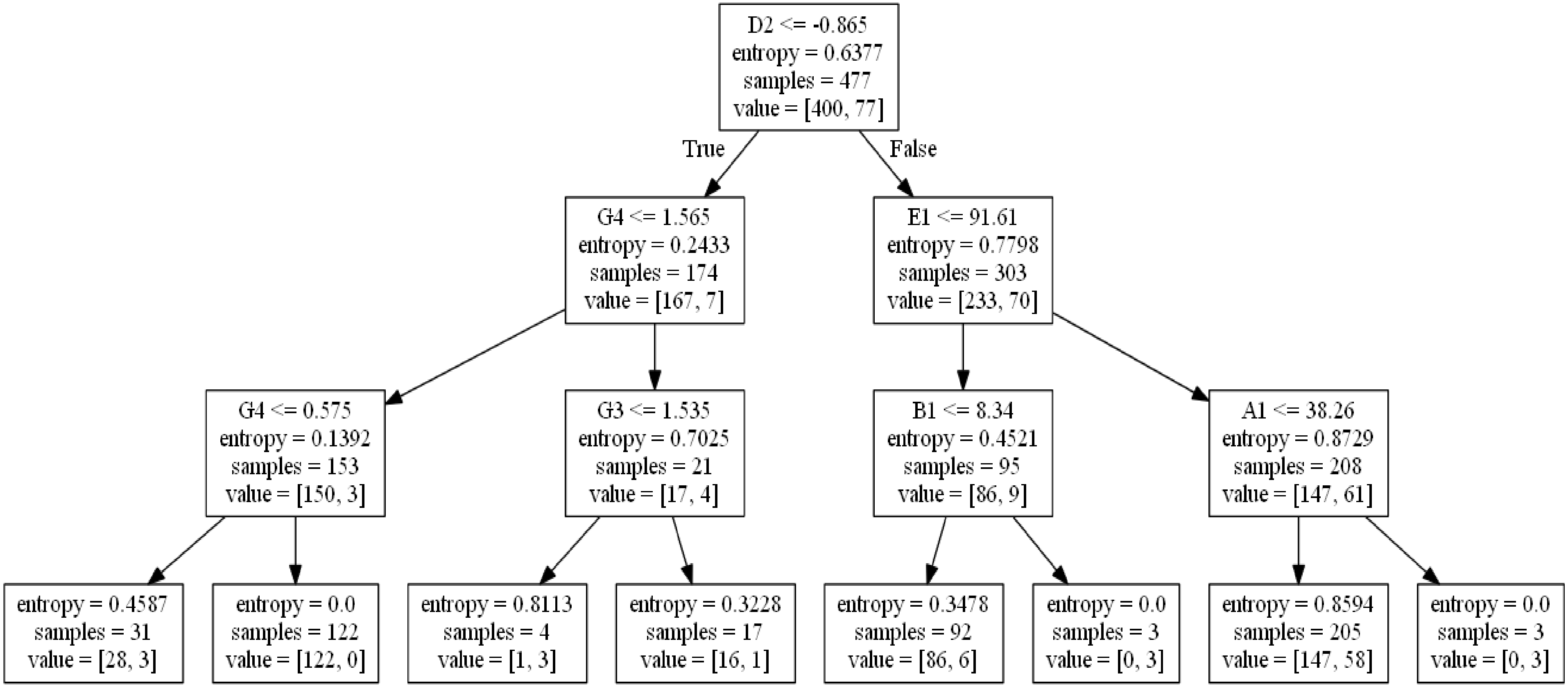
A1:ROE—綜合損益: 38.27
A2:營業利潤率: 1.0
A3:貝裡比率: 1.0
A4:營業資產報酬率: 1.0
B1:研究發展費用率: 1.0
B2:現金流量比率: 1.0
B3:有息負債利率: 1.0
C1:每股淨值: 1.0
C2:常續性EPS: 1.0
D1:營收成長率: 1.0
D2:總資產報酬成長率: 1.0
E1:速動比率: 91.65
E2:淨值/資產: 1.0
F1:總資產周轉天數: 1.0
G1:P/E: 1.0
G2:P/B: 1.0
G3:PSR: 1.0
G4:Tobins Q: 1.0

股價勝大盤?

1.PPN: **Bad**
2.LR: **Bad**
3.SVM: **Bad**
4.TREE: **Good**
5.FOREST: **Bad**
6.KNN: **Under construction**

重新查詢

決策樹分析結果



- 若指標 **D2 > -0.865, E1 > 91.61 , A1 >38.26** 時, 預測該股優於大盤 **(3家)**
- 若指標 **D2 <= -0.865, 0.575 < G4 <= 1.565** 時,預測該股劣於大盤 **(122家)**

D2:總資產報酬成長率 **E1**:速動比率 **A1:** ROE—綜合損益 **G4:** Tobins Q

財務指標

指標類別	指標名稱	指標代碼
獲利能力指標	ROE—綜合損益	A1
	營業利潤率	A2
	貝裡比率	A3
	營業資產報酬率	A4
成本費用率指標	研究發展費用率	B1
	現金流量比率	B2
	有息負債利率	B3
每股比率指標	每股淨值	C1
	常續性EPS	C2
成長率指標	營收成長率	D1
	總資產報酬成長率	D2
償債能力指標	速動比率	E1
	淨值/資產	E2
經營能力指標	總資產周轉天數	F1
相對價格指標	P/E	G1
	P/B	G2
	PSR	G3
	Tobins Q	G4

Q&A

感謝聆聽

歡迎提問與討論