Name: Ziyang Zhang

UNI: zz2732

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# scikit-learn
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as
LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
as QDA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import scale

# statsmodels
import statsmodels.api as sm
import statsmodels.formula.api as smf

%matplotlib inline
sns.set_style("darkgrid")
```

```
Bad key "text.kerning_factor" on line 4 in
/Users/Zhang/opt/anaconda3/lib/python3.7/site-packages/matplotlib/mpl-d
ata/stylelib/_classic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.templ
ate
or from the matplotlib source distribution
```

Question 2

Load data first:

In [106]:
```
df = pd.read_csv('Data/Weekly.csv')
df
```

Out[106]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
0	1990	0.816	1.572	-3.936	-0.229	-3.484	0.154976	-0.270	Down
1	1990	-0.270	0.816	1.572	-3.936	-0.229	0.148574	-2.576	Down
2	1990	-2.576	-0.270	0.816	1.572	-3.936	0.159837	3.514	Up
3	1990	3.514	-2.576	-0.270	0.816	1.572	0.161630	0.712	Up
4	1990	0.712	3.514	-2.576	-0.270	0.816	0.153728	1.178	Up
...	...	...	...	...	...	...	...	...	...
1084	2010	-0.861	0.043	-2.173	3.599	0.015	3.205160	2.969	Up
1085	2010	2.969	-0.861	0.043	-2.173	3.599	4.242568	1.281	Up
1086	2010	1.281	2.969	-0.861	0.043	-2.173	4.835082	0.283	Up
1087	2010	0.283	1.281	2.969	-0.861	0.043	4.454044	1.034	Up
1088	2010	1.034	0.283	1.281	2.969	-0.861	2.707105	0.069	Up

1089 rows × 9 columns

Replace the values of 'Direction' feature from string to integers.

In [107]:
```
df['Direction'] = df['Direction'].replace(['Down','Up'],[0,1])
df
```

Out[107]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
0	1990	0.816	1.572	-3.936	-0.229	-3.484	0.154976	-0.270	0
1	1990	-0.270	0.816	1.572	-3.936	-0.229	0.148574	-2.576	0
2	1990	-2.576	-0.270	0.816	1.572	-3.936	0.159837	3.514	1
3	1990	3.514	-2.576	-0.270	0.816	1.572	0.161630	0.712	1
4	1990	0.712	3.514	-2.576	-0.270	0.816	0.153728	1.178	1
...	...	...	...	...	...	...	...	...	...
1084	2010	-0.861	0.043	-2.173	3.599	0.015	3.205160	2.969	1
1085	2010	2.969	-0.861	0.043	-2.173	3.599	4.242568	1.281	1
1086	2010	1.281	2.969	-0.861	0.043	-2.173	4.835082	0.283	1
1087	2010	0.283	1.281	2.969	-0.861	0.043	4.454044	1.034	1
1088	2010	1.034	0.283	1.281	2.969	-0.861	2.707105	0.069	1

1089 rows × 9 columns

Part (a)

Numerical summaries:

In [108]:
```
# Descriptive Stats
df.describe()
```

Out[108]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volun
count	1089.000000	1089.000000	1089.000000	1089.000000	1089.000000	1089.000000	1089.0000
mean	2000.048669	0.150585	0.151079	0.147205	0.145818	0.139893	1.5746
std	6.033182	2.357013	2.357254	2.360502	2.360279	2.361285	1.6866
min	1990.000000	-18.195000	-18.195000	-18.195000	-18.195000	-18.195000	0.0874
25%	1995.000000	-1.154000	-1.154000	-1.158000	-1.158000	-1.166000	0.3320
50%	2000.000000	0.241000	0.241000	0.241000	0.238000	0.234000	1.0026
75%	2005.000000	1.405000	1.409000	1.409000	1.409000	1.405000	2.0537
max	2010.000000	12.026000	12.026000	12.026000	12.026000	12.026000	9.3282

In [109]:
```
# Correlation Matrix
df.corr()
```

Out[109]:

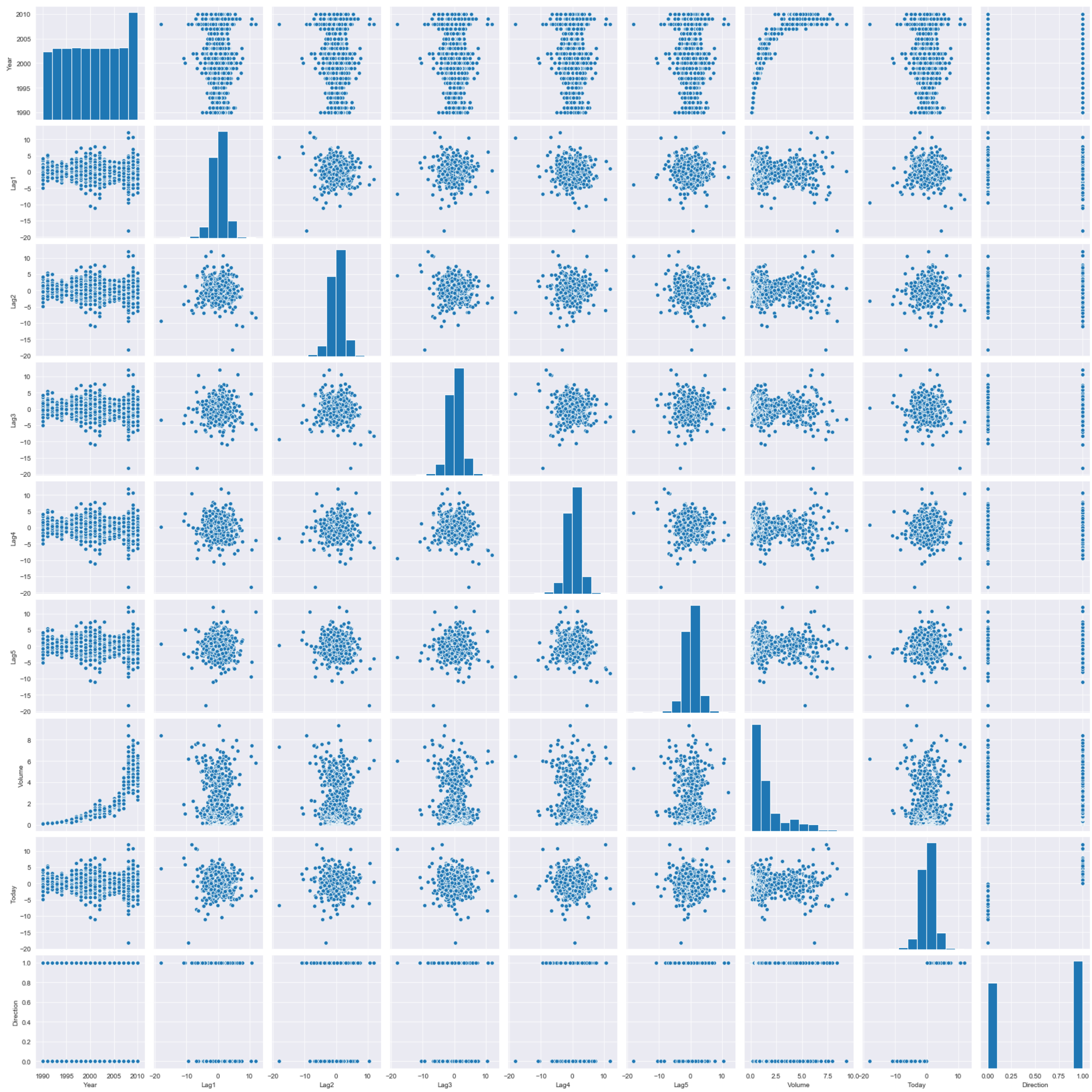
	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today
Year	1.000000	-0.032289	-0.033390	-0.030006	-0.031128	-0.030519	0.841942	-0.032460
Lag1	-0.032289	1.000000	-0.074853	0.058636	-0.071274	-0.008183	-0.064951	-0.075032
Lag2	-0.033390	-0.074853	1.000000	-0.075721	0.058382	-0.072499	-0.085513	0.059167
Lag3	-0.030006	0.058636	-0.075721	1.000000	-0.075396	0.060657	-0.069288	-0.071244
Lag4	-0.031128	-0.071274	0.058382	-0.075396	1.000000	-0.075675	-0.061075	-0.007826
Lag5	-0.030519	-0.008183	-0.072499	0.060657	-0.075675	1.000000	-0.058517	0.011013
Volume	0.841942	-0.064951	-0.085513	-0.069288	-0.061075	-0.058517	1.000000	-0.033078
Today	-0.032460	-0.075032	0.059167	-0.071244	-0.007826	0.011013	-0.033078	1.000000
Direction	-0.022200	-0.050004	0.072696	-0.022913	-0.020549	-0.018168	-0.017995	0.720025

'Year' and 'Volume is positively correlated with a high correlation coefficient of 0.84

Graphical Summaries:
Pairplot: check patterns for all possible pairs of features.

In [110]:
```
sns.pairplot(df)
```

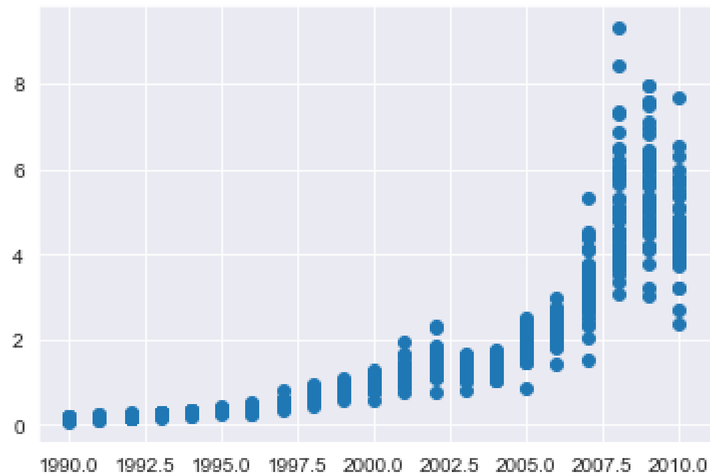
Out[110]: <seaborn.axisgrid.PairGrid at 0x7ff4e8b92910>



Check specific pairs of features:

In [115]:
```
plt.scatter(df['Year'],df['Volume'])
```

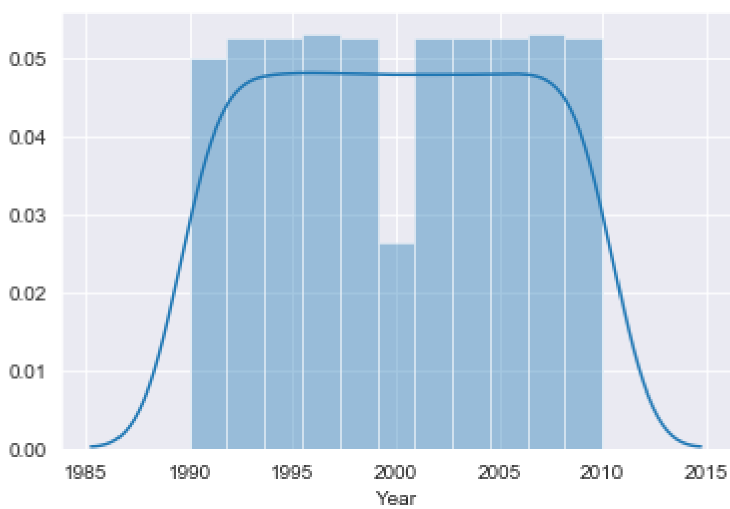
Out[115]: <matplotlib.collections.PathCollection at 0x7ff4cfab7a90>



The plot above shows a postively correlated pattern between 'Year' and 'Volume'

In [112]:
```
sns.distplot(df['Year'])
```

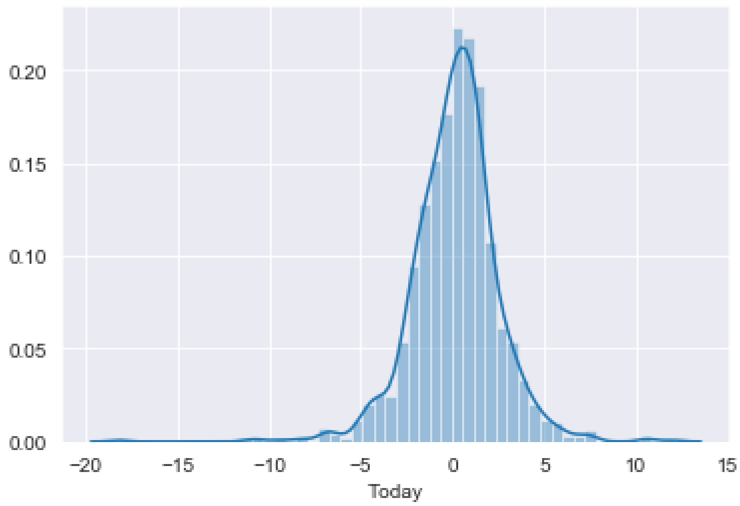
Out[112]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff4cf92a510>



The feature 'Year' is uniformly distributed except for 2000 with only half data points of other years.

In [113]:
```
sns.distplot(df['Today'])
```

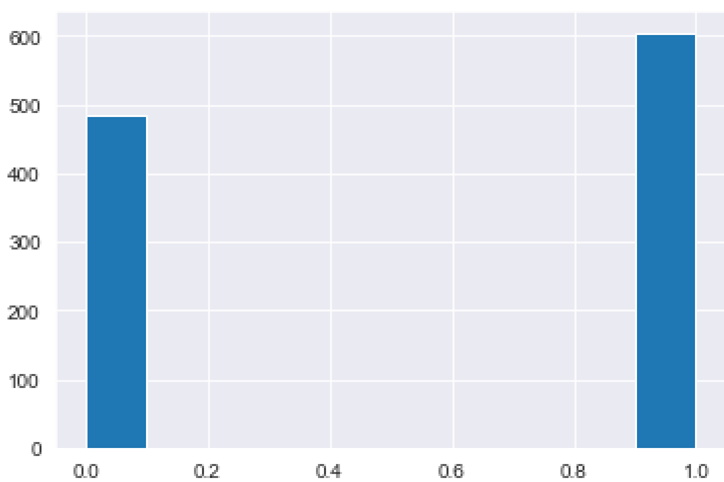
Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff4cf9feb90>



The distribution of 'Today'(percentage returns) looks like a Normal Distribution with mean of 0.

In [120]:
```
plt.hist(df['Direction'])
```

Out[120]:
```
(array([484.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 605.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



The number of days with positive returns is a little more than the number of days with negative returns.

Part (b) Logistic Regression

In [123]:
```
formula = 'Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume'
model_b = smf.logit(formula, data=df).fit()
model_b.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.682441
         Iterations 4
```

Out[123]:

Logit Regression Results

Dep. Variable:	Direction	**No. Observations:**	1089
Model:	Logit	**Df Residuals:**	1082
Method:	MLE	**Df Model:**	6
Date:	Thu, 15 Oct 2020	**Pseudo R-squ.:**	0.006580
Time:	06:01:01	**Log-Likelihood:**	-743.18
converged:	True	**LL-Null:**	-748.10
Covariance Type:	nonrobust	**LLR p-value:**	0.1313

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	0.2669	0.086	3.106	0.002	0.098	0.435
Lag1	-0.0413	0.026	-1.563	0.118	-0.093	0.010
Lag2	0.0584	0.027	2.175	0.030	0.006	0.111
Lag3	-0.0161	0.027	-0.602	0.547	-0.068	0.036
Lag4	-0.0278	0.026	-1.050	0.294	-0.080	0.024
Lag5	-0.0145	0.026	-0.549	0.583	-0.066	0.037
Volume	-0.0227	0.037	-0.616	0.538	-0.095	0.050

Significant predictor: 'Lag2' has a p-value smaller than 0.05, therefore statistically significant.

In [128]:
```
print(model_b.pvalues[model_b.pvalues<0.05].drop('Intercept'))
```

```
Lag2    0.029601
dtype: float64
```

Part (c) Confusion Matrix from Logistic Regression

In [129]:
```
features = ['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume']
response = 'Direction'
X = df[features]
y = df[response]
logreg = LogisticRegression(penalty='none')
logreg.fit(X, y)
y_pred = logreg.predict(X)
df_confusion = pd.crosstab(y, y_pred)
df_confusion = pd.crosstab(y, y_pred, rownames=['Actual'], colnames=['Pr
edicted'], margins=True)
display(df_confusion)
```

Predicted	0	1	All
Actual			
0	54	430	484
1	48	557	605
All	102	987	1089

0 stands for 'Down' and 1 stands for 'Up'.

In [143]:
```
# percentage of 'Up' in 'Direction' of all data points.
df['Direction'].mean()
```

Out[143]: 0.5555555555555556

In [206]:
```
# Overall accuracy:
oa_c = (df_confusion[0][0]+df_confusion[1][1])/df_confusion['All']['All'
]
oa_c
```

Out[206]: 0.5610651974288338

Overall prediction accuracy is 0.561, only a little bit better than naively predicting 'Direction' going up based on the percentage of 'Up' in our dataset, which is 0.556

In [207]:
```
# False positive rate:
fp_c = df_confusion[1][0]/df_confusion['All'][0]
fp_c
```

Out[207]: 0.8884297520661157

A very high False Positive rate, not a good sign. 80% of stocks that actually went down were predicted going up.

In [208]:
```
# True positive rate(sensitivity):
tp_c = df_confusion[1][1]/df_confusion['All'][1]
tp_c
```

Out[208]: 0.9206611570247933

A very high 0.92 True positive rate, meaning 92% of stocks that actually went up were predicted correctly. Good at predicting True Positives.

Part (d) Logistic Regression with only 'Lag2' as predictor.

In [154]:
```
df_train = df[(df['Year']>=1990)&(df['Year']<=2008)]
df_test = df[(df['Year']>=2009)&(df['Year']<=2010)]
features_d = ['Lag2']
response_d = 'Direction'
X_d = df_train[features_d]
y_d = df_train[response_d]
logreg_d = LogisticRegression(penalty='none')
logreg_d.fit(X_d, y_d)
y_pred_d = logreg_d.predict(df_test[features_d])
y_test = df_test[response_d]
df_confusion_d = pd.crosstab(y_test, y_pred_d)
df_confusion_d = pd.crosstab(y_test, y_pred_d, rownames=['Actual'], coln
ames=['Predicted'], margins=True)
display(df_confusion_d)
```

Predicted	0	1	All
Actual			
0	9	34	43
1	5	56	61
All	14	90	104

In [209]:
```
# Overall accuracy:
oa_d = (df_confusion_d[0][0]+df_confusion_d[1][1])/df_confusion_d['All']
['All']
oa_d
```

Out[209]: 0.625

In [210]:
```
# False positive rate:
fp_d = df_confusion_d[1][0]/df_confusion_d['All'][0]
fp_d
```

Out[210]: 0.7906976744186046

In [211]:
```
# True positive rate(sensitivity):
tp_d = df_confusion_d[1][1]/df_confusion_d['All'][1]
tp_d
```

Out[211]: 0.9180327868852459

Comments: Both overall accuracy and False Positive rates are better. So, using only 'Lag2' in a Logistic Regression model is better than using all features.

Part (e) LDA

In [159]:
```
lda = LDA()
lda.fit(X_d, y_d)

# Priors, group means, and coefficients of linear discriminants
priors = pd.DataFrame(lda.priors_, index=lda.classes_).T
print("Prior probabilities of groups:")
display(priors)
gmeans = pd.DataFrame(lda.means_, index=lda.classes_, columns=features_d
)
print("\nGroup means:")
display(gmeans)
coef = pd.DataFrame(lda.scalings_, columns=['LD1'], index=features_d)
print("\nCoefficients of linear discriminants:")
display(coef)
```

```
Prior probabilities of groups:
```

	0	1
0	0.447716	0.552284

```
Group means:
```

	Lag2
0	-0.035683
1	0.260366

```
Coefficients of linear discriminants:
```

	LD1
Lag2	0.441416

In [160]:
```
# Compute the confusion Matrix:
y_pred_e = lda.predict(df_test[features_d])
y_test_e = df_test[response_d]
df_confusion_e = pd.crosstab(y_test_e, y_pred_e)
df_confusion_e = pd.crosstab(y_test_e, y_pred_e, rownames=['Actual'], co
lnames=['Predicted'], margins=True)
display(df_confusion_e)
```

Predicted	0	1	All
Actual			
0	9	34	43
1	5	56	61
All	14	90	104

In [212]:
```
# Overall accuracy:
oa_e = (df_confusion_e[0][0]+df_confusion_e[1][1])/df_confusion_e['All']
['All']
oa_e
```

Out[212]: 0.625

In [213]:
```
# False positive rate:
fp_e = df_confusion_e[1][0]/df_confusion_e['All'][0]
fp_e
```

Out[213]: 0.7906976744186046

In [214]:
```
# True positive rate(sensitivity):
tp_e = df_confusion_e[1][1]/df_confusion_e['All'][1]
tp_e
```

Out[214]: 0.9180327868852459

Comments:
Exactly the same accuracy numbers as the Logistic Regression model in Part (d).

Part (f) QDA

In [165]:
```
qda = QDA()
qda.fit(X_d, y_d)

# Priors, group means, and coefficients of quadratic discriminants
priors_f = pd.DataFrame(qda.priors_, index=qda.classes_, columns=['']).T
print("Prior probabilities of groups:")
display(priors_f)
gmeans_f = pd.DataFrame(qda.means_, index=qda.classes_, columns=features
_d)
print("\nGroup means:")
display(gmeans_f)
```

Prior probabilities of groups:

	0	1
	0.447716	0.552284

Group means:

	Lag2
0	-0.035683
1	0.260366

In [197]:
```
# Compute the Confusion Matrix:
y_pred_f = qda.predict(df_test[features_d])
y_test_f = df_test[response_d]
df_confusion_f = pd.crosstab(y_test_f, y_pred_f)
df_confusion_f = pd.crosstab(y_test_f, y_pred_f, rownames=['Actual'], co
lnames=['Predicted'], margins=True)
display(df_confusion_f)
```

Predicted	1	All
Actual		
0	43	43
1	61	61
All	104	104

In [198]:
```
print(y_pred_f)
# All predictions on the test dataset are 1's, i.e. 'Up' direction.
# The number of '0', i.e. 'Down' predictions are omitted, because there
 zero 'Down' predictions.
# Add them manually to the confusion matrix.
df_confusion_f.insert(loc=0, column=0, value=[0,0,0])
display(df_confusion_f)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

Predicted	**0**	**1**	**All**
Actual			
0	0	43	43
1	0	61	61
All	0	104	104

In [215]:
```
# Overall accuracy:
oa_f = (df_confusion_f[0][0]+df_confusion_f[1][1])/df_confusion_f['All']
['All']
oa_f
```

Out[215]: 0.5865384615384616

In [216]:
```
# False positive rate:
fp_f = df_confusion_f[1][0]/df_confusion_f['All'][0]
fp_f
```

Out[216]: 1.0

In [217]:
```
# True positive rate(sensitivity):
tp_f = df_confusion_f[1][1]/df_confusion_f['All'][1]
tp_f
```

Out[217]: 1.0

Comments:
All predictions in the test dataset are 1, i.e. 'Up' direction.
Both False Positive rate and True Positive rate are 100%, since the model is only predicting 1 value.
Overall accuracy dropped compared to the LDA and Logistic Regression using only 'Lag2' as the predictor.

Part (g) KNN with K=1

In [202]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_d, y_d)
y_pred_g = knn.predict(df_test[features_d])
y_test_g = df_test[response_d]
df_confusion_g = pd.crosstab(y_test_g, y_pred_g)
df_confusion_g = pd.crosstab(y_test_g, y_pred_g, rownames=['Actual'], colnames=['Predicted'], margins=True)
display(df_confusion_g)
```

Predicted	0	1	All
Actual			
0	21	22	43
1	31	30	61
All	52	52	104

In [218]:
```
# Overall accuracy:
oa_g = (df_confusion_g[0][0]+df_confusion_g[1][1])/df_confusion_g['All']['All']
oa_g
```

Out[218]: 0.49038461538461536

In [219]:
```
# False positive rate:
fp_g = df_confusion_g[1][0]/df_confusion_g['All'][0]
fp_g
```

Out[219]: 0.5116279069767442

In [220]:
```
# True positive rate(sensitivity):
tp_g = df_confusion_g[1][1]/df_confusion_g['All'][1]
tp_g
```

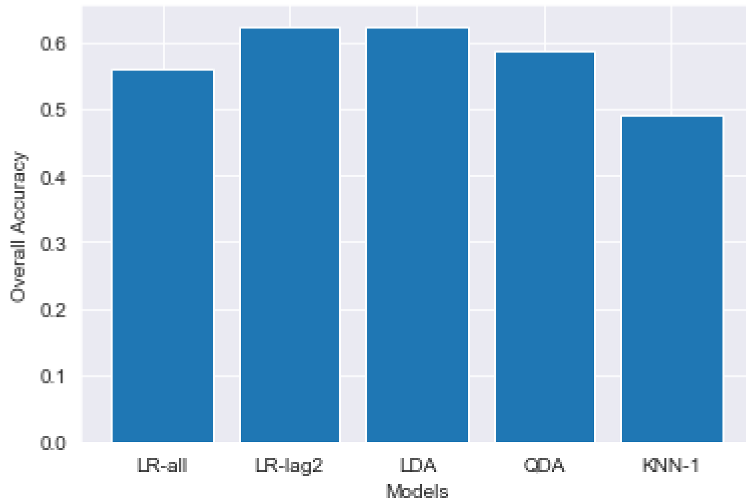
Out[220]: 0.4918032786885246

Comments:
Overall accuracy dropped below 50%
False Positive rate dropped significantly compared to all the previous models, which is a good sign. Should consider this model, if False Positive predictions can cause much more damage than False Negative predictions.
True positive rate also dropped below 50%.

Part (h) Which performs the best?

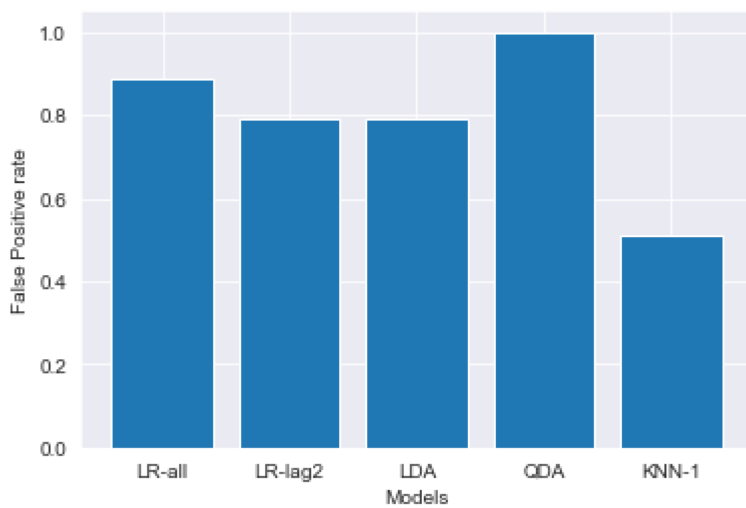
In [224]:
```
# overall accuracy: the higher, the better
plt.bar(['LR-all','LR-lag2','LDA','QDA','KNN-1'],[oa_c,oa_d,oa_e,oa_f,oa
_g])
plt.xlabel('Models')
plt.ylabel('Overall Accuracy')
```

Out[224]: Text(0, 0.5, 'Overall Accuracy')



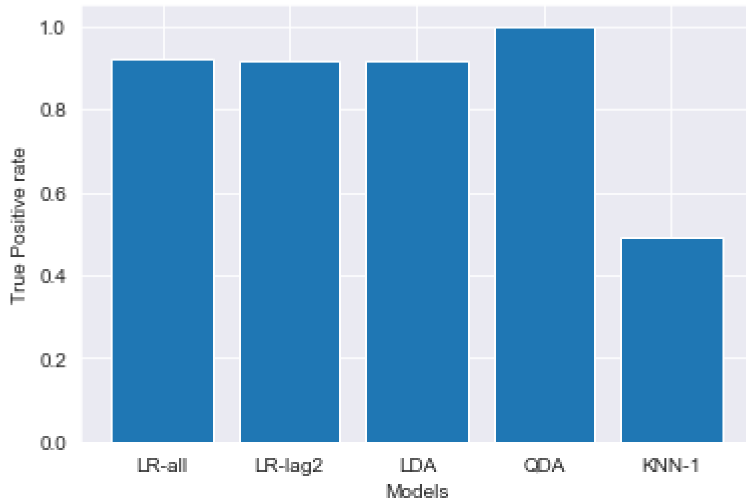
In [225]:
```
# False Positive rate: the lower, the better
plt.bar(['LR-all','LR-lag2','LDA','QDA','KNN-1'],[fp_c,fp_d,fp_e,fp_f,fp
_g])
plt.xlabel('Models')
plt.ylabel('False Positive rate')
```

Out[225]: Text(0, 0.5, 'False Positive rate')



In [226]:
```
# True Positive rate: the higher, the better
plt.bar(['LR-all','LR-lag2','LDA','QDA','KNN-1'],[tp_c,tp_d,tp_e,tp_f,tp
_g])
plt.xlabel('Models')
plt.ylabel('True Positive rate')
```

Out[226]: Text(0, 0.5, 'True Positive rate')



Comments:
By comparing the overall accuracy of all the models from Part(d) to Part(g), LDA and Logistic Regresion models with only 'Lag2' perform the best with both the highest accuracy rate tied at 62.5%, followed by QDA, Logistic Regression with all features and KNN with K=1.
One thing that is worth noting is that the KNN model with $K = 1$ has a much lower False Positive rate compared to all other models. Depending on the specific investment strategy, maybe sometimes a lower False Positive rate would be more important than a higher overall accuracy.

Part (i) Try different combinations of predictors.

Logistic Regression with 3 terms: 'Lag1', 'Lag2' and 'Lag3'.

In [238]:
```
features_i_lg3 = ['Lag1','Lag2','Lag3']
response_i_lg3 = 'Direction'
X_i_lg3 = df_train[features_i_lg3]
y_i_lg3 = df_train[response_i_lg3]
logreg_i_lg3 = LogisticRegression(penalty='none')
logreg_i_lg3.fit(X_i_lg3, y_i_lg3)
y_pred_i_lg3 = logreg_i_lg3.predict(df_test[features_i_lg3])
y_test_i_lg3 = df_test[response_i_lg3]
df_confusion_i_lg3 = pd.crosstab(y_test_i_lg3, y_pred_i_lg3)
df_confusion_i_lg3 = pd.crosstab(y_test_i_lg3, y_pred_i_lg3, rownames=[
'Actual'], colnames=['Predicted'], margins=True)
display(df_confusion_i_lg3)
```

Predicted	0	1	All
Actual			
0	8	35	43
1	9	52	61
All	17	87	104

In [245]:
```
# Overall accuracy:
oa_i_lg3 = (df_confusion_i_lg3[0][0]+df_confusion_i_lg3[1][1])/df_confus
ion_i_lg3['All']['All']
oa_i_lg3
```

Out[245]: 0.5769230769230769

In [246]:
```
# False positive rate:
fp_i_lg3 = df_confusion_i_lg3[1][0]/df_confusion_i_lg3['All'][0]
fp_i_lg3
```

Out[246]: 0.813953488372093

In [247]:
```
# True positive rate(sensitivity):
tp_i_lg3 = df_confusion_i_lg3[1][1]/df_confusion_i_lg3['All'][1]
tp_i_lg3
```

Out[247]: 0.8524590163934426

Logistic Regression with 2 terms and 1 interaction term: 'Lag1', 'Lag2', 'Lag1xLag2'

In [ ]:
```
# Add interaction terms 'Lag1xLag2' into the training and test DataFrame
s:
df_train['Lag1*Lag2'] = df_train['Lag1']*df_train['Lag2']
df_test['Lag1*Lag2'] = df_test['Lag1']*df_test['Lag2']
```

In [244]:
```
features_i_lg12 = ['Lag1','Lag2','Lag1*Lag2']
response_i_lg12 = 'Direction'
X_i_lg12 = df_train[features_i_lg12]
y_i_lg12 = df_train[response_i_lg12]
logreg_i_lg12 = LogisticRegression(penalty='none')
logreg_i_lg12.fit(X_i_lg12, y_i_lg12)
y_pred_i_lg12 = logreg_i_lg12.predict(df_test[features_i_lg12])
y_test_i_lg12 = df_test[response_i_lg12]
df_confusion_i_lg12 = pd.crosstab(y_test_i_lg12, y_pred_i_lg12)
df_confusion_i_lg12 = pd.crosstab(y_test_i_lg12, y_pred_i_lg12, rownames
=['Actual'], colnames=['Predicted'], margins=True)
display(df_confusion_i_lg12)
```

Predicted	0	1	All
Actual			
0	7	36	43
1	8	53	61
All	15	89	104

In [249]:
```
# Overall accuracy:
oa_i_lg12 = (df_confusion_i_lg12[0][0]+df_confusion_i_lg12[1][1])/df_con
fusion_i_lg12['All']['All']
oa_i_lg12
```

Out[249]: 0.5769230769230769

In [250]:
```
# False positive rate:
fp_i_lg12 = df_confusion_i_lg12[1][0]/df_confusion_i_lg12['All'][0]
fp_i_lg12
```

Out[250]: 0.8372093023255814

In [251]:
```
# True positive rate(sensitivity):
tp_i_lg12 = df_confusion_i_lg12[1][1]/df_confusion_i_lg12['All'][1]
tp_i_lg12
```

Out[251]: 0.8688524590163934

LDA with 2 terms and 1 interaction term: 'Lag1', 'Lag2', 'Lag1xLag2'

In [252]:
```
lda_i = LDA()
lda_i.fit(X_i_lg12, y_i_lg12)

# Priors, group means, and coefficients of linear discriminants
priors_i = pd.DataFrame(lda_i.priors_, index=lda_i.classes_).T
print("Prior probabilities of groups:")
display(priors_i)
gmeans_i = pd.DataFrame(lda_i.means_, index=lda_i.classes_, columns=feat
ures_i_lg12)
print("\nGroup means:")
display(gmeans_i)
coef_i = pd.DataFrame(lda_i.scalings_, columns=['LD1'], index=features_i
_lg12)
print("\nCoefficients of linear discriminants:")
display(coef_i)
```

```
Prior probabilities of groups:
```

	0	1
0	0.447716	0.552284

```
Group means:
```

	Lag1	Lag2	Lag1*Lag2
0	0.289444	-0.035683	-0.801449
1	-0.009213	0.260366	-0.139363

```
Coefficients of linear discriminants:
```

	LD1
Lag1	-0.285485
Lag2	0.295080
Lag1*Lag2	0.009629

In [254]:
```
# Compute the confusion Matrix:
y_pred_il = lda_i.predict(df_test[features_i_lg12])
y_test_il = df_test[response_i_lg12]
df_confusion_il = pd.crosstab(y_test_il, y_pred_il)
df_confusion_il = pd.crosstab(y_test_il, y_pred_il, rownames=['Actual'],
colnames=['Predicted'], margins=True)
display(df_confusion_il)
```

Predicted Actual	0	1	All
0	7	36	43
1	8	53	61
All	15	89	104

In [256]:
```
# Overall accuracy:
oa_il = (df_confusion_il[0][0]+df_confusion_il[1][1])/df_confusion_il['A
ll']['All']
oa_il
```

Out[256]: 0.5769230769230769

In [257]:
```
# False positive rate:
fp_il = df_confusion_il[1][0]/df_confusion_il['All'][0]
fp_il
```

Out[257]: 0.8372093023255814

In [258]:
```
# True positive rate(sensitivity):
tp_il = df_confusion_il[1][1]/df_confusion_il['All'][1]
tp_il
```

Out[258]: 0.8688524590163934

QDA with 2 terms and 1 interaction term: 'Lag1', 'Lag2', 'Lag1xLag2'

In [253]:
```
qda_i = QDA()
qda_i.fit(X_i_lg12, y_i_lg12)

# Priors, group means, and coefficients of quadratic discriminants
priors_iq = pd.DataFrame(qda_i.priors_, index=qda_i.classes_, columns=[
'']).T
print("Prior probabilities of groups:")
display(priors_iq)
gmeans_iq = pd.DataFrame(qda_i.means_, index=qda_i.classes_, columns=fea
tures_i_lg12)
print("\nGroup means:")
display(gmeans_iq)
```

```
Prior probabilities of groups:
```

	0	1
	0.447716	0.552284

```
Group means:
```

	Lag1	Lag2	Lag1*Lag2
0	0.289444	-0.035683	-0.801449
1	-0.009213	0.260366	-0.139363

In [255]:
```
# Compute the confusion Matrix:
y_pred_iq = qda_i.predict(df_test[features_i_lg12])
y_test_iq = df_test[response_i_lg12]
df_confusion_iq = pd.crosstab(y_test_iq, y_pred_iq)
df_confusion_iq = pd.crosstab(y_test_iq, y_pred_iq, rownames=['Actual'],
colnames=['Predicted'], margins=True)
display(df_confusion_iq)
```

Predicted	0	1	All
Actual			
0	23	20	43
1	36	25	61
All	59	45	104

In [260]:
```
# Overall accuracy:
oa_iq = (df_confusion_iq[0][0]+df_confusion_iq[1][1])/df_confusion_iq['A
ll']['All']
oa_iq
```

Out[260]: 0.46153846153846156

In [261]:
```
# False positive rate:
fp_iq = df_confusion_iq[1][0]/df_confusion_iq['All'][0]
fp_iq
```

Out[261]: 0.46511627906976744

In [262]:
```
# True positive rate(sensitivity):
tp_iq = df_confusion_iq[1][1]/df_confusion_iq['All'][1]
tp_iq
```

Out[262]: 0.4098360655737705

KNN with k=3

In [227]:
```
knn_3 = KNeighborsClassifier(n_neighbors=3)
knn_3.fit(X_d, y_d)
y_pred_i_k3 = knn_3.predict(df_test[features_d])
y_test_i_k3 = df_test[response_d]
df_confusion_i_k3 = pd.crosstab(y_test_i_k3, y_pred_i_k3)
df_confusion_i_k3 = pd.crosstab(y_test_i_k3, y_pred_i_k3, rownames=['Act
ual'], colnames=['Predicted'], margins=True)
display(df_confusion_i_k3)
```

Predicted	**0**	**1**	**All**
Actual			
0	15	28	43
1	20	41	61
All	35	69	104

In [229]:
```
# Overall accuracy:
oa_i_k3 = (df_confusion_i_k3[0][0]+df_confusion_i_k3[1][1])/df_confusion
_i_k3['All']['All']
oa_i_k3
```

Out[229]: 0.5384615384615384

In [230]:
```
# False positive rate:
fp_i_k3 = df_confusion_i_k3[1][0]/df_confusion_i_k3['All'][0]
fp_i_k3
```

Out[230]: 0.6511627906976745

In [231]:
```
# True positive rate(sensitivity):
tp_i_k3 = df_confusion_i_k3[1][1]/df_confusion_i_k3['All'][1]
tp_i_k3
```

Out[231]: 0.6721311475409836

KNN with k=10

In [228]:
```
knn_10 = KNeighborsClassifier(n_neighbors=10)
knn_10.fit(X_d, y_d)
y_pred_i_k10 = knn_10.predict(df_test[features_d])
y_test_i_k10 = df_test[response_d]
df_confusion_i_k10 = pd.crosstab(y_test_i_k10, y_pred_i_k10)
df_confusion_i_k10 = pd.crosstab(y_test_i_k10, y_pred_i_k10, rownames=[
'Actual'], colnames=['Predicted'], margins=True)
display(df_confusion_i_k10)
```

Predicted	0	1	All
Actual			
0	22	21	43
1	24	37	61
All	46	58	104

In [232]:
```
# Overall accuracy:
oa_i_k10 = (df_confusion_i_k10[0][0]+df_confusion_i_k10[1][1])/df_confus
ion_i_k10['All']['All']
oa_i_k10
```

Out[232]: 0.5673076923076923

In [233]:
```
# False positive rate:
fp_i_k10 = df_confusion_i_k10[1][0]/df_confusion_i_k10['All'][0]
fp_i_k10
```

Out[233]: 0.4883720930232558

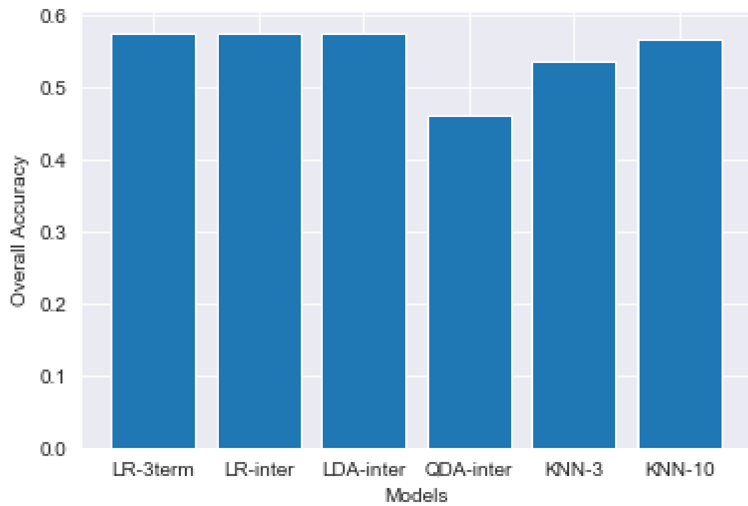
In [234]:
```
# True positive rate(sensitivity):
tp_i_k10 = df_confusion_i_k10[1][1]/df_confusion_i_k10['All'][1]
tp_i_k10
```

Out[234]: 0.6065573770491803

Compare all experiment models above:

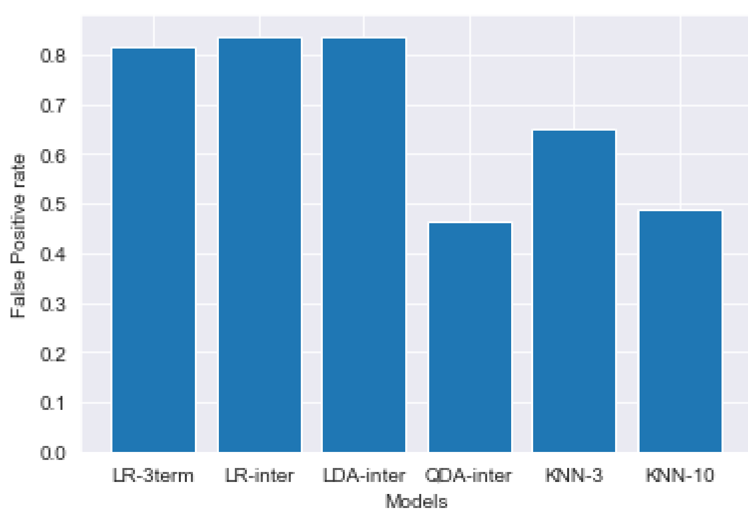
In [263]:
```
# overall accuracy: the higher, the better
plt.bar(['LR-3term','LR-inter','LDA-inter','QDA-inter','KNN-3','KNN-10'
],[oa_i_lg3,oa_i_lg12,oa_il,oa_iq,oa_i_k3,oa_i_k10])
plt.xlabel('Models')
plt.ylabel('Overall Accuracy')
```

Out[263]: Text(0, 0.5, 'Overall Accuracy')



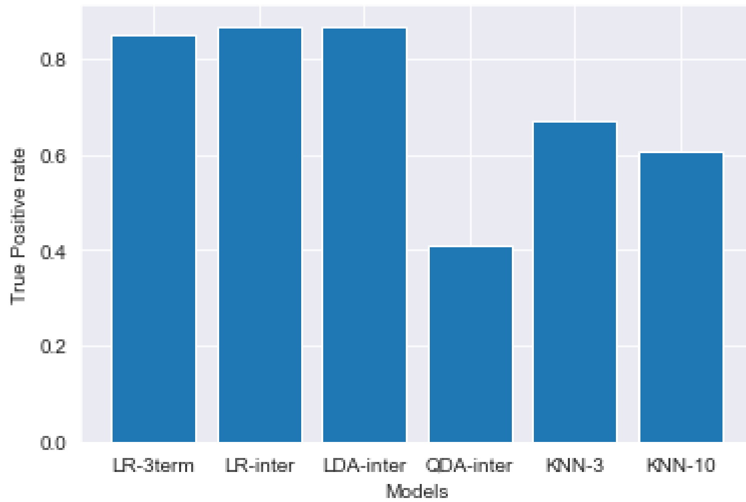
In [264]:
```
# False Positive rate: the lower, the better
plt.bar(['LR-3term','LR-inter','LDA-inter','QDA-inter','KNN-3','KNN-10'
],[fp_i_lg3,fp_i_lg12,fp_il,fp_iq,fp_i_k3,fp_i_k10])
plt.xlabel('Models')
plt.ylabel('False Positive rate')
```

Out[264]: Text(0, 0.5, 'False Positive rate')



In [265]:
```
# True Positive rate: the higher, the better
plt.bar(['LR-3term','LR-inter','LDA-inter','QDA-inter','KNN-3','KNN-10'
],[tp_i_lg3,tp_i_lg12,tp_il,tp_iq,tp_i_k3,tp_i_k10])
plt.xlabel('Models')
plt.ylabel('True Positive rate')
```

Out[265]: Text(0, 0.5, 'True Positive rate')



Comments:

Logistic regression models and LDA model with a interaction term perform best in terms of the overall accuracy. QDA model with a interaction term perform best with the lowest False Positive Rate.

Question 3

Part 1

Given the unconstrained maximization problem:

$$\max_{\alpha} \frac{\alpha^T B\alpha}{\alpha^T W\alpha}$$

Using the $scale\ invariance$ of the Rayleigh quotient, rewrite into a constrained maximization problem:

$$\max_{\alpha} \alpha^T B\alpha$$
$$s.t.\ \alpha^T W\alpha = 1$$

Solve the optimization problem above, using Lagrange Multipliers.
First, define the Lagrangian form, with λ being the Lagrange Multiplier:

$$L(\alpha, \lambda) = \alpha^T B\alpha + \lambda(\alpha^T W\alpha - 1)$$

Now, take partial derivatives with respect to α and λ, and set them equal to 0:

$$\frac{\partial L(\alpha, \lambda)}{\partial \alpha} = 2B\alpha + 2\lambda W\alpha = 0$$

$$\frac{\partial L(\alpha, \lambda)}{\partial \lambda} = \alpha^T W\alpha - 1 = 0$$

Solve the 1st equation ang get:

$$-B\alpha = \lambda W\alpha$$
$$-W^{-1}B\alpha = \lambda\alpha$$

We get an eigenvalue problem, in eigen decomposition form.

The optimal solution a^* will be the eigenvector corresponding to the matrix $-W^{-1}B$ with the largest eigenvalue λ.

Part 2

From in-class lecture slides, we derived that the $l_t h$ discriminant variable is

$$Z_l = v_l^T D^{-\frac{1}{2}} U^T x$$

Therefore, the 1_{st} discriminant variable is equal to $v_1^T D^{-\frac{1}{2}} U^T x$

Since $W = \sum$
By eigen-decomposition, $W = (W^{1/2})^T W^{1/2}$
Compute $B^* = (W^{-\frac{1}{2}})^T BW^{-\frac{1}{2}}$
The discriminant coordinates are $a_l = W^{-1/2}v_l^*$

Therefore, essentially by finding the lienar combination $Z = a^T X$ such that between-class variance is maximized relative to the within-class variance. We are finding the optimal a^*, such that $a^* x$ is the 1st discriminant variable Z_1 above.

Question 4

Part 1

For a binary logistic regression, we have derived the following in class:

$$P(Y=1|X) = \frac{exp(\beta_{10} + \beta_k^T x)}{1 + \sum_{\iota=1}^{2} exp(\beta_{\iota 0} + \beta_\iota^T x)}$$

$$P(Y=0|X) = \frac{1}{1 + \sum_{\iota=1}^{2} exp(\beta_{\iota 0} + \beta_\iota^T x)}$$

If we predict $Class\ 1$ when $P(Y=1|X) > P(Y=0|X)$, it is equivalent to: $\frac{P(Y=1|X)}{P(Y=1|X)} > 1$.

Try to prove it is a linear classifier via $monotone\ transformation$ by taking log on both sides to get the log-ratio between 2 classes:

$$log(\frac{P(Y=1|X)}{P(Y=0|X)}) > 0$$

Expand the term on the left side and we get:

$$log(exp(\beta_{10} + \beta_k^T x) - log(1 + \sum_{\iota=1}^{2} exp(\beta_{\iota 0} + \beta_\iota^T x)) - log(1) + log(1 + \sum_{\iota=1}^{2} exp(\beta_{\iota 0} + \beta_\iota^T x)) > 0$$

The 3rd term $log(1) = 0$. The 2nd and the last term cancelled out, so we get:

$$log(exp(\beta_{10} + \beta_k^T x) > 0$$

Equivalently, we get the classifier below:
Predicting $Class\ 1$ when

$$\beta_{10} + \beta_k^T x > 0$$

Therefore, the decision boundary is $\beta_{10} + \beta_k^T x$ = 0 shows that logistic regression yields a linear classifier.

Part 2

Define the prior probabilities over classes Y, for k = 0,1:

$$P(Y=1) = \pi, P(Y=0) = 1 - \pi$$

Derive the parametric forms of $P(Y|X)$ under the Gaussian class-conditional probabilities:

$$P(Y=1|X) = \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1) + P(Y=0)P(X|Y=0)}$$

$$P(Y=1|X) = \frac{1}{1 + exp(ln\frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}$$

$$P(Y=1|X) = \frac{1}{1 + exp(ln\frac{P(Y=0)}{P(Y=1)} + \sum_j ln(\frac{P(X_j|Y=0)}{P(X_j|Y=1)}))}$$

$$P(Y=1|X) = \frac{1}{1 + exp(ln\frac{1-\pi}{\pi} + \sum_j ln(\frac{P(X_j|Y=0)}{P(X_j|Y=1)}))}$$

Substitute Gaussian PDF's into the conditional probabilities:

$$P(Y=1|X) = \frac{1}{1 + exp(ln\frac{1-\pi}{\pi} + \sum_j (\frac{\mu_{j0}-\mu_{j1}}{\sigma_j^2} X_j + \frac{\mu_{j0}^2-\mu_{j1}^2}{2\sigma_j^2}))}$$

Let $w_j = \frac{\mu_{j0}-\mu_{j1}}{\sigma_j^2}$ for $j = 1, 2, \ldots, n$ and $w_0 = ln\frac{1-\pi}{\pi} + \sum_j \frac{\mu_{j0}^2-\mu_{j1}^2}{2\sigma_j^2}$

We then have:

$$P(Y=1|X) = \frac{1}{1 + exp(w_0 + \sum_{j=1}^n w_j X_j)}$$

$$P(Y=0|X) = \frac{exp(w_0 + \sum_{j=1}^n w_j X_j)}{1 + exp(w_0 + \sum_{j=1}^n w_j X_j)}$$

Similarly, set the log-odds of these 2 probabilities to 0 and solve to get the decision boundary:

$$log(\frac{P(Y=1|X)}{P(Y=0|X)})=0$$

$$log(1)-log(1+exp(w_0+\sum_{i=j}^{n}w_jX_j))-log(exp(w_0+\sum_{j=1}^{n}w_jX_j))+log(1+exp(w_0+\sum_{j=1}^{n}w_jX_j))=0$$

$$log(exp(w_0+\sum_{j=1}^{n}w_jX_j))=0$$

$$w_0+\sum_{j=1}^{n}w_jX_j=0$$

Therefore, the decision boundary is $w_0+\sum_{j=1}^{n}w_jX_j=0$, a linear classifier in the same parametric form as Logistic Regression above.

Part 3

By comparing the 2 classifiers, the parameters in Logistic Regression can be expressed in terms of the parameters w_j in the Naive Bayes.

The optimal parameters β in Logistic Regression is computed iteratively in the following method:

$$\beta^{new}=argmin_{\beta}(Z-X_{\beta})^TW(Z-X_{\beta})$$

Where W is the diagonal matrix with $P(X_j;\beta)(1-P(X_j;\beta))$ for $j=1,2,\ldots,n$ on the diagonal.

Yes, if the assumptions of Gaussian class-conditional probabilities hold, the Naive Bayes and Logistic Regression classifiers will converge towards a same classifier as the training sample size grows to infinity.

Part 4

I think the Naive Bayes classifier will perform better given that the class conditional distributions are Gaussian. Since the assumption of Bayes Theorem hold, the resulting model will have very low bias compared to the data used in real life.
The generative classifer will converge much more quickly than the Logistic Regression, which is a discriminative classifier.
This will allow us to use fewer training samples to get the optimal parameters with the reasonably low bias at the same time, since the Gaussian conditional distributions hold.

Question 5

Part (a)

$P(o_1 \neq o_j) = 1 - \frac{1}{n}$
Because every observation has the same probability to be chosen and the probability of the 1st observation being the j-th observation is $\frac{1}{n}$.

Part (b)

$P(o_2 \neq o_j) = 1 - \frac{1}{n}$
Same logic as in Part(a), because bootstrap sampling is performed with replacement.

Part (c)

Because we have n observations in total, and we have known that every time, for $k = 1, \ldots, n$, $P(o_k \neq o_j) = \frac{n-1}{n}$, the probability that j-th observation is not in the samples will just be the $\prod_{k=1}^{n} P(o_k \neq o_j) = (1 - \frac{1}{n})^n$, for $k = 1, \ldots, n$.

Part (d)

n = 5

$P(in) = 1 - P(out) = 1 - (1 - \frac{1}{n})^n = 1 - (1 - \frac{1}{5})^5 \approx 0.672$

Part (e)

n = 100

$$P(in) = 1 - P(out) = 1 - (1 - \frac{1}{n})^n = 1 - (1 - \frac{1}{100})^{100} \approx 0.634$$

Part (f)

n = 10000

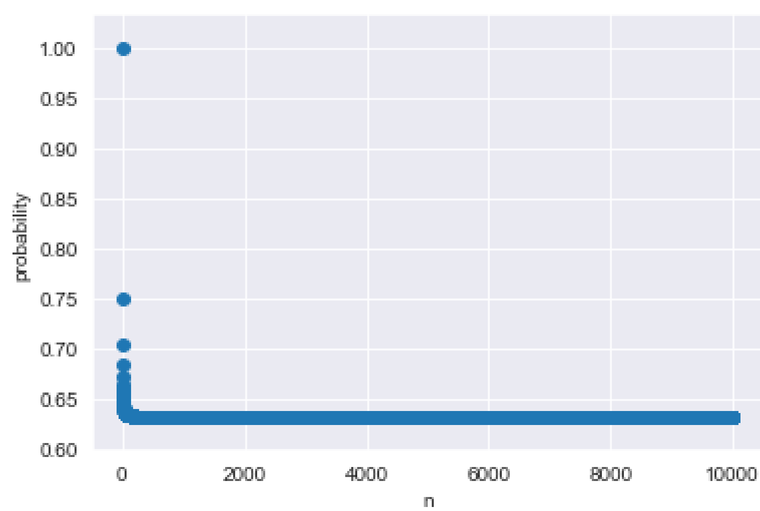
$$P(in) = 1 - P(out) = 1 - (1 - \frac{1}{n})^n = 1 - (1 - \frac{1}{10000})^{10000} \approx 0.632$$

Part (g)

In [13]:
```
n = np.arange(1,10001)
p = 1-(1-1/n)**n

plt.scatter(n,p)
plt.xlabel('n')
plt.ylabel('probability')
```

Out[13]: `Text(0, 0.5, 'probability')`



Comment:

When $n = 1$, we are only sampling 1 point from 1 point, the probability that this one point is in the sample is 1. After that the probability decreases extremely until it nearly reaches 0.63 and stayed above it.
Proof: $1 - \lim_{n \to \infty} 1 - (\frac{1}{n})^n = 1 - e^{-1} \approx 0.632$

Part (h)

n = 100, check if 4-th observation is in the bootstrap sample or not.

In [100]:
```
count = 0
for i in range(10000):
    count += np.sum(np.random.randint(1,101,size=100) == 4)>0
count/10000
```

Out[100]:
```
0.6355
```

Comment:

The result above is very close to the TRUE probability we obtained in Part (e) by formula $P \approx 0.634$