Lab - Logistic Regression, LDA, QDA, and KNN

Import notebook functions

```
from notebookfuncs import *
```

Examine the Smarket data — part of the ISLP library.

Consists of percentage returns for the S&P 500 stock index over 1,250 days, from the beginning of 2001 until the end of 2005.

For each date, we have recorded the percentage returns for each of the five previous trading days, Lag1 through Lag5. We have also recorded Volume (the number of shares traded on the previous day, in billions), Today (the percentage return on the date in question) and Direction (whether the market was Up or Down on this date).

Import the libraries

```
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from ISLP import load_data
from ISLP.models import (ModelSpec as MS , summarize)
```

New imports for this lab

```
from ISLP import confusion_table
from ISLP.models import contrast
from sklearn.discriminant_analysis import (LinearDiscriminantAnalysis as LDA , QuadraticDisc
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

Load the Smarket data.

```
Smarket = load_data('Smarket')
Smarket.describe()
```

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	To
count	1250.000000	1250.000000	1250.000000	1250.000000	1250.000000	1250.00000	1250.000000	12
mean	2003.016000	0.003834	0.003919	0.001716	0.001636	0.00561	1.478305	0.
std	1.409018	1.136299	1.136280	1.138703	1.138774	1.14755	0.360357	1.
min	2001.000000	-4.922000	-4.922000	-4.922000	-4.922000	-4.92200	0.356070	-4
25%	2002.000000	-0.639500	-0.639500	-0.640000	-0.640000	-0.64000	1.257400	-0
50%	2003.000000	0.039000	0.039000	0.038500	0.038500	0.03850	1.422950	0.
75%	2004.000000	0.596750	0.596750	0.596750	0.596750	0.59700	1.641675	0.
max	2005.000000	5.733000	5.733000	5.733000	5.733000	5.73300	3.152470	5.

```
Smarket.columns
```

```
Index(['Year', 'Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume', 'Today',
       'Direction'],
      dtype='object')
```

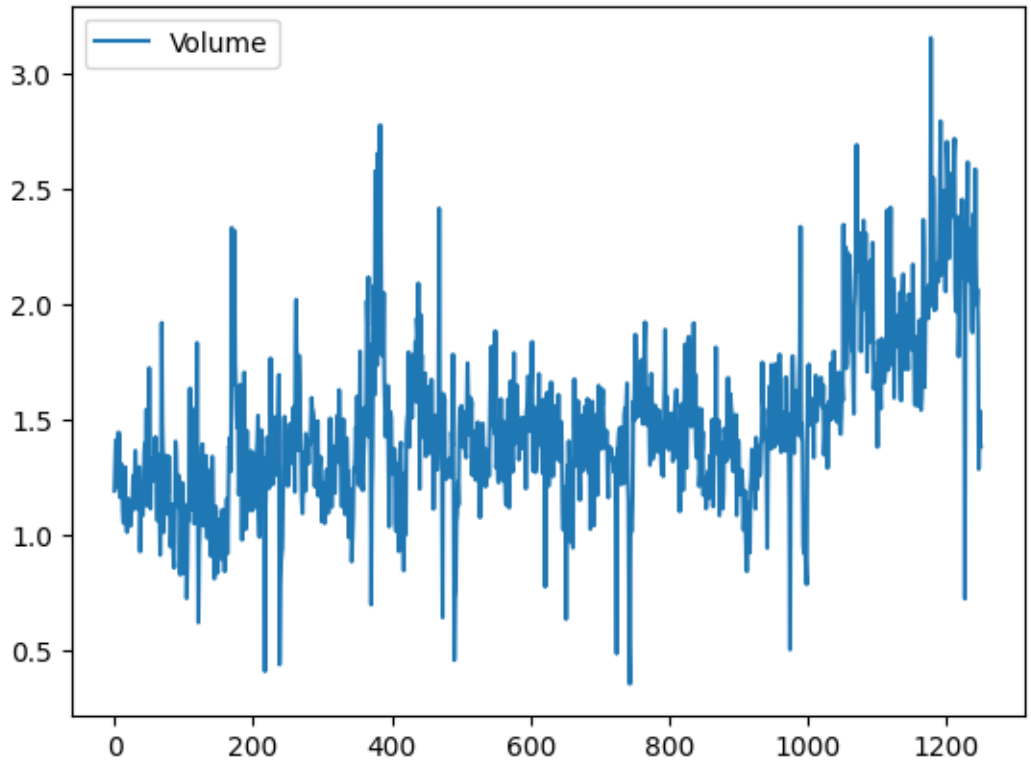
```
Smarket.Direction = Smarket.Direction.astype("category")
```

```
Smarket.corr(numeric_only=True)
```

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today
Year	1.000000	0.029700	0.030596	0.033195	0.035689	0.029788	0.539006	0.030095
Lag1	0.029700	1.000000	-0.026294	-0.010803	-0.002986	-0.005675	0.040910	-0.026155
Lag2	0.030596	-0.026294	1.000000	-0.025897	-0.010854	-0.003558	-0.043383	-0.010250
Lag3	0.033195	-0.010803	-0.025897	1.000000	-0.024051	-0.018808	-0.041824	-0.002448
Lag4	0.035689	-0.002986	-0.010854	-0.024051	1.000000	-0.027084	-0.048414	-0.006900
Lag5	0.029788	-0.005675	-0.003558	-0.018808	-0.027084	1.000000	-0.022002	-0.034860
Volume	0.539006	0.040910	-0.043383	-0.041824	-0.048414	-0.022002	1.000000	0.014592
Today	0.030095	-0.026155	-0.010250	-0.002448	-0.006900	-0.034860	0.014592	1.000000

- As one would expect, the correlations between the lagged return variables and today's return are close to zero. (Why? Random walk?) The only substantial correlation is between Year and Volume. By plotting the data we see that Volume is in creasing over time. In other words, the average number of shares traded daily increased from 2001 to 2005.

```
Smarket.plot(y='Volume');
```



Logistic Regression

Fit a logistic regression model to predict Direction using Lag1 through Lag5 and Volume.

We use the sm.GLM() function which fits Generalized Linear Models (GLMs) which includes logistic regression. We could alos sm.Logit() which fits a logit model directly. The syntax of sm.GLM() is similar to that of sm.OLS(), except that we must pass in the argument family=sm.families.Binomial() in order to tell statsmodels to run a logistic regression rather than some other type of GLM.

```
allvars = Smarket.columns.drop(['Today', 'Direction', 'Year'])
design = MS(allvars)
X = design.fit_transform(Smarket)
y = Smarket.Direction == 'Up'
family = sm.families.Binomial()
glm = sm.GLM(y, X, family=family)
results = glm.fit()
summarize(results)
```

	coef	std err	z	P>|z|
intercept	-0.1260	0.241	-0.523	0.601
Lag1	-0.0731	0.050	-1.457	0.145
Lag2	-0.0423	0.050	-0.845	0.398
Lag3	0.0111	0.050	0.222	0.824
Lag4	0.0094	0.050	0.187	0.851
Lag5	0.0103	0.050	0.208	0.835
Volume	0.1354	0.158	0.855	0.392

```
(results.pvalues.idxmin(), results.pvalues.min())
```

```
('Lag1', 0.14523155718601094)
```

- The smallest p-value here is associated with Lag1.
- The negative coefficient for this predictor suggests that if the market had a positive return yesterday then it is less likely to go up today.
- However, at a value of 0.15, the p-value is still relatively large.
- So there is no clear evidence of a real association between Lag1 and Direction.

```
results.params
```

```
intercept   -0.126000
Lag1        -0.073074
Lag2        -0.042301
Lag3         0.011085
Lag4         0.009359
Lag5         0.010313
Volume       0.135441
dtype: float64
```

Predict

The predict() method of results can be used to predict the probability that the market will go up, given values of the predictors. This method returns predictions on the probability scale. If no data set is supplied to the predict() function, then the probabilities are computed for the training data that was used to fit the logistic regression model. As with linear regression, one can pass an optional exog argument consistent with a design matrix if desired. Here we have printed only the first ten probabilities.

```
probs = results.predict()
probs[:10]
```

```
array([0.50708413, 0.48146788, 0.48113883, 0.51522236, 0.51078116,
       0.50695646, 0.49265087, 0.50922916, 0.51761353, 0.48883778])
```

In order to make a prediction as to whether the market will go up or down on a particular day, we must convert these predicted probabilities into class labels, Up or Down.

```
labels = np.array(['Down']*1250)
labels[probs >0.5] = "Up"
```

The confusion_table() function from the ISLP package summarizes these confusion predictions, showing how many observations were correctly or incorrectly classified. Our function, which is adapted from a similar function in the module sklearn.metrics, transposes the resulting matrix and includes row and column labels. The confusion_table() function takes as first argument the predicted labels, and second argument the true labels.

```
confusion_table(labels , Smarket.Direction)
```

Truth Predicted	Down	Up
Down	145	141
Up	457	507

The diagonal elements of the confusion matrix indicate correct predictions, while the off-diagonals represent incorrect predictions. Hence our model correctly predicted that the market would go up on 507 days and that it would go down on 145 days, for a total of 507 + 145 = 652 correct predictions. The np.mean() function can be used to compute the fraction of days for which the prediction was correct. In this case, logistic regression correctly predicted the movement of the market 52.2% of the time.

```
(507+145) /1250 , np.mean(labels == Smarket.Direction)
```

```
(0.5216, 0.5216)
```

At first glance, it appears that the logistic regression model is working a little better than random guessing. However, this result is misleading because we trained and tested the model on the same set of 1,250 observations. In other words, 100 − 52.2 = 47.8% is the training error rate. As we have seen previously, the training error rate is often overly optimistic — it tends to underestimate the test error rate. In order to better assess the accuracy of the logistic regression model in this setting, we can fit the model using part of the data, and then examine how well it predicts the held out data. This will yield a more realistic error rate, in the sense that in practice we will be interested in our model's performance not on the data that we used to fit the model, but rather on days in the future for which the market's movements are unknown.

Train and Test

To implement this strategy, we first create a Boolean vector corresponding to the observations from 2001 through 2004. We then use this vector to create a held out data set of observations from 2005.

```
train = (Smarket.Year < 2005)
Smarket_train = Smarket.loc[train]
Smarket_test = Smarket.loc[~train];
```

```
Smarket_train.shape
```

```
(998, 9)
```

```
Smarket_test.shape
```

```
(252, 9)
```

Fit the training data

```
X_train , X_test = X.loc[train], X.loc[~train]
y_train , y_test = y.loc[train], y.loc[~train]
glm_train = sm.GLM(y_train , X_train, family=sm.families.Binomial())
results = glm_train.fit()
probs = results.predict(exog=X_test)
```

```
998     0.528220
999     0.515669
1000    0.522652
1001    0.513854
1002    0.498334
          ...
1245    0.483637
1246    0.506048
1247    0.516658
1248    0.516124
1249    0.508072
Length: 252, dtype: float64
```

We compare the predictions for 2005 to the actual movements of the market over that time period. We will first store the test and training labels (recall y_test is binary).

```
D = Smarket.Direction
L_train , L_test = D.loc[train], D.loc[~train]
```

Now we threshold the fitted probability at 50% to form our predicted labels.

```
labels = np.array(['Down']*len(L_test))
labels[probs > 0.5] = 'Up'
confusion_table(labels , L_test)
```

Truth Predicted	Down	Up
Down	77	97
Up	34	44

The test accuracy is about 48% while the error rate is about 52%

```
np.mean(labels == L_test), np.mean(labels != L_test)
```

```
(0.4801587301587302, 0.5198412698412699)
```

The results are rather disappointing: the test error rate is 52%, which is worse than random guessing! One would not generally expect to be able to use previous days' returns to predict future market performance.

Trying a more effective model

The p-values in our original regression were quite underwhelming since none of them were less than 0.05, our preferred level of significance. Since Lag1 and Lag2 have the lowest p-values, let's drop all the other predictors from our logistic model and check our results.

```
model = MS(['Lag1', 'Lag2']).fit(Smarket)
X = model.transform(Smarket)
X_train , X_test = X.loc[train], X.loc[~train]
glm_train = sm.GLM(y_train , X_train, family=sm.families.Binomial())
results = glm_train.fit()
probs = results.predict(exog=X_test)
labels = np.array (['Down']*len(X_test))
labels[probs >0.5] = 'Up'
confusion_table(labels , L_test)
```

Truth Predicted	Down	Up
Down	35	35
Up	76	106

Let's evaluate the overall accuracy as well as the accuracy within the days when logistic regression predicts an increase.

```
(35+106) /252 ,106/(106+76)
```

```
(0.5595238095238095, 0.5824175824175825)
```

Now the results appear to be a little better: 56% of the daily movements have been correctly predicted. It is worth noting that in this case, a much simpler strategy of predicting that the market will increase every day will also be correct 56% of the time! Hence, in terms of overall error rate, the logistic regression method is no better than the naive approach. However, the confusion matrix shows that on days when logistic regression predicts an increase in the market, it has a 58% accuracy rate. This suggests a possible trading strategy of buying on days when the model predicts an increasing market, and avoiding trades on days when a decrease is predicted. Of course one would need to investigate more carefully whether this small improvement was real or just due to random chance.

Suppose that we want to predict the returns associated with particular values of Lag1 and Lag2. In particular, we want to predict Direction on a day when Lag1 and Lag2 equal 1.2 and 1.1, respectively, and on a day when they equal 1.5 and −0.8. We do this using the predict() function.

```
newdata = pd.DataFrame ({'Lag1':[1.2 , 1.5], 'Lag2':[1.1 , -0.8]});
newX = model.transform(newdata)
results.predict(newX)
```

```
0    0.479146
1    0.496094
dtype: float64
```

Linear Discriminant Analysis

We begin by performing LDA on the Smarket data, using the function LinearDiscriminantAnalysis(), which we have abbreviated LDA(). We fit the model using only the observations before 2005.

```
lda = LDA( store_covariance=True)
```

```
LinearDiscriminantAnalysis(store_covariance=True)
```

Since the LDA estimator automatically adds an intercept, we should remove the column corresponding to the intercept in both X_train and X_test. We can also directly use the labels rather than the Boolean vectors y_train.

```
X_train , X_test = [M.drop(columns =['intercept']) for M in [X_train , X_test ]]
lda.fit(X_train , L_train)
```

```
LinearDiscriminantAnalysis(store_covariance=True)
```

Having fit the model, we can extract the means in the two classes with the means_ attribute. These are the average of each predictor within each class, and are used by LDA as estimates of µk . These suggest that there is a tendency for the previous 2 days' returns to be negative on days when the market increases, and a tendency for the previous days' returns to be positive on days when the market declines.

```
lda.means_
```

```
array([[ 0.04279022,  0.03389409],
       [-0.03954635, -0.03132544]])
```

The estimated prior probabilities are stored in the priors_ attribute. The package sklearn typically uses this trailing _ to denote a quantity estimated when using the fit() method. We can be sure of which entry corresponds to which label by looking at the classes_ attribute.

```
lda.classes_
```

```
array(['Down', 'Up'], dtype='<U4')
```

The LDA output indicates that $\hat{\pi}_{Down} = 0.492$ and $\hat{\pi}_{Up} = 0.508$.

```
lda.priors_
```

```
array([0.49198397, 0.50801603])
```

The linear discriminant vectors can be found in the scalings_ attribute:

```
lda.scalings_
```

```
array([[-0.64201904],
       [-0.51352928]])
```

These values provide the linear combination of Lag1 and Lag2 that are used to form the LDA decision rule. In other words, these are the multipliers of the elements of X = x in (4.24).

$$\delta_k = x^T \Sigma^{-1} \mu_k + \frac{\mu_k^T \Sigma^{-1} \mu_k}{2} + log(\pi_k)$$

If $-0.64 \times$ Lag1 $- 0.51 \times$ Lag2 is large, then the LDA classifier will predict a market increase, and if it is small, then the LDA classifier will predict a market decline.

```
lda_pred = lda.predict(X_test)
```

```
array(['Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up',
       'Down', 'Up', 'Up', 'Up', 'Up', 'Up', 'Down', 'Up', 'Up', 'Down',
       'Down', 'Down', 'Up', 'Down', 'Down', 'Up', 'Up', 'Up', 'Down',
       'Down', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Down', 'Down', 'Up',
       'Up', 'Up', 'Up', 'Down', 'Down', 'Up', 'Up', 'Up', 'Up', 'Up',
       'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Down',
       'Down', 'Up', 'Up', 'Down', 'Down', 'Down', 'Up', 'Up', 'Up', 'Up',
       'Up', 'Up', 'Up', 'Down', 'Up', 'Down', 'Down', 'Up', 'Up', 'Up',
       'Up', 'Up', 'Down', 'Up', 'Down', 'Down', 'Up', 'Up', 'Up', 'Up',
       'Up', 'Up', 'Down', 'Down', 'Down', 'Down', 'Up', 'Up', 'Up', 'Up',
       'Up', 'Down', 'Up', 'Up', 'Down', 'Up', 'Up', 'Up', 'Up', 'Up',
       'Up', 'Up', 'Up', 'Up', 'Up', 'Down', 'Up', 'Up', 'Up', 'Up', 'Up',
       'Up', 'Down', 'Down', 'Up', 'Up', 'Down', 'Up', 'Up', 'Down',
       'Down', 'Down', 'Up', 'Up', 'Up', 'Up', 'Up', 'Down', 'Up', 'Up',
       'Up', 'Up', 'Down', 'Down', 'Up', 'Up', 'Down', 'Down', 'Up', 'Up',
       'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up',
       'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Down', 'Down', 'Up', 'Down',
       'Down', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Down', 'Up', 'Up',
       'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Down', 'Down', 'Up', 'Up',
       'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Down', 'Down', 'Up',
       'Down', 'Up', 'Up', 'Down', 'Down', 'Up', 'Up', 'Down', 'Down',
       'Up', 'Down', 'Down', 'Up', 'Up', 'Up', 'Up', 'Down', 'Down', 'Up',
       'Up', 'Up', 'Down', 'Down', 'Down', 'Down', 'Down', 'Up', 'Up',
       'Up', 'Up', 'Down', 'Down', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up',
       'Down', 'Down', 'Up', 'Up', 'Up', 'Up', 'Up', 'Down', 'Up', 'Up',
       'Up', 'Up'], dtype='<U4')
```

As we observed in our comparison of classification methods (Section 4.5), the LDA and logistic regression predictions are almost identical.

```
confusion_table(lda_pred , L_test)
```

Truth Predicted	Down	Up
Down	35	35
Up	76	106

We can also estimate the probability of each class for each point in a training set. Applying a 50% threshold to the posterior probabilities of being in class one allows us to recreate the predictions contained in lda_pred.

```
lda_prob = lda.predict_proba(X_test)
np.all(np.where(lda_prob [:,1] >= 0.5, 'Up','Down') == lda_pred)
```

```
True
```

Above, we used the np.where() function that creates an array with value 'Up' for indices where the second column of lda_prob (the estimated posterior probability of 'Up') is greater than 0.5. For problems with more than two classes the labels are chosen as the class whose posterior probability is highest:

```
np.all( [lda.classes_[i] for i in np.argmax(lda_prob , 1)] == lda_pred )
```

```
True
```

If we wanted to use a posterior probability threshold other than 50% in order to make predictions, then we could easily do so. For instance, suppose that we wish to predict a market decrease only if we are very certain that the market will indeed decrease on that day — say, if the posterior probability is at least 90%. We know that the first column of lda_prob corresponds to the label Down after having checked the classes_ attribute, hence we use the column index 0 rather than 1 as we did above.

```
np.sum(lda_prob [:,0] > 0.9)
```

```
0
```

No days in 2005 meet that threshold! In fact, the greatest posterior probability of decrease in all of 2005 was 52.02%.

Quadratic Discriminant Analysis

Fit a QDA model to the Smarket data.

```
qda = QDA(store_covariance=True)
qda.fit(X_train , L_train)
```

```
QuadraticDiscriminantAnalysis(store_covariance=True)
```

The QDA() function will again compute means_ and priors_.

```
qda.means_ , qda.priors_
```

```
(array([[ 0.04279022,  0.03389409],
        [-0.03954635, -0.03132544]]),
 array([0.49198397, 0.50801603]))
```

The QDA() classifier will estimate one covariance per class. Here is the estimated covariance in the first class:

```
qda.covariance_[0]
```

```
array([[ 1.50662277, -0.03924806],
       [-0.03924806,  1.53559498]])
```

List all the covariances.

```
qda.covariance_
```

```
[array([[ 1.50662277, -0.03924806],
        [-0.03924806,  1.53559498]]),
 array([[ 1.51700576, -0.02787349],
        [-0.02787349,  1.49026815]])]
```

The output contains the group means. But it does not contain the coefficients of the linear discriminants, because the QDA classifier involves a quadratic, rather than a linear function. The predict() function works in exactly the same fashion as for LDA.

```
qda_pred = qda.predict(X_test)
confusion_table(qda_pred, L_test)
```

Truth Predicted	Down	Up
Down	30	20
Up	81	121

Interestingly, the QDA predictions are accurate almost 60% of the time, even though the 2005 data was not used to fit the model.

```
np.mean(qda_pred == L_test)
```

```
0.5992063492063492
```

This level of accuracy is quite impressive for stock market data, which is known to be quite hard to model accurately. This suggests that the quadratic form assumed by QDA may capture the true relationship more accurately than the linear forms assumed by LDA and logistic regression. However, we recommend evaluating this method's performance on a larger test set before betting that this approach will consistently beat the market!

```
The score() function is an alternate way of obtaining the level of accuracy.
```

```
qda.score(X_test, L_test)
```

```
0.5992063492063492
```

```
allDone();
```

```
<IPython.lib.display.Audio object>
```