Homework 3 solutions

Q1) Let's write the likelihood function:

$$\mathcal{L}(\theta) = \prod_{i=1}^{N} \left(\frac{1}{1+e^{-X_i\theta}}\right)^{Y_i} \left(1 - \frac{1}{1+e^{-X_i\theta}}\right)^{1-Y_i}$$

$$\Rightarrow -\log \mathcal{L}(\theta) = \sum_{i=1}^{N} Y_i \log\left(1+e^{-X_i\theta}\right) - (1-Y_i)\left(\log\left(e^{-X_i\theta}\right) - \log\left(1+e^{-X_i\theta}\right)\right)$$

$$= \sum_{i=1}^{N} Y_i \log\left(1+e^{-X_i\theta}\right) + (1-Y_i)\left(\log\left(1 \quad \cdot X_i\theta\right)\right)$$

if $\log(1+e^{-X_i\theta})$ is convex then $-\log\mathcal{L}(\theta)$ is convex.

$$\Rightarrow \frac{\partial}{\partial\theta_j}\log\left(1+e^{-X_i\theta}\right) = -\frac{X_{ij}}{1+e^{-X_i\theta}}$$

$$\frac{\partial^2}{\partial\theta_j^2}\log\left(1+e^{-X_i\theta}\right) = \frac{X_{ij}^2}{\left(1+e^{-X_i\theta}\right)^2}$$

$$\frac{\partial^2}{\partial\theta_j\theta_k}\log\left(1+e^{-X_i\theta}\right) = \frac{X_{ij}X_{ik}}{\left(1+e^{-X_i\theta}\right)^2}$$

$\Rightarrow$ the Hessian is $X_i X_i^T \cdot \frac{1}{(1+e^{-X_i\theta})^2}$

which is a positive-definite matrix

$\Rightarrow$ convex

Q2) Let K be the number of classes

we model $P(Y_i = k \mid X_i)$ as logistic regression with the parameter θ_k

where $\dim(\theta_k) = \dim(X_i)$

the constraint is: $\sum_{k=1}^{K} P(Y_i = k \mid X_i) = 1$

$\Rightarrow$ we only need to train $K-1$ models for $\theta_1, \ldots, \theta_{K-1}$

the total parameters size is $(K-1) \times \dim(X_i)$

a) $K = 3$

$\dim(X_i) = 3$ (included bias term)

$\Rightarrow (3-1) \times 3 = 6$

b) Categorical X_1 with 5 categories can be translate to 4 binary variables

$\Rightarrow \dim(X) = 1 + 4 + 1 = 6$

$\Rightarrow (3-1) \times 6 = 12$

3. *[Analytical question.]* Consider a problem of classifying a response Y with 3 classes, and two predictors X_1, X_2, using generative modeling. Assume that both X_1 and X_2 are continuous. For each of the following, state the model, and specify the total number of parameters:

   (a) Naïve Bayes with Gaussian distributions

   (b) Linear discriminant analysis

   (c) Quadratic discriminant analysis

Let classes $K \in \{0, 1, 2\}$, P = # Features = 2, N = # samples

Let $X_1, X_2 \in \mathbb{R}^N$

(a) Naive Bayes with Gaussian Distribution:

With Naive Bayes assumption, the predictors are conditionally independent given Y.

$$P(Y=k|X) = P(X|Y=k)P(Y=k) = P(X_1|Y=k)\,P(X_2|Y=k)\,P(Y=k)$$

$$= \frac{1}{\sqrt{2\pi\sigma_{k1}^2}} \exp\left\{-\frac{(X_1-\mu_{k1})^2}{2\sigma_{k1}^2}\right\} \cdot \frac{1}{\sqrt{2\pi\sigma_{k2}^2}} \exp\left\{-\frac{(X_2-\mu_{k2})^2}{2\sigma_{k2}^2}\right\}$$

$$= \prod_{p=1}^{P} \frac{1}{\sqrt{2\pi\sigma_{kp}^2}} \exp\left\{-\frac{(X_p-\mu_{kp})^2}{2\sigma_{kp}^2}\right\} \cdot \theta_{Y=k}$$

(Here $\theta_{Y=k} = P(Y=k)$, μ_{kp} = mean of distribution for class k & Feature p)

Total # parameters = $K * p * 2 + (K-1)$ ← Priors

$= 3 * 2 * 2 + 2 = \boxed{14}$

(b) Linear Discriminant Analysis:

$F_k(X)$ = multivariate Gaussian with same variance

$$P(y=k|X) = \frac{1}{\sqrt{2\pi|\Sigma|_k}} \exp\left\{-\frac{1}{2}(X-\mu_k)^T \Sigma_k^{-1}(X-\mu_k)\right\} \cdot \theta_{Y=K}$$

Here $\mu_k \in \mathbb{R}^p$,

$\theta_{y=k} = P(Y=k)$

For all class there is one variance, and we're learning a $\mu \in \mathbb{R}^2$ vector per class.
Also, we have to estimate 2 parameters for our prior distribution that is a Multinoulli.

$\therefore$ total # parameters $= K*P + \frac{P*(P-1)}{2} + (K-1)$

$= 3*2 + \frac{2(1)}{2} + 2 = 6+1+2$

$= \boxed{9}$

(c) Quadratic Discriminant Analysis:

The model is same as LDA, but now we have different variance per class.

$\therefore$ Total # parameters $= K\left(P + \frac{P*(P-1)}{2}\right) + (K-1)$

$= 3\left(2 + \frac{2}{2}\right) + 2$

$= 9 + 2 = \boxed{11}$

4. *[Analytical question.]* Assume you have the following training set with three binary features X_1, X_2 and X_3, and a binary response Y.

X_1	X_2	X_3	Y
0	1	1	0
1	0	0	1
1	1	0	1
0	1	1	1
1	1	0	1
1	0	1	0
0	0	1	0

(a) Estimate $P(Y = 1|X_1 = 1, X_2 = 0, X_3 = 1)$ and $P(Y = 1|X_1 = 1, X_2 = 1, X_3 = 1)$ using Bayes rule, with the naïve Bayes assumption

(b) Estimate $P(Y = 1|X_1 = 1, X_2 = 0, X_3 = 1)$ and $P(Y = 1|X_1 = 1, X_2 = 1, X_3 = 1)$ using Bayes rule, without the naïve Bayes assumption

(a) With Naive Bayes Assumption:

$$P(Y=1 \mid X_1=1, X_2=0, X_3=1) = \frac{P(X_1=1, X_2=0, X_3=1 \mid Y=1) P(Y=1)}{P(X_1=1, X_2=0, X_3=1 \mid Y=0) P(Y=0) + P(X_1=1, X_2=0, X_3=1 \mid Y=1) P(Y=1)}$$

$$\overset{N.B.}{=} \frac{P(X_1=1|Y=1)P(X_2=0|Y=1)P(X_3=1|Y=1)\ P(Y=1)}{P(X_1=1|Y=0)P(X_2=0|Y=0)(X_3=1|Y=0)P(Y=0) + P(X_1=1|Y=1)P(X_2=0|Y=1)P(X_3=1|Y=1)P(Y=1)}$$

$$= \frac{\left(\frac{3}{4}\right)\left(\frac{1}{4}\right)\left(\frac{1}{4}\right)\left(\frac{4}{7}\right)}{\left(\frac{1}{3}\right)\left(\frac{2}{3}\right)(1)\left(\frac{3}{7}\right) + \left(\frac{3}{4}\right)\left(\frac{1}{4}\right)\left(\frac{1}{4}\right)\left(\frac{4}{7}\right)}$$

Similarly,

$$P(Y=1|X_1=1,X_2=1,X_3=1) = \frac{P(X_1=1,X_2=1,X_3=1|Y=1)P(Y=1)}{P(X_1=1,X_2=1,X_3=1|Y=0)P(Y=0) + P(X_1=1,X_2=1,X_3=1|Y=1)P(Y=1)}$$

$$\overset{N.B.}{=} \frac{P(X=1|Y=1)P(X_2=1|Y=1)P(X_3=1|Y=1)\ P(Y=1)}{P(X_1=1|Y=0)P(X_2=1|Y=0)P(X_3=1|Y=0)P(Y=0) + P(X_1=1|Y=1)P(X_2=1|Y=1)P(X_3=1|Y=1)P(Y=1)}$$

$$= \frac{\left(\frac{3}{4}\right)\left(\frac{3}{4}\right)\left(\frac{1}{4}\right)\left(\frac{4}{7}\right)}{\left(\frac{1}{3}\right)\left(\frac{1}{3}\right)(1)\left(\frac{3}{7}\right) + \left(\frac{3}{4}\right)\left(\frac{3}{4}\right)\left(\frac{1}{4}\right)\left(\frac{4}{7}\right)}$$

(b) Without N.B. assumption:

$$P(Y=1|X_1=1,X_2=0,X_3=1) = \frac{P(X_1=1,X_2=0,X_3=1|Y=1)P(Y=1)}{P(X_1=1,X_2=0,X_3=1|Y=0)P(Y=0) + P(X_1=1,X_2=0,X_3=1|Y=1)P(Y=1)}$$

$$= \frac{0\left(\frac{4}{7}\right)}{\left(\frac{1}{3}\right)\left(\frac{3}{7}\right) + 0\left(\frac{4}{7}\right)} = 0$$

$$P(Y=1|X_1=1,X_2=1,X_3=1) = \frac{P(X_1=1,X_2=1,X_3=1|Y=1)P(Y=1)}{P(X_1=1,X_2=1,X_3=1|Y=0)P(Y=0) + P(X_1=1,X_2=1,X_3=1|Y=1)P(Y=1)}$$

$$= \frac{0\left(\frac{4}{7}\right)}{0\left(\frac{3}{7}\right) + 0\left(\frac{4}{7}\right)} = \text{Undefined}$$

Homework 3 - Q5 solution

a) Data generator

```
data_generator <- function() {
  X <- data.frame(X1=runif(50, 0,3), X2=runif(50, 0,3))
  Y <- sapply(1/(1 + exp(3 - X$X1 - X$X2)), function(p) rbinom(1,1,p)==1)

  return(list(Xb=as.matrix(mutate(X, b=1)), X=as.matrix(X), Y=Y))
}
```

```
d <- data_generator()
dd <- data_generator()
```

b) Train logistic regression with gradient descents

```
logistic <- function(x) sapply(x, function(i) 1/(1+exp(-i)))

logistic_gradient <- function(d, theta) {
  return(
    t(d$Xb)%*%(d$Y - logistic(d$Xb%*%theta))
  )
}
```

```
logistic_gradient_descent_batch <- function(d, alpha) {
  theta <- rep(0, dim(d$Xb)[2])

  while(TRUE) {
    grad <- logistic_gradient(d, theta)

    if (sum(abs(grad)) < 0.001)
      return(theta)

    theta <- theta + alpha*grad
  }
}

logistic_gradient_descent_stochastic <- function(d, alpha) {
  theta <- rep(0, dim(d$Xb)[2])

  for (k in 1:200) {
    for (i in 2:length(d$Y)) {
      di <- list(Xb=d$Xb[(i-1):i,], Y=d$Y[(i-1):i])
      grad <- logistic_gradient(di, theta)
      theta <- theta + alpha*grad
    }
  }

  return(theta)
}
```

```r
theta <- logistic_gradient_descent_batch(d, 0.008)
theta_stoch <- logistic_gradient_descent_stochastic(d, 0.008)
```

```r
theta
```

```
##          [,1]
## X1  0.4434797
## X2  1.1324594
## b  -2.7972619
```

```r
theta_stoch
```

```
##          [,1]
## X1  0.4565995
## X2  1.0703508
## b  -2.6285970
```

The results obtained by batch and stochastic are very comparable.

c) Linear Discriminant Analysis

In this part, we will build three functions: train LDA parameters, predict the outcomes given the trained LDA parameters, and draw decision boundary

```r
lda_train <- function(d) {
  estimators <- function(class) {
    p <- sum(d$Y == class)/length(d$Y)
    m <- apply(d$X[d$Y == class, ], 2, mean)
    c <- matrix(
      apply(
        apply(d$X[d$Y == class, ], 1, function(r) (r-m)%*%t(r-m)),
        1, sum
      ),
      nrow=2
    )

    return(list(p=p,m=m,c=c))
  }

  est0 <- estimators(FALSE)
  est1 <- estimators(TRUE)
  cov <- 1/(length(d$Y) - 2)*(est0$c + est1$c)

  return(list(est0=est0, est1=est1, cov=cov))
}

lda_predict <- function(X, est) {
  score0 <- apply(X, 1, function(x) {
    -0.5*(x-est$est0$m) %*% solve(est$cov) %*% (x-est$est0$m) +
    log(est$est0$p)
  })
  score1 <- apply(X, 1, function(x) {
    -0.5*(x-est$est1$m) %*% solve(est$cov) %*% (x-est$est1$m) +
    log(est$est1$p)
  })
```

```
  return(score1 > score0)
}

lda_boundary_line <- function(est) {
  md <- est$est1$m - est$est0$m
  b <- -0.5*(
      est$est1$m%*%solve(est$cov)%*%est$est1$m -
      est$est0$m%*%solve(est$cov)%*%est$est0$m
    ) + log(est$est1$p/est$est0$p)
  t <- solve(est$cov)%*%md

  return(c(t, b))
}
```

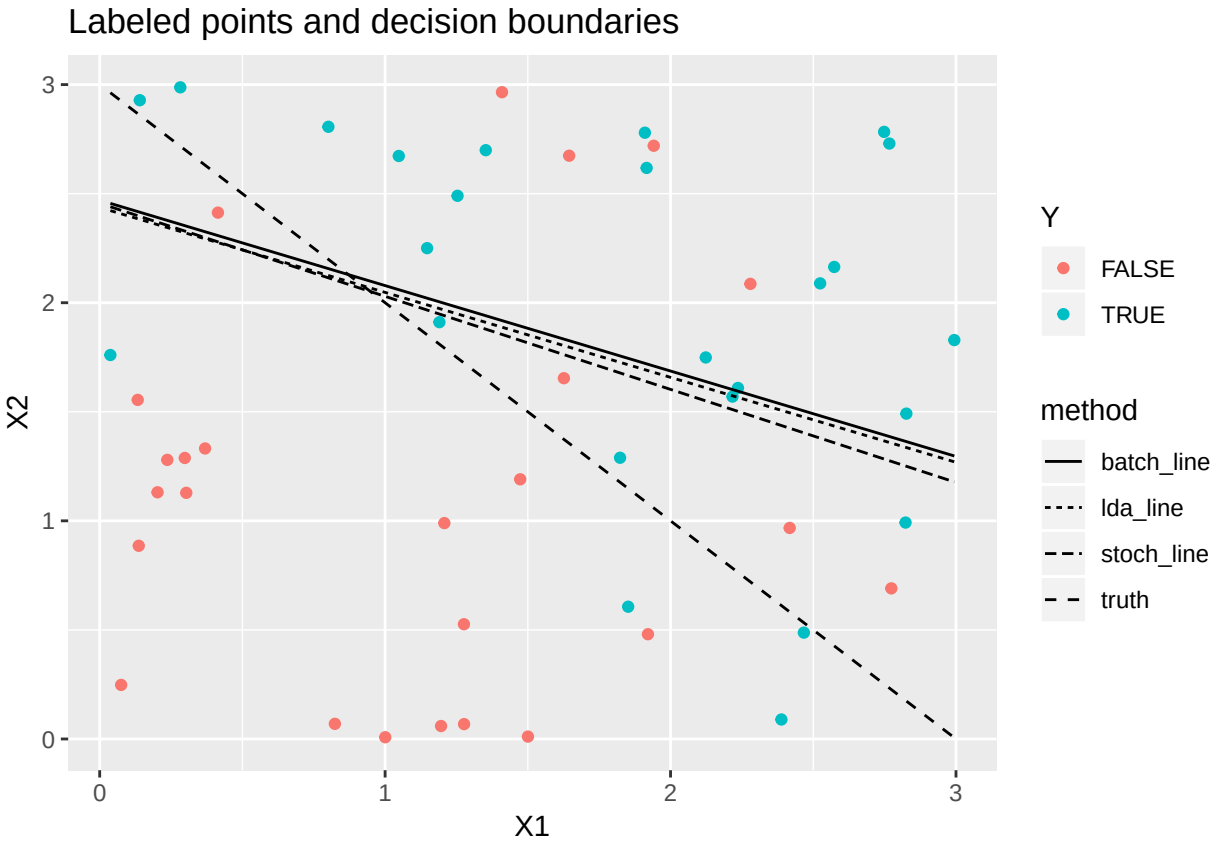
```
est <- lda_train(d)
```

d) Comparison

```
lda_bound <- lda_boundary_line(est)

df.plotting <- data.frame(dd$X) %>%
  mutate(
    Y=dd$Y,
    truth=3-X1,
    batch_line=-(theta[3]+theta[1]*X1)/theta[2],
    stoch_line=-(theta_stoch[3]+theta_stoch[1]*X1)/theta_stoch[2],
    lda_line=-(lda_bound[3]+lda_bound[1]*X1)/lda_bound[2]
  )
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
lines <- df.plotting %>% select(
    truth,
    batch_line,
    stoch_line,
    lda_line
  ) %>% gather(key="method") %>% mutate(X1=rep(df.plotting$X1, 4))

df.plotting %>%
  ggplot() +
    geom_point(aes(x=X1, y=X2, color=Y)) +
    geom_line(data=lines, aes(linetype=method, x=X1, y=value)) +
    ggtitle("Labeled points and decision boundaries")
```



The graph suggests that the three implementations are very comparable. This also suggests that their performances will be very similar given this data generator.

```
obtain_accuracies <- function(d, dd) {
  theta <- logistic_gradient_descent_batch(d, 0.008)
  theta_stoch <- logistic_gradient_descent_stochastic(d, 0.008)

  Y_lda_pred <- lda_predict(dd$X, lda_train(d))
  Y_batch <- logistic(dd$Xb%*%theta) > 0.5
  Y_stoch <- logistic(dd$Xb%*%theta_stoch) > 0.5
  Y_truth <- logistic(dd$Xb%*%c(1,1,-3)) > 0.5

  return(c(
    sum(Y_batch == dd$Y)/length(dd$Y),
    sum(Y_stoch == dd$Y)/length(dd$Y),
    sum(Y_lda_pred == dd$Y)/length(dd$Y),
    sum(Y_truth == dd$Y)/length(dd$Y)
  ))
}

obtain_accuracies(d, dd)
```

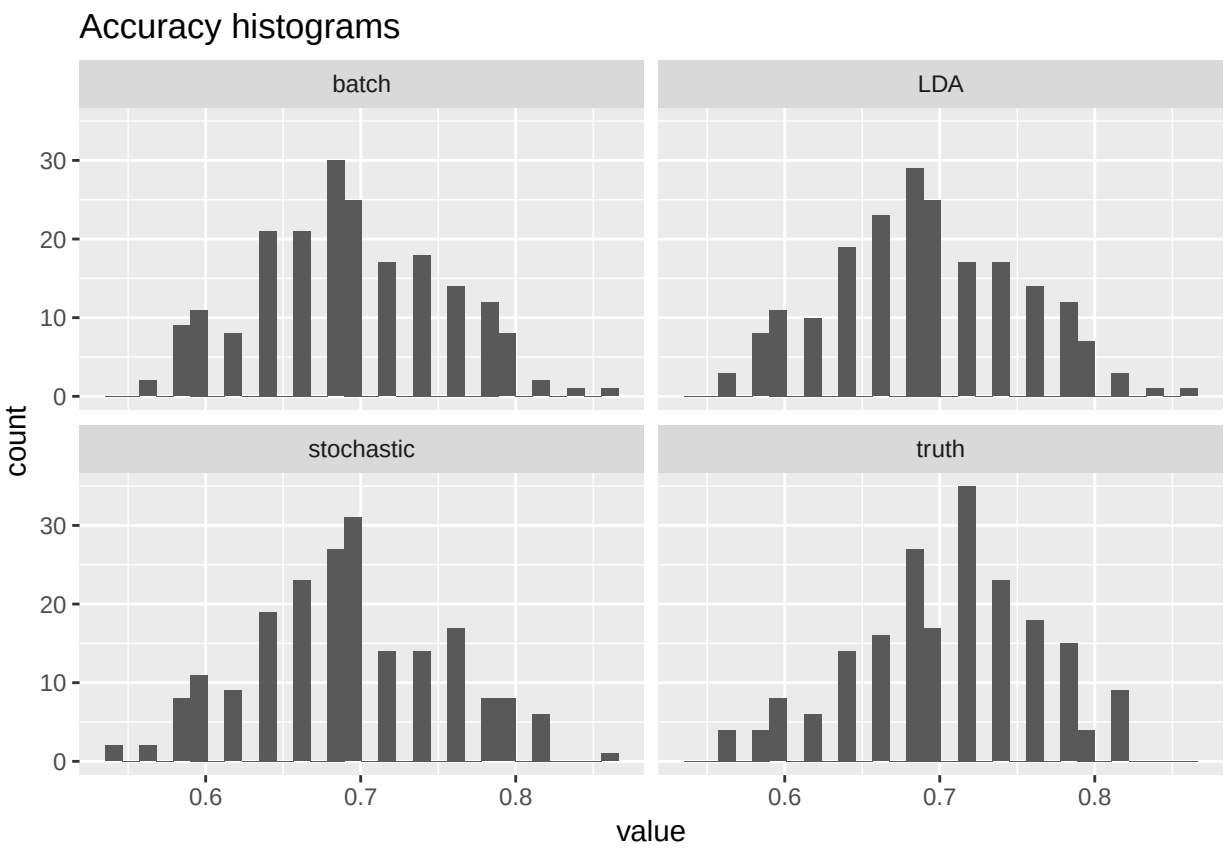
```
## [1] 0.74 0.76 0.74 0.78
```

As suggested, their predicted performance are very similar. (Extra) The last number is accuracy given that we predict with the truth function. This helps us see the upper bound of the accuracy since predicting with the truth function is the best prediction we can get.

e) Repeated simulation

```
data.frame(t(sim_res)) %>%
  rename(batch=X1, stochastic=X2, LDA=X3, truth=X4) %>%
  gather() %>% ggplot() + geom_histogram(aes(x=value)) + facet_wrap(~key) +
  ggtitle("Accuracy histograms")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Through 200 repeated iterations, we can see that the three implementations are almost identical given how the dataset is generated. Note that, if the dataset is generated with a different mechanism, the similarity beteween Logistic Regression and LDA are not guaranteed.

We can also observe that the performances of the three implementations are also very close the the performance of prediction given the truth function. Technically, the accuracy is the function of the dataset, hence it is a random variable. In this cass, the true distribution of the accuracy is given in the truth histogram. We can see that the accuracy from other impementations estimate this true distribution very well.

6. Case Study

The dataset for this homework is posted on Piazza. It documents 482 initial public offerings (IPOs) of private companies. The goal is to predict which companies attract venture capital funding.

In [47]:
```
import math
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import roc_auc_score,roc_curve,auc

import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="ticks")

% matplotlib inline
```

In [48]:
```
data = pd.read_csv("data.csv")
data.shape
del data['Id']
data.shape

y = data['vc']
del data['vc']
X = data
```

(a) Partition and Exploration

Partition the dataset into a training, developmetn adn a validation subsets of equal size, by randomly selecting rows in the dataset.

Partition:

In [49]:
```
def split_data(X,y):

    X_train, X_rem, y_train, y_rem = train_test_split(X, y, train_size = 0.3333, random_state=42)
    X_dev, X_val, y_dev, y_val = train_test_split(X_rem, y_rem, train_size = 0.5, random_state=42)


    return X_train.reset_index(drop=True), y_train.reset_index(drop=True), X_dev.reset_index(drop=Tru
e), y_dev.reset_index(drop=True), X_val.reset_index(drop=True), y_val.reset_index(drop=True)
```

In [50]:
```
X_train, y_train, X_dev, y_dev, X_val, y_val = split_data(X,y)

print(X_train.shape)
print(X_dev.shape)
print(X_val.shape)
```

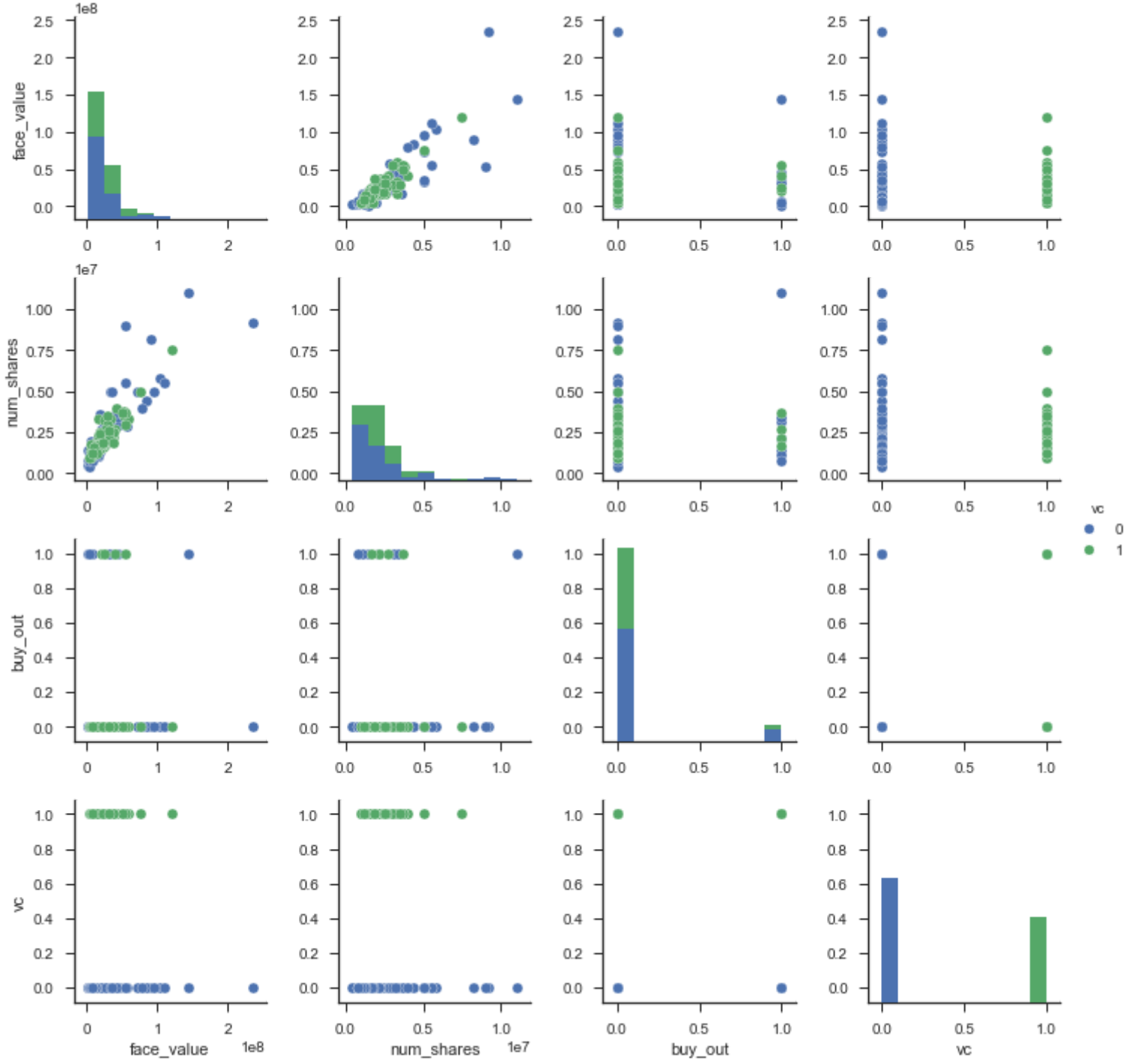
```
(160, 3)
(161, 3)
(161, 3)
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_split.py:2026: FutureWarning: From v
ersion 0.21, test_size will always complement train_size unless both are specified.
  FutureWarning)
```

Exploration & Feature Transformations:

Explore the training set: report one-variable summary statistics, two-variable summary statistics, and discuss your findings.

In [51]:
```
Train = pd.concat([X_train,y_train],axis=1)
ploot = sns.pairplot(Train, hue="vc")
```


face_value
num_shares
buy_out
vc
1e8
1e7
vc
0
1

Interestingly, from the plot Face Value vs. num_shares, vc = 1 when the face_value and number of shares is lower. It makes sense as VCs like to fund small startups/companies rather than the big ones.

This dataset is tricky, as no variable is linearly separating VC. so, if we just train classifier with these variables, it will not perform well. Let's include squares of continuous features and see how they distinguishes two classes.

Both continuous variables have very large values, we can scale them by StandardScaler (min-max normalization):

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Also, let's create dummies for buy_out column.

In [52]:
```
def standard_scaler(X,min_,max_):
    return (X - min_)/(max_ - min_)

# Including squares
def squares(X_train,X_dev,X_val,cols=None):
    for col in cols:
        X_train[col+"_2"] = np.square(X_train[col])
        X_dev[col+"_2"] = np.square(X_dev[col])
        X_val[col+"_2"] = np.square(X_val[col])

    return X_train, X_dev, X_val

X_train,X_dev,X_val = squares(X_train, X_dev, X_val, cols = ['face_value','num_shares'])

X_train['value_per_share'] = X_train['face_value']/X_train['num_shares']
X_dev['value_per_share'] = X_dev['face_value']/X_dev['num_shares']
X_val['value_per_share'] = X_val['face_value']/X_val['num_shares']

# scaling face-value
fv_min = X_train['face_value'].min()
fv_max = X_train['face_value'].max()

X_train['face_value'] = standard_scaler(X_train['face_value'],fv_min,fv_max)
X_dev['face_value'] = standard_scaler(X_dev['face_value'],fv_min,fv_max)
X_val['face_value'] = standard_scaler(X_val['face_value'],fv_min,fv_max)

# scaling num_shares
sh_min = X_train['num_shares'].min()
sh_max = X_train['num_shares'].max()

X_train['num_shares'] = standard_scaler(X_train['num_shares'],fv_min,fv_max)
X_dev['num_shares'] = standard_scaler(X_dev['num_shares'],fv_min,fv_max)
X_val['num_shares'] = standard_scaler(X_val['num_shares'],fv_min,fv_max)

# scaling face_value_2
fv_min = X_train['face_value_2'].min()
fv_max = X_train['face_value_2'].max()

X_train['face_value_2'] = standard_scaler(X_train['face_value_2'],fv_min,fv_max)
X_dev['face_value_2'] = standard_scaler(X_dev['face_value_2'],fv_min,fv_max)
X_val['face_value_2'] = standard_scaler(X_val['face_value_2'],fv_min,fv_max)
```

```
# scaling num_shares_2

sh_min = X_train['num_shares_2'].min()
sh_max = X_train['num_shares_2'].max()

X_train['num_shares_2'] = standard_scaler(X_train['num_shares_2'],fv_min,fv_max)
X_dev['num_shares_2'] = standard_scaler(X_dev['num_shares_2'],fv_min,fv_max)
X_val['num_shares_2'] = standard_scaler(X_val['num_shares_2'],fv_min,fv_max)


# buy_out dummies
X_train = pd.get_dummies(X_train,columns=['buy_out'])
X_dev = pd.get_dummies(X_dev,columns=['buy_out'])
X_val = pd.get_dummies(X_val,columns=['buy_out'])
```

Let's add few interaction variables and see if we're getting any edge.

Consider following interction:

$$value\_per\_share = \frac{face\_value}{num\_shares}$$

Let's also consider log transformation to reduce skewness as well as the effect of outliers.

In [53]:
```
def convert_log(X_train,X_dev,X_val,cols=None):
    for col in cols:
        X_train[col] = np.log1p(X_train[col])
        X_dev[col] = np.log1p(X_dev[col])
        X_val[col] = np.log1p(X_val[col])

    return X_train,X_dev,X_val

X_train,X_dev,X_val = convert_log(X_train,X_dev, X_val, cols = ['value_per_share','face_value','num_s
hares'])
```

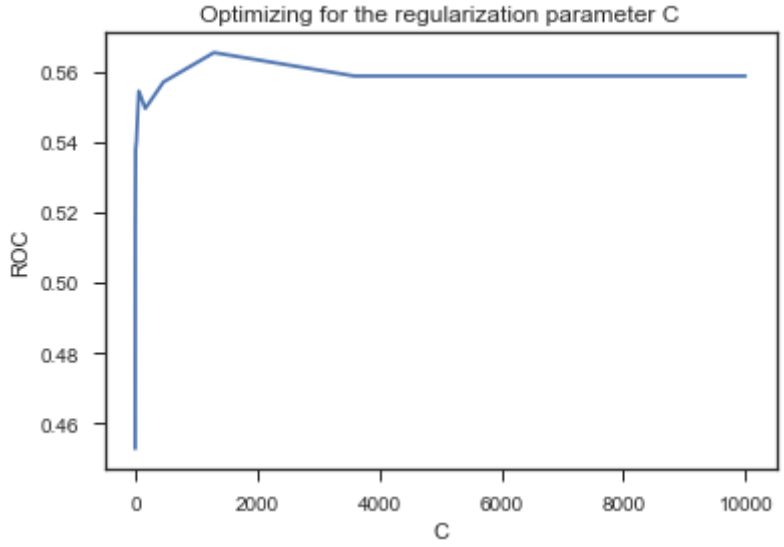
Key Points

- There are no features completely discriminating the two classes.
- The continuous variables contain very large values and are skewed.
- Very small number of training samples are available for analysis.

(b) Logistic Regression

Fit logistic regression on the training set. Consider transformations of variables, and the inclusion of higher-order terms if needed. Select the model with the best area under the ROC curve on the development set.

In [69]:
```
scores = {}
for c in np.logspace(0, 4, 10):
    clf = LogisticRegression(random_state=0, solver='lbfgs',C = c,
                             multi_class='ovr').fit(X_train,y_train)
    scores[c] = roc_auc_score(y_dev,clf.predict(X_dev))

lists = sorted(scores.items()) # sorted by key, return a list of tuples

x, y = zip(*lists) # unpack a list of pairs into two tuples

plt.plot(x, y)
plt.xlabel("C")
plt.ylabel("ROC")
plt.title("Optimizing for the regularization parameter C")
plt.show()
```



In [55]:
```
print("Best AUC score on development set: ", scores[max(scores, key=scores.get)])
print("C: ", max(scores, key=scores.get))

lr = LogisticRegression(random_state=0, solver='lbfgs',C = max(scores, key=scores.get),
                        multi_class='ovr').fit(X_train,y_train)
```

```
Best AUC score on development set:  0.565426356589
C:  1291.54966501
```

In [56]:
```
print(X_train.columns)
print(lr.coef_)
```

```
Index(['face_value', 'num_shares', 'face_value_2', 'num_shares_2',
       'value_per_share', 'buy_out_0', 'buy_out_1'],
      dtype='object')
[[ 11.29856757  -5.0989456  -24.1676772   -0.8249028    0.76817686
   -0.03350996  -0.69554026]]
```

In logistic regression, coeff > 0 means positive association with P{success}, and coeff < 0 means negative association (take a look at notes for the interpretation of betas in terms of log odds ratios). As we can see here, our new interaction variable has positive coefficient. It makes sense as the value per share increases, the P(success) (The chance of getting funding) increases. Interestingly, Number of shares has negative coefficient that tells that it is negatively associated with probability of getting VC funding.

(c) Linear Discriminant Analysis

Fit linear discriminant analysis on the training set. Consider transformations of variables, and the inclusion of higher-order terms if needed. Select the model with the best area under the ROC curve on the development set.

For LDA, we can only train the model based on continuous features as we assume that they're normally distributed.

In [57]:
```
X_train_cont = X_train.copy()
X_dev_cont = X_dev.copy()
X_val_cont = X_val.copy()

X_train_cont.drop(['buy_out_0', 'buy_out_1'], axis=1,inplace=True)
X_dev_cont.drop(['buy_out_0', 'buy_out_1'], axis=1,inplace=True)
X_val_cont.drop(['buy_out_0', 'buy_out_1'], axis=1,inplace=True)
```

In [58]:
```
clf = LinearDiscriminantAnalysis()

clf.fit(X_train_cont,y_train)

roc_auc_score(y_dev,clf.predict(X_dev_cont))
```

Out[58]: 0.58798449612403103

We performed hyperparameter optimization for regularization in logistic regression. Without any hyperparameter optimization, LDA performed much better than Logistic Regression. Note that, it didn't even use the buy_out categorical feature.

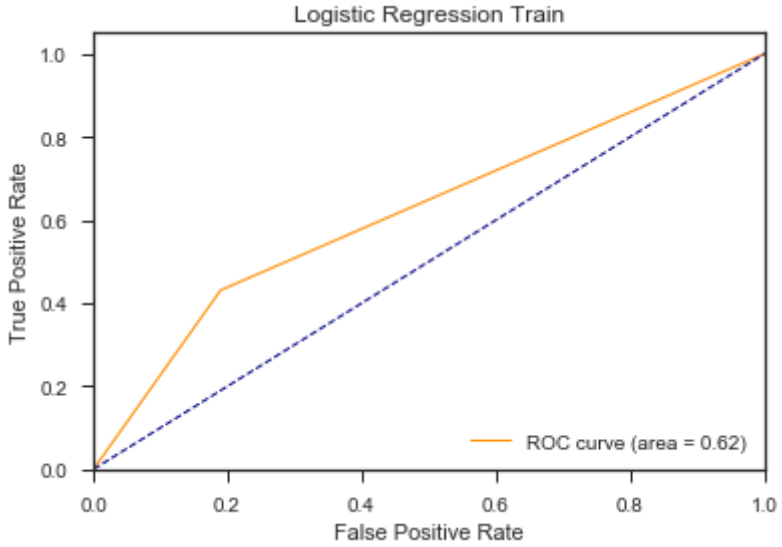
(d) ROC curves

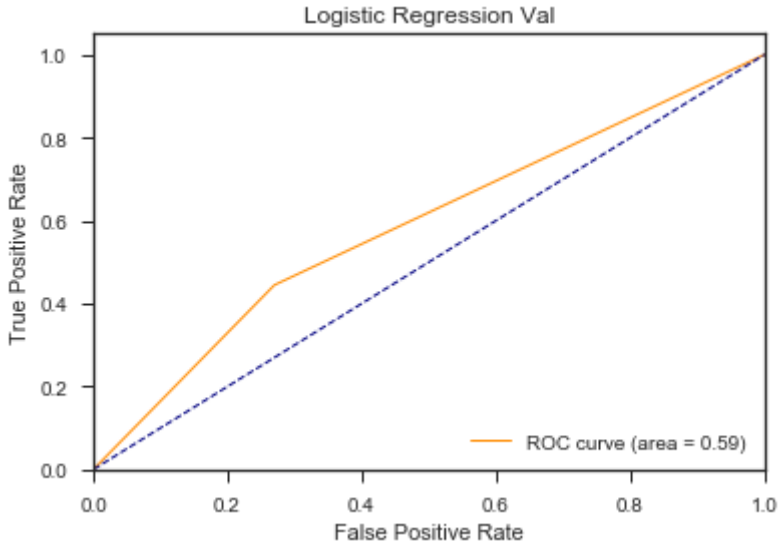
Evaluate the performance of the classifiers using ROC curves on the training and on the validation set.

In [59]:
```
def plot_roc(y_true,y_pred, title):
    fpr, tpr, thresholds = roc_curve(y_true,y_pred)

    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc="lower right")
    plt.show()
```

Logistic Regression Train:

In [60]:
```
plot_roc(y_train,lr.predict(X_train),"Logistic Regression Train")
plot_roc(y_val,lr.predict(X_val),"Logistic Regression Val")
```

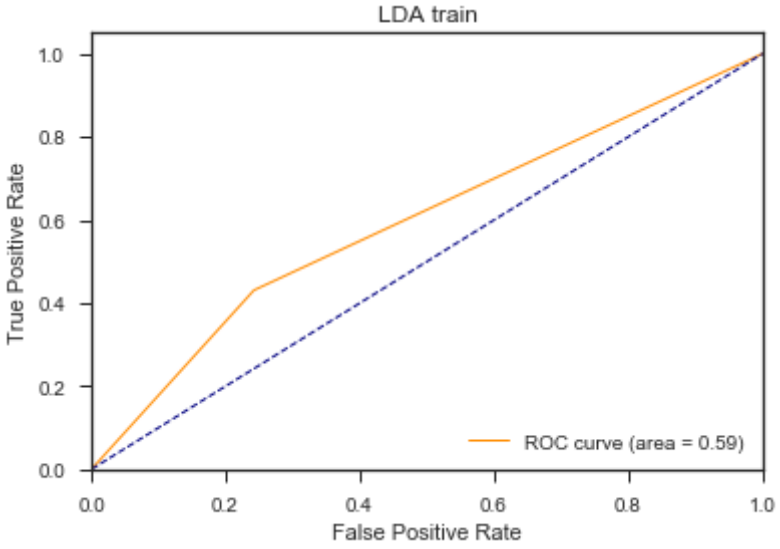


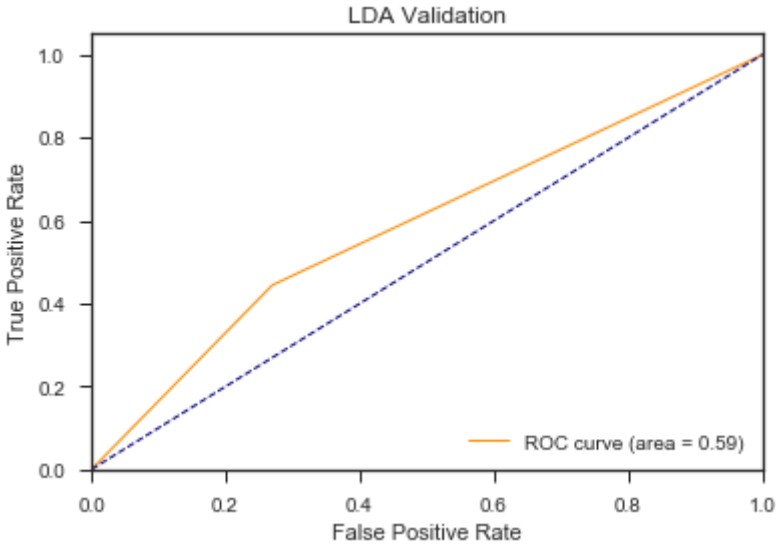


Logistic Regression Val:

LDA Train:

In [61]:
```
plot_roc(y_train,clf.predict(X_train_cont),"LDA train")
plot_roc(y_val,clf.predict(X_val_cont),"LDA Validation")
```





(e) Summary

Summarize your findings. How do the results differ between the training and the validation set? Which approach(es) perform(s) better on the validation set? What is are the reasons for this difference in performance? Which models are more interpretable?

As we can see, logistic regression has better training ROC curve than validation. It tells that the model learned is not generalized enough. However, the performance of both models are similar on validation set. But given these results, one would prefer LDA as there is no difference between train and validation set which refers that model was able to capture the distributions well enough.

Interpretibility:

In logistic regression we can more easily do subset selection - this allows us to identify the predictive features. In LDA we always have to have linear combinations of everything, which gives less insight. Also, in LDA you have to ignore the categorical predictors.

Also, we were able to infer the association in logistic regression that is not possible with LDA.