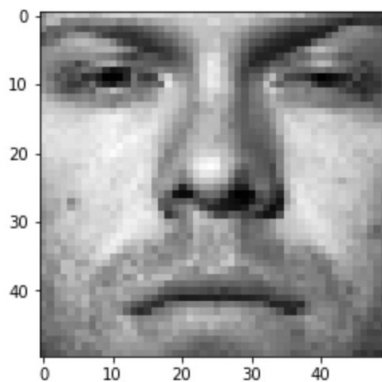


### Eigenface for face recognition.

(python code written in 3.x version)

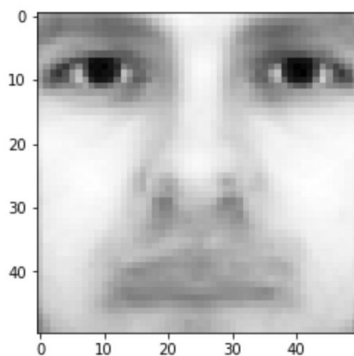
(b) The output shape of the train\_data, train\_labels, and the face image are displayed below:

(540, 2500) (540,)



(c) Average Face:

```
avg_face = train_data.mean(axis=0)
plt.imshow(avg_face.reshape(50,50), cmap = cm.Greys_r)
plt.show()
```



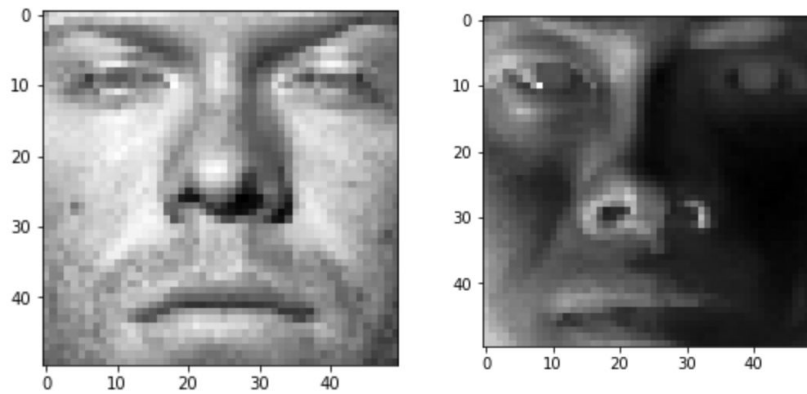
(d) Mean Subtraction:

The first image is the 10th face in the training data after subtracting the average face value, while the image on the right is the 10th face in the testing data after subtracting the average face value.

```

new_train_data = train_data
for row in new_train_data:
    for i in range(len(row)):
        row[i] = row[i] - avg_face[i]
plt.imshow(new_train_data[10, :].reshape(50,50), cmap = cm.Greys_r)

```

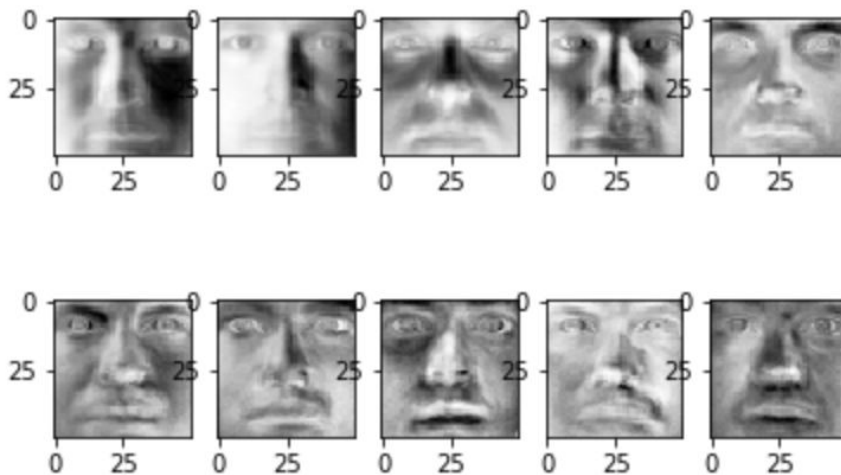


(e) First 10 Eigenfaces after SVD:

```

u, s, vh = np.linalg.svd(test_data, full_matrices=True)
count = 1
for i in range(10):
    ax = plt.subplot(2, 5, count)
    ax.imshow(vh[i, :].reshape(50,50), cmap = cm.Greys_r)
    count += 1

```



(f) Low-rank Approximation.

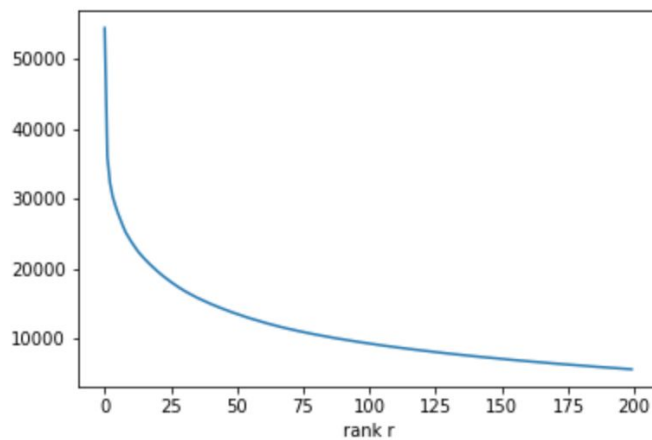
rank-r approximation error:

```
def plot_error():
    u, s, vh = np.linalg.svd(train_data, full_matrices=True)
    X = np.mat(train_data)
    error = []

    for r in range(1, 201):
        u_r = np.mat(u[:, :r])
        s_r = np.mat(np.diag(s)[:r, :r])
        vh_r = np.mat(vh[:r, :])
        X_r = u_r * s_r * vh_r
        e = np.linalg.norm(X - X_r)
        error.append(e)

    plt.plot(error)
    plt.xlabel('rank r')
    plt.show()
```

plot\_error()



(g) Eigenface Feature.

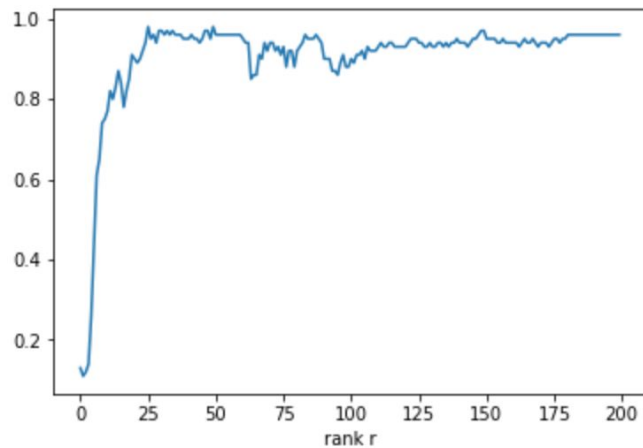
```
def train_face(vh, r):
    X = train_data
    return X.dot(vh[:r, :].T)

def test_face(vh, r):
    X = test_data
    return X.dot(vh[:r, :].T)
```

(h) Face Recognition.

```
logreg = LogisticRegression()
logreg.fit(train_face(vh, 10), train_labels)
score_log = logreg.score(test_face(vh_test, 10), test_labels)
score_log
```

We train the dataset using logistic regression with rank 10 and get the score of 0.75. The image below is the accuracy between rank 1 and rank 200.

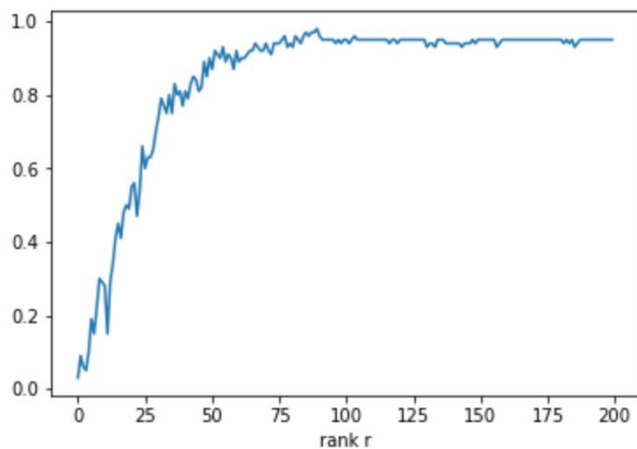


We train the dataset using one vs rest classifier and get a score of 0.29 at rank 10.

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC

ovr = OneVsRestClassifier(LinearSVC(random_state=0))
ovr.fit(train_face(vh, 10), train_labels)
score_ovr = ovr.score(test_face(vh_test, 10), test_labels)
score_ovr
```

We are expecting higher accuracy with higher ranks:



## What's Cooking?

(python code written in 2.x version)

- (a) Joined the What's Cooking competition on Kaggle and downloaded its datasets.
- (b) There are 39774 samples (dishes), 20 categories (types of cuisine), and 6714 unique ingredients in the training set.
- (c) Represent each dish by a binary ingredient feature vector.  
Use  $n \times d$  feature matrix to represent all the dishes in training set and test set, where  $n$  is the number of dishes.

```
train_matrix = []
for item in train_data.ingredients:
    feature_vtr = []
    for igd in unique_igd:
        if igd in item:
            feature_vtr.append(1)
        else:
            feature_vtr.append(0)
    train_matrix.append(feature_vtr)
```

```
train_np = np.array(train_matrix)
print train_np.shape
```

(39774, 6714)

- (d) Using Naïve Bayes Classifier to perform 3 fold cross-validation  
Prior assumption with Gaussian distribution: (Using sklearn library)

```
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.model_selection import cross_val_score

# Naive Bayes Classifier with Gaussian distribution
clf_gaus = GaussianNB()
val_gaus = cross_val_score(clf_gaus, train_np, cuisines, cv=3)
```

Prior assumption with Bernoulli distribution:

```
# Naive Bayes Classifier with Bernoulli distribution
clf_brnl = BernoulliNB()
val_brnl = cross_val_score(clf_brnl, train_np, cuisines, cv=3)
```

Calculate average accuracies:

```
# average the 3 scores from validations
print "Gaussian: accuracy is ", val_gaus.mean()
print "Bernoulli: accuracy is ", val_brnl.mean()
```

Gaussian: accuracy is 0.3794937938208349

Bernoulli: accuracy is 0.6835876576455521

(e)

Bernoulli prior performs better than Gaussian prior in terms of cross-validation accuracy.

Because we use binary encode to get ingredient vectors, which only have two status (either 1 or 0), it is more reasonable to use Bernoulli prior in this case, comparing with Gaussian prior, which would perform better with continuous data.

(f) Using Logistic Regression Model to perform 3 fold cross-validation

```
from sklearn.linear_model import LogisticRegression
clf_logr = LogisticRegression()
val_logr = cross_val_score(clf_logr, train_np, cuisines, cv=3)

print val_logr
print "Logistic Regression: accuracy is ", val_logr.mean()
```

Logistic Regression: average accuracy is 0.7757586704089684


This is better than the previous classifier.

(g) Use Logistic Regression model to get predictions on test data, and submit to kaggle.

Code:

```
clf_logr.fit(train_np, cuisines)
res = clf_logr.predict(test_np)|
```

Kaggle Result:



### What's Cooking?

Use recipe ingredients to categorize the cuisine

1,388 teams · 3 years ago

[Overview](#) [Data](#) [Kernels](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Team](#) [My Submissions](#) [Late Submission](#)

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission.csv	just now	1 seconds	0 seconds	0.78338

Complete

[Jump to your position on the leaderboard ▾](#)

The score we get is 0.78338.

1.Ex. 4.1 Show how to solve the generalized eigenvalue problem  $\max a^T B a$  subject to  $a^T W a = 1$  by transforming to a standard eigenvalue problem.

According to Lagrange multipliers, we define  $L(a) = a^T B a + \lambda(a^T w a - 1)$

$$\frac{\partial L(a)}{\partial a} = 2Ba + \lambda(2(wa)) = 0$$

$$Ba + \lambda w a = 0$$

$$w^{-1} B a = -\lambda a$$

This is a standard eigenvalue problem.

2. Ex. 4.2

- A handwritten solution at the end.

3. SVD of Rank Deficient Matrix.

a. Compute  $M^T M$  and  $MM^T$

$$M^T M = \begin{bmatrix} 39 & 57 & 60 \\ 57 & 118 & 53 \\ 60 & 53 & 127 \end{bmatrix}$$

$$MM^T = \begin{bmatrix} 10 & 9 & 26 & 3 & 26 \\ 9 & 62 & 8 & -5 & 85 \\ 26 & 8 & 72 & 10 & 50 \\ 3 & -5 & 10 & 2 & -1 \\ 26 & 85 & 50 & -1 & 138 \end{bmatrix}$$

### b&c. Eigenvalues and Eigenvectors

$M^T M$ :

$$\begin{bmatrix} 39-\lambda & 57 & 60 \\ 57 & 118-\lambda & 53 \\ 60 & 53 & 127-\lambda \end{bmatrix}$$

$$-\lambda^3 + 284\lambda^2 - 14883\lambda$$

Eigenvalues are:

- $\lambda_1 = 0$
- $\lambda_2 = -\sqrt{5281} + 142$

$$- \lambda_3 = \sqrt{5281} + 142$$

Eigenvectors are:

$$v_1 = \begin{pmatrix} -3 \\ 1 \\ 1 \end{pmatrix}$$

$$v_2 = \begin{pmatrix} \frac{-\sqrt{5281} + 68}{219} \\ \frac{-\sqrt{5281} - 5}{73} \\ 1 \end{pmatrix}$$

$$v_3 = \begin{pmatrix} \frac{\sqrt{5281} + 68}{219} \\ \frac{\sqrt{5281} - 5}{73} \\ 1 \end{pmatrix}$$

$MM^T$  :

$$\det \begin{pmatrix} 10 - \lambda & 9 & 26 & 3 & 26 \\ 9 & 62 - \lambda & 8 & -5 & 85 \\ 26 & 8 & 72 - \lambda & 10 & 50 \\ 3 & -5 & 10 & 2 - \lambda & -1 \\ 26 & 85 & 50 & -1 & 138 - \lambda \end{pmatrix}$$

Eigenvalues are:

- $\lambda_1 = 0$
- $\lambda_2 = 7.24713$
- $\lambda_3 = 44.74929$
- $\lambda_4 = 45.01035$
- $\lambda_5 = 69.33045$
- $\lambda_6 = 214.66984$

Eigenvectors are:

For  $\lambda = 0$ , we have eigenvectors =

$$\begin{pmatrix} -\frac{20}{7} \\ \frac{2}{7} \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} -\frac{3}{7} \\ \frac{1}{7} \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -\frac{11}{7} \\ -\frac{8}{7} \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

For other values of  $\lambda$ , we have eigenvectors with all 0s.



**d. SVD**

$$M = U\Sigma V^T$$

U: eigenvector of  $MM^T$

$$U = \begin{bmatrix} 0.1649 & -0.2450 \\ 0.4716 & 0.4533 \\ 0.3365 & -0.8294 \\ 0.0033 & -0.1697 \\ 0.7982 & 0.1331 \end{bmatrix}$$

$\Sigma$ : diagonal matrix, every element is the square root of the eigenvalue

$$\Sigma = \begin{bmatrix} 14.6516 & 0 \\ 0 & 8.3264 \end{bmatrix}$$

V: eigenvector of  $M^T M$

$$V = \begin{bmatrix} 0.4262 & 0.0146 \\ 0.6150 & 0.7286 \\ 0.6634 & -0.6848 \end{bmatrix}$$

**(e) Set your smaller singular value to 0 and compute the one-dimensional approximation to the matrix M.**

$$\begin{bmatrix} 0.1649 \\ 0.4716 \\ 0.3365 \\ 0.0033 \\ 0.7982 \end{bmatrix} \cdot [14.6516] \cdot [0.4262 \quad 0.6150 \quad 0.6634]$$

$$\begin{bmatrix} 1.0297 & 1.4859 & 1.6028 \\ 2.9449 & 4.2495 & 4.5839 \\ 2.1013 & 3.0321 & 3.2707 \\ 0.0206 & 0.0297 & 0.0321 \\ 4.9844 & 7.1924 & 7.7584 \end{bmatrix}$$

Question 2.

a). From 4.9, we know that:

$$\begin{aligned} \log \frac{\Pr(G=2 | X=x)}{\Pr(G=1 | X=x)} &= \log \frac{f_2(x)}{f_1(x)} + \log \frac{\pi_2}{\pi_1} \\ &= \log \frac{\pi_2}{\pi_1} - \frac{1}{2}(\mu_2 + \mu_1)^T \Sigma^{-1}(\mu_2 - \mu_1) + x^T \Sigma^{-1}(\mu_2 - \mu_1) \\ &= \log\left(\frac{N_2}{N}\right) - \log\left(\frac{N_1}{N}\right) - \frac{1}{2} \hat{\mu}_2^T \hat{\Sigma}^{-1} \hat{\mu}_2 + \frac{1}{2} \hat{\mu}_1^T \hat{\Sigma}^{-1} \hat{\mu}_1 + x^T \Sigma^{-1}(\hat{\mu}_2 - \hat{\mu}_1) \end{aligned}$$

Therefore, the LDA rule classifies to class 2 if

$$x^T \Sigma^{-1}(\hat{\mu}_2 - \hat{\mu}_1) > \frac{1}{2} \hat{\mu}_2^T \hat{\Sigma}^{-1} \hat{\mu}_2 - \frac{1}{2} \hat{\mu}_1^T \hat{\Sigma}^{-1} \hat{\mu}_1 + \log\left(\frac{N_1}{N}\right) - \log\left(\frac{N_2}{N}\right).$$

and class 1 otherwise.

b).  $\sum_{i=1}^N (y_i - \beta_0 - \beta^T x_i)^2$

to minimize the least square criterion, we derive the above eqn in terms of  $\beta, \beta_0$ :

$$\begin{aligned} \frac{\partial RSS}{\partial \beta_0} &= -2 \sum_{i=1}^N (y_i - \beta_0 - \beta^T x_i) = 0. \quad \beta_0 = \frac{1}{N} \sum_{i=1}^N (y_i - \beta^T x_i) \\ \frac{\partial RSS}{\partial \beta} &= -2 \sum_{i=1}^N x_i (y_i - \beta_0 - \beta^T x_i) = 0. \quad \sum_{i=1}^N x_i (y_i - \frac{1}{N} \sum_{j=1}^N (y_j - \beta^T x_j) - \beta^T x_i) = 0. \\ &\quad \sum_{i=1}^N x_i [\beta^T (x_i - \bar{x})] = \sum_{i=1}^N x_i (y_i - \frac{1}{N} \sum_{j=1}^N y_j) = \sum_{k=1}^2 \sum_{i \neq k} x_i (y_k - \frac{N_1 y_1 + N_2 y_2}{N}) \\ &\quad \text{Since } \sum_{j \neq k} x_i = N_k \hat{\mu}_k, \text{ we get } \sum_{k=1}^2 N_k \hat{\mu}_k \left( \frac{N y_k - (N_1 y_1 + N_2 y_2)}{N} \right) = \frac{N_1 N_2}{N} (y_2 - y_1) (\hat{\mu}_2 - \hat{\mu}_1) \\ &\quad \text{since } y_1 = \frac{N}{N_1}, y_2 = \frac{N}{N_2}, \text{ we have } \dots = N (\hat{\mu}_2 - \hat{\mu}_1) \end{aligned}$$

$$\begin{aligned} (N-2) \hat{\Sigma} &= \sum_{k=1}^2 \sum_{i \neq k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T \quad \text{since } x_i^T \hat{\mu}_k = x_i \hat{\mu}_k^T \quad \& \quad \sum_{j \neq k} x_i = N_k \hat{\mu}_k \\ &= \sum_{k=1}^2 \sum_{i \neq k} (x_i x_i^T - x_i \hat{\mu}_k^T + \hat{\mu}_k \hat{\mu}_k^T) = \sum_{k=1}^2 \left( \sum_{i \neq k} x_i x_i^T - N_k \hat{\mu}_k \hat{\mu}_k^T \right) \\ &= \sum_{i=1}^N x_i x_i^T - (N_1 \hat{\mu}_1 \hat{\mu}_1^T + N_2 \hat{\mu}_2 \hat{\mu}_2^T). \end{aligned}$$

$$\sum_{i=1}^N x_i [\beta^T (x_i - \bar{x})] = \sum_{i=1}^N x_i [(x_i - \bar{x})^T \beta] = \left( \sum_{i=1}^N x_i x_i^T - \sum_{i=1}^N x_i \bar{x}^T \right) \cdot \beta.$$

$$\begin{aligned} \text{Since } \sum_{i=1}^N x_i \bar{x}^T &= \left( \sum_{k=1}^2 \sum_{i \neq k} \right) \left( \frac{\sum_{j=1}^2 \sum_{i \neq k} x_k}{N} \right)^T = \frac{1}{N} (N_1 \hat{\mu}_1 + N_2 \hat{\mu}_2) (N_1 \hat{\mu}_1^T + N_2 \hat{\mu}_2^T) \\ &= [(N-2) \hat{\Sigma} + (N_1 \hat{\mu}_1 \hat{\mu}_1^T + N_2 \hat{\mu}_2 \hat{\mu}_2^T) - \frac{1}{N} (N_1 \hat{\mu}_1 + N_2 \hat{\mu}_2) (N_1 \hat{\mu}_1^T + N_2 \hat{\mu}_2^T)] \cdot \beta \end{aligned}$$

$$= [(N-2) \hat{\Sigma} + \frac{N_1 N_2}{N} (\hat{\mu}_2 \hat{\mu}_2^T - \hat{\mu}_1 \hat{\mu}_1^T - \hat{\mu}_2 \hat{\mu}_1^T + \hat{\mu}_1 \hat{\mu}_1^T)] \cdot \beta$$

so, we get  $[(N-2) \hat{\Sigma} + \frac{N_1 N_2}{N} \hat{\Sigma}_B] \beta = N (\hat{\mu}_2 - \hat{\mu}_1)$  where  $\hat{\Sigma}_B = (\hat{\mu}_2 - \hat{\mu}_1)(\hat{\mu}_2 - \hat{\mu}_1)^T$

c) from last question, we have:  $\hat{\Sigma}_B \beta = (\hat{\mu}_2 - \hat{\mu}_1)(\hat{\mu}_2 - \hat{\mu}_1)^T \beta$  where  $(\hat{\mu}_2 - \hat{\mu}_1)^T \beta$  is a scalar

So we can conclude  $\hat{\Sigma}_B \beta$  is in direction of  $(\hat{\mu}_2 - \hat{\mu}_1)$

from b), we have  $(N-2)\hat{\Sigma} \hat{\beta} + \frac{N_1 N_2}{N} \hat{\Sigma}_B \beta = N(\hat{\mu}_2 - \hat{\mu}_1)$

$$\hat{\beta} = \frac{1}{N-2} \hat{\Sigma}^{-1} \left[ N(\hat{\mu}_2 - \hat{\mu}_1) - \frac{N_1 N_2}{N} (\hat{\mu}_2 - \hat{\mu}_1)(\hat{\mu}_2 - \hat{\mu}_1)^T \beta \right] = \frac{1}{N-2} \hat{\Sigma}^{-1} (\hat{\mu}_2 - \hat{\mu}_1) \left[ N - \frac{N_1 N_2}{N} (\hat{\mu}_2 - \hat{\mu}_1)^T \beta \right]$$

Therefore, we have  $\hat{\beta} \propto \hat{\Sigma}^{-1} (\hat{\mu}_2 - \hat{\mu}_1)$

d). If we replace  $N$  with  $\frac{N_1 N_2}{N} (y_2 - y_1)$  from b) to satisfy the two class condition.

We get  $\hat{\beta} = \frac{N_1 N_2}{N} \cdot \frac{1}{N-2} \hat{\Sigma}^{-1} (\hat{\mu}_2 - \hat{\mu}_1) [(y_2 - y_1) - (\hat{\mu}_2 - \hat{\mu}_1)^T \beta] \propto \hat{\Sigma}^{-1} (\hat{\mu}_2 - \hat{\mu}_1)$ .

e) with the boundary condition  $y_i = 0$ . from c), we have  $\hat{\beta}_0 = \frac{1}{N} \sum_{i=1}^N \hat{\beta}^T x_i = \hat{\beta}^T \hat{\mu}$