CSE 541: Interactive Learning - Homework 3

Abhishek Saini

Contextual Bandits

Problem 1.1

Consider a stochastic contextual bandit environment where $(C_t)_{t=1}^n$ is a sequence of contexts sampled from distribution ξ on $\mathcal{C}$ and the rewards are $(X_t)_{t=1}^n$, where the conditional law of X_t given C_t and A_t is $P_{C_tA_t}$. The mean reward when choosing action $i \in [k]$ having observed context $c \in \mathcal{C}$ is $\mu(c,i) = \int x dP_{ci}(x)$. Let Φ be a subset of functions from $\mathcal{C}$ to $[k]$. The regret is

$$R_n = n \sup_{\phi \in \Phi} \mu(\phi) - \mathbb{E}\left[\sum_{t=1}^n X_t\right]$$

where $\mu(\phi) = \mu(c, \phi(c)) d\xi(c)$. Consider a variation of explore-then-commit, which explores uniformly at random for the first m rounds. Then define

$$\hat{\mu}(\phi) = \frac{k}{m} \sum_{t=1}^m \mathbf{1}\{A_t = \phi(C_t)\} X_t$$

For rounds $t > m$, the algorithm chooses $A_t = \phi^*(C_t)$, where

$$\phi^* = \arg\max_{\phi \in \Phi} \hat{\mu}(\phi) = \arg\max_{\phi \in \Phi} \sum_{t=1}^n \hat{X}_{t\phi(C_t)}$$

where $\hat{X}_{ti} = k\mathbf{1}\{A_t = \phi(C_t)\}X_t$. Show that when Φ is finite, then for appropriately tuned m the expected regret of this algorithm satisfies

$$R_n = O\left(n^{2/3}(k \log(|\Phi|))^{1/3}\right)$$

Lemma 1. $\mathbb{E}[\hat{\mu}(\phi)] = \mu(\phi)$

Proof.

$$\mathbb{E}[\hat{\mu}(\phi)] = \mathbb{E}[\mathbb{E}[\hat{\mu}(\phi) \mid C_t]] \qquad \text{Law of Iterated Expectations} \tag{1}$$

$\mathbb{E}[\hat{\mu}(\phi) \mid C_t]$ is a random variable of the form $g(C_t)$ and let's try to get an expression for g.

$$\begin{aligned} g(c) &= \mathbb{E}[\hat{\mu}(\phi) \mid C_t = c] \\ &= \mathbb{E}\left[\frac{k}{m}\sum_{t=1}^m \mathbf{1}\{A_t = \phi(C_t)\}X_t \,\middle|\, C_t = c\right] && \text{definition of } \hat{\mu}(\phi) \\ &= \frac{k}{m}\sum_{t=1}^m \mathbb{E}\left[\mathbf{1}\{A_t = \phi(C_t)\}X_t \mid C_t = c\right] && \text{linearity of expectation} \end{aligned} \tag{2}$$

Now, let's look at $\mathbb{E}\left[\mathbf{1}\{A_t = \phi(C_t)\}X_t \mid C_t = c\right]$

$$\begin{aligned}
\mathbb{E}\left[\mathbf{1}\{A_t = \phi(C_t)\}X_t \mid C_t = c\right] &= \mathbb{E}\left[\mathbf{1}\{A_t = \phi(c)\}X_t \mid C_t = c\right] && \text{conditioning} \\
&= \sum_{a' \in [k]} \mathbb{E}\left[\mathbf{1}\{A_t = \phi(c)\}X_t \mid C_t = c, A_t = a'\right] \mathbb{P}(A_t = a' \mid C_t = c) && \text{tower rule} \\
&= \sum_{a' \in [k]} \mathbb{E}\left[\mathbf{1}\{A_t = \phi(c)\}X_t \mid C_t = c, A_t = a'\right] \frac{1}{k} && \text{uniform exploration} \\
&= \sum_{a' \in [k]} \mathbb{E}\left[\mathbf{1}\{a' = \phi(c)\}X_t \mid C_t = c, A_t = a'\right] \frac{1}{k} && \text{conditioning} \\
&= \sum_{a' \in [k]} \mathbf{1}\{a' = \phi(c)\}\, \mathbb{E}\left[X_t \mid C_t = c, A_t = a'\right] \frac{1}{k} && \text{linearity of expectation} \\
&= \sum_{a' \in [k]} \mathbf{1}\{a' = \phi(c)\} \mu(c, a') \frac{1}{k} && \text{definition of } \mu(c, i) \\
&= \mu(c, \phi(c)) \frac{1}{k}
\end{aligned}$$

Substituting this result back in (2),

$$g(c) = \frac{k}{m} \sum_{t=1}^{m} \mu(c, \phi(c)) \frac{1}{k} = \mu(c, \phi(c))$$

Plugging this expression of $g(C_t) = \mathbb{E}[\hat{\mu}(\phi) \mid C_t]$ back in (1), we get,

$$\begin{aligned}
\mathbb{E}[\hat{\mu}(\phi)] &= \mathbb{E}[\mu(C_t, \phi(C_t))] \\
&= \mu(c, \phi(c)) d\xi(c) = \mu(\phi)
\end{aligned}$$

□

Consider $Z_t = k\mathbf{1}\{A_t = \phi(C_t)\}X_t$.
Thus, $\hat{\mu}(\phi) = \frac{1}{m}\sum_{t=1}^{m} Z_t$ and from Lemma 1, $\mathbb{E}[\hat{\mu}(\phi)] = \mu(\phi) = \frac{1}{m}\sum_{t=1}^{m} \mathbb{E}[Z_t]$. Therefore,

$$|\hat{\mu}(\phi) - \mu(\phi)| = \left| \frac{1}{m} \sum_{t=1}^{m} Z_t - \mathbb{E}[Z_t] \right|$$

We can bound this using Bernstein's Inequality.

Lemma 2. *(Bernstein's inequality). Let $X_1, \ldots, X_m$ be independent random variables such that $\frac{1}{m}\sum_{i=1}^{m} \mathbb{E}[(X_i - \mathbb{E}[X_i])^2] \leq \sigma^2$ and $|X_i| \leq B$. Then*

$$\left| \frac{1}{m} \sum_{i=1}^{m} X_i - \mathbb{E}[X_i] \right| \leq \sqrt{\frac{2\sigma^2 \log(2/\delta)}{m}} + \frac{2B\log(2/\delta)}{3m}$$

with probability at least $1 - \delta$.

Lemma 3. *Assuming $X_t \in [0, 1]$,*

i. $|Z_t| \leq k$

ii. $\frac{1}{m}\sum_{t=1}^{m} \mathbb{E}[(Z_t - \mathbb{E}[Z_t])^2] \leq k$

Proof. (i)

$$\begin{aligned}|Z_t| &= |k\mathbf{1}\{A_t = \phi(C_t)\}X_t| \\ &\le k|X_t| \\ &\le k && \text{from assumption}\end{aligned}$$

(ii)

$$\begin{aligned}\frac{1}{m}\sum_{t=1}^{m}\mathbb{E}[(Z_t - \mathbb{E}[Z_t])^2] &\le \frac{1}{m}\sum_{t=1}^{m}\mathbb{E}[Z_t^2] && \text{mean minimizes MSE} \\ &= \frac{1}{m}\sum_{t=1}^{m}\mathbb{E}[k^2\mathbf{1}\{A_t = \phi(C_t)\}X_t^2] && \text{definition of } Z_t \\ &= \frac{k^2}{m}\sum_{t=1}^{m}\mathbb{E}[\mathbf{1}\{A_t = \phi(C_t)\}X_t^2] && \text{linearity of expectation} \\ &= \frac{k^2}{m}\sum_{t=1}^{m}\mathbb{E}[\mathbb{E}[\mathbf{1}\{A_t = \phi(C_t)\}X_t^2 \mid C_t]] && \text{law of iterated expectations} \qquad (3)\end{aligned}$$

Similar to what we did in Lemma 1, $\mathbb{E}[\mathbf{1}\{A_t = \phi(C_t)\}X_t^2 \mid C_t]$ is a random variable. From the previous calculations, we can see,

$$\begin{aligned}\mathbb{E}\left[\mathbf{1}\{A_t = \phi(C_t)\}X_t^2 \mid C_t = c\right] &= \sum_{a'\in[k]}\mathbf{1}\{a' = \phi(c)\}\,\mathbb{E}\left[X_t^2 \mid C_t = c, A_t = a'\right]\frac{1}{k} \\ &\le \sum_{a'\in[k]}\mathbf{1}\{a' = \phi(c)\}\frac{1}{k} = \frac{1}{k} && \text{from assumption}\end{aligned}$$

Hence, the random variable $\mathbb{E}[\mathbf{1}\{A_t = \phi(C_t)\}X_t^2 \mid C_t]$ is always bounded by $\frac{1}{k}$. Plugging this back in (3),

$$\frac{1}{m}\sum_{t=1}^{m}\mathbb{E}[(Z_t - \mathbb{E}[Z_t])^2] \le \frac{k^2}{m}\sum_{t=1}^{m}\mathbb{E}[1/k] = k$$

□

Lemma 4. *For all $\phi \in \Phi$, with probability at least $1-\delta$,*

$$|\hat{\mu}(\phi) - \mu(\phi)| \le \sqrt{\frac{2k\log(2|\Phi|/\delta)}{m}} + \frac{2k\log(2|\Phi|/\delta)}{3m}$$

Proof. Lemma 3, gives the σ^2 and B needed to apply Bernstein's inequality to $|\hat{\mu}(\phi) - \mu(\phi)|$ for a fixed $\phi \in \Phi$. Therefore, applying Bernstein's inequality, we get,

$$|\hat{\mu}(\phi) - \mu(\phi)| \le \sqrt{\frac{2k\log(2/\delta)}{m}} + \frac{2k\log(2/\delta)}{3m}$$

with probability at least $1-\delta$.
To apply this over all $\phi \in \Phi$ (if Φ is finite), we can easily union bound this by using $\delta/|\Phi|$ in each of the inequalities for $\phi \in \Phi$ which concludes the proof. □

Lemma 5.

$$R_n = O\left(n^{2/3}(k\log(|\Phi|/\delta))^{1/3}\right)$$

Proof. Whenever $m \geq 2k\log(2|\Phi|/\delta)$, the following inequality, simplifies the inequality in Lemma 4, and we get, for all $\phi \in \Phi$, with probability at least $1-\delta$,

$$|\hat{\mu}(\phi) - \mu(\phi)| \leq \sqrt{\frac{4k\log(2|\Phi|/\delta)}{m}} \qquad \text{from lecture notes}$$

We are interested in bounding the regret at each time step. Beyond time m, we play according to the strategy, $\phi^* = \arg\max_{\phi\in\Phi} \hat{\mu}(\phi)$. Let ϕ_{opt} be the optimal strategy that maximizes $\mu(\phi)$ Therefore,

$$\begin{aligned}
\sup_{\phi\in\Phi} \mu(\phi) - \mu(\phi^*) &= \mu(\phi_{opt}) - \mu(\phi^*) \\
&= \mu(\phi_{opt}) - \hat{\mu}(\phi_{opt}) + \hat{\mu}(\phi_{opt}) - \mu(\phi^*) + \hat{\mu}(\phi^*) - \hat{\mu}(\phi^*) \\
&\leq \mu(\phi_{opt}) - \hat{\mu}(\phi_{opt}) - \mu(\phi^*) + \hat{\mu}(\phi^*) && \hat{\mu}(\phi_{opt}) - \hat{\mu}(\phi^*) \leq 0 \\
&\leq |\mu(\phi_{opt}) - \hat{\mu}(\phi_{opt})| + |-\mu(\phi^*) + \hat{\mu}(\phi^*)| && \text{triangle inequality} \\
&\leq 2\sqrt{\frac{4k\log(2|\Phi|/\delta)}{m}} && \text{w.p. at least } 1-\delta \qquad (4)
\end{aligned}$$

Let ξ denote the event when the above inequality holds.

$$\begin{aligned}
R_n &= \sum_{t=1}^{m} \mu(\phi_{opt}) - \mathbb{E}[X_t] + \sum_{t=m+1}^{n} \mathbb{E}[\mu(\phi_{opt}) - \mu(\phi^*)] \\
&\leq \sum_{t=1}^{m} 1 + \sum_{t=m+1}^{n} \mathbb{E}[\mu(\phi_{opt}) - \mu(\phi^*)] && \text{max regret while exploring} \\
&= m + \sum_{t=m+1}^{n} \mathbb{E}[\mu(\phi_{opt}) - \mu(\phi^*)] \\
&= m + \sum_{t=m+1}^{n} \mathbb{E}[\mu(\phi_{opt}) - \mu(\phi^*) \mid \xi]\,\mathbb{P}(\xi) + \mathbb{E}[\mu(\phi_{opt}) - \mu(\phi^*) \mid \xi^c]\,\mathbb{P}(\xi^c) && \text{tower rule} \\
&\leq m + \sum_{t=m+1}^{n} \mathbb{E}[\mu(\phi_{opt}) - \mu(\phi^*) \mid \xi]\,\mathbb{P}(\xi) + \mathbb{E}[\mu(\phi_{opt}) - \mu(\phi^*) \mid \xi^c]\delta && \mathbb{P}(\xi^c) \leq \delta \\
&\leq m + \sum_{t=m+1}^{n} \mathbb{E}[\mu(\phi_{opt}) - \mu(\phi^*) \mid \xi]\,\mathbb{P}(\xi) + \delta && \text{worst case regret} \\
&\leq m + \sum_{t=m+1}^{n} \mathbb{E}[\mu(\phi_{opt}) - \mu(\phi^*) \mid \xi] + \delta && \text{probability always less than 1} \\
&\leq m + \sum_{t=m+1}^{n} 2\sqrt{\frac{4k\log(2|\Phi|/\delta)}{m}} + \delta && \text{from (4)} \\
&= m + 2(n-m)\sqrt{\frac{4k\log(2|\Phi|/\delta)}{m}} + (n-m)\delta \\
&\leq m + 2n\sqrt{\frac{4k\log(2|\Phi|/\delta)}{m}} + n\delta
\end{aligned}$$

Optimizing the above bound with respect to m gives $m = n^{2/3}(4k\log(2|\Phi|/\delta))^{1/3}$. Substituting this m in the bound above, we get,

$$R_n \leq 3n^{2/3}(4k\log(2|\Phi|/\delta))^{1/3} + n\delta$$

Setting $\delta = n^{-1/3}$,

$$R_n \leq 3n^{2/3}(4k\log\left(2|\Phi|n^{1/3}\right))^{1/3} + n^{2/3} = O(n^{2/3}(k\log\left(|\Phi|n^{1/3}\right))^{1/3})$$

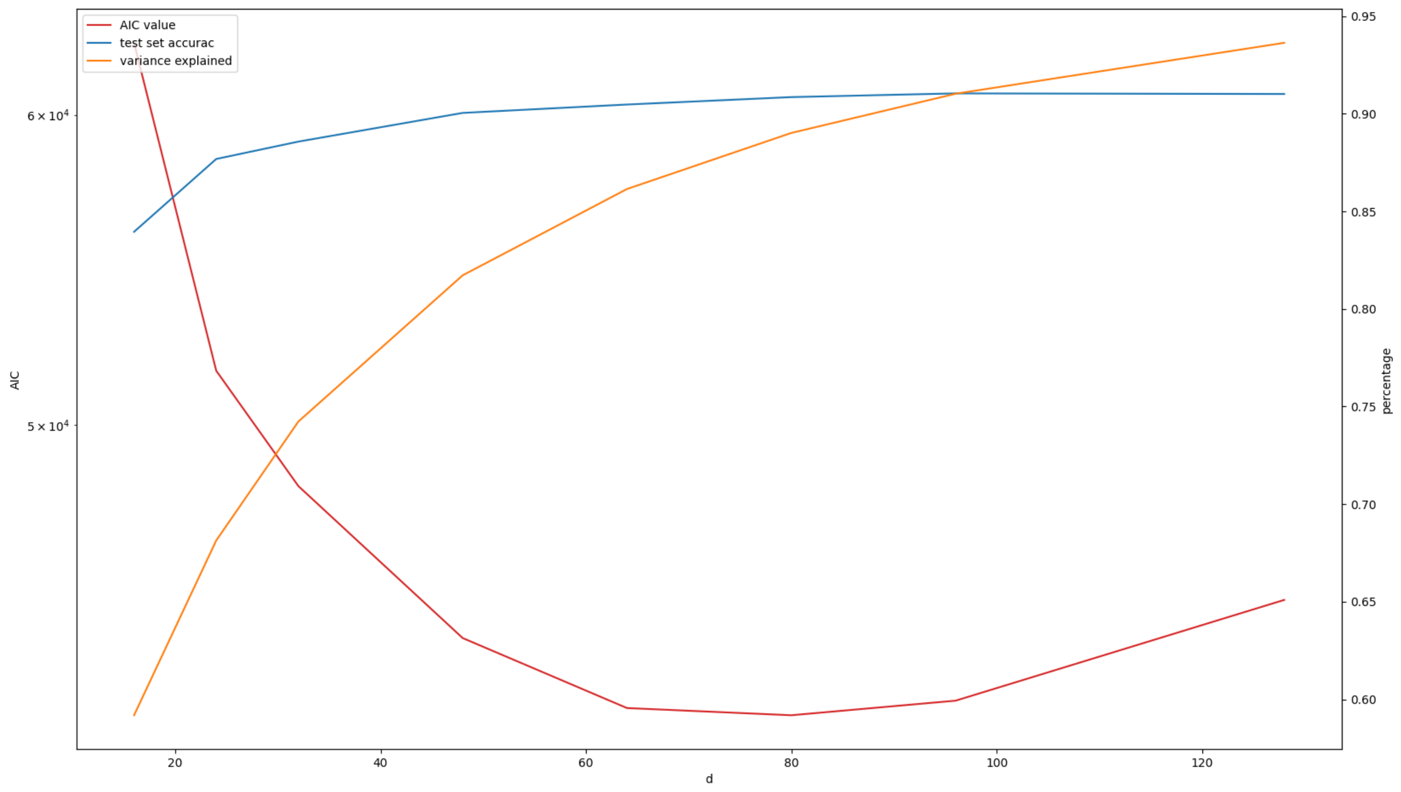
$\square$

Experiments

Data set was down-sampled without replacement to 50,000 data points with 5,000 data points for each class.

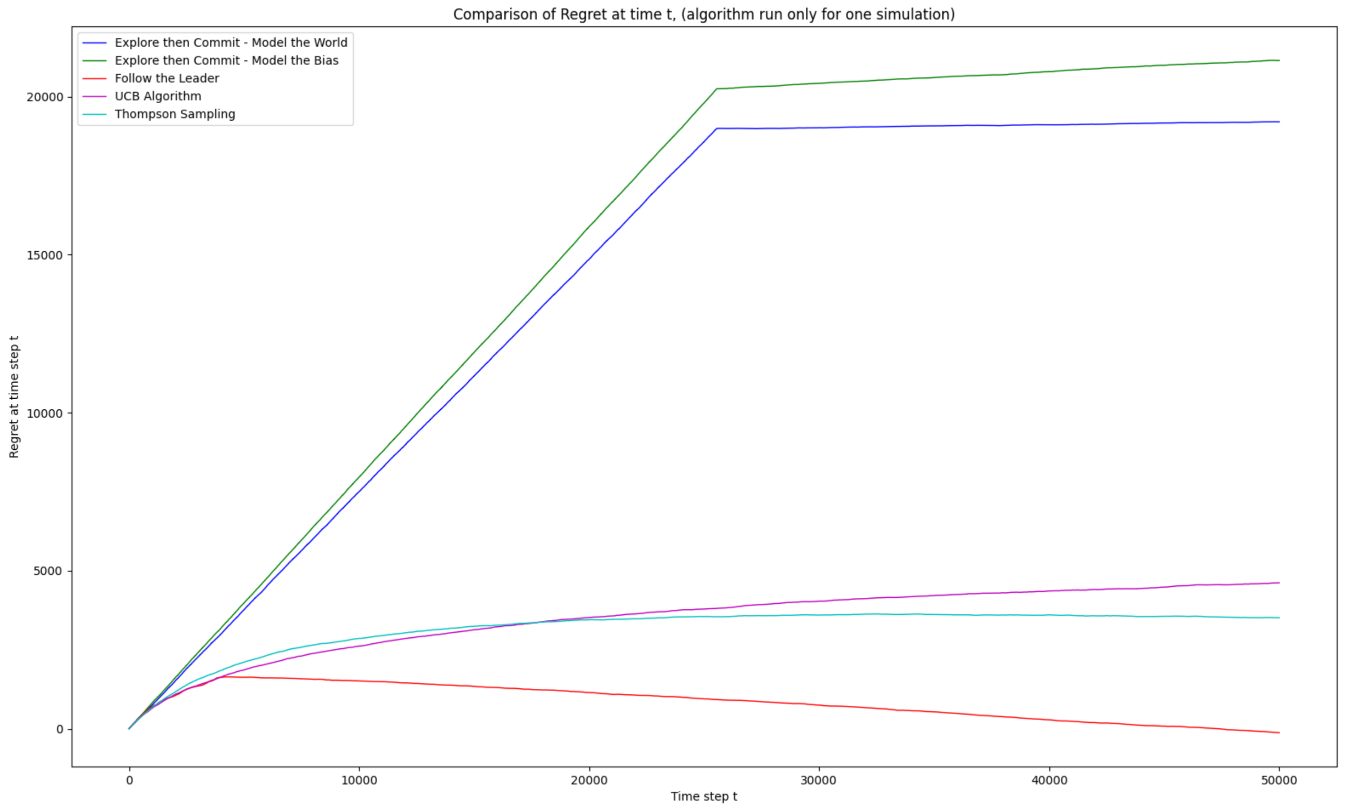
Model selection

A logistic regression model was fit for $d \in [16, 24, 32, 48, 64, 80, 96, 128]$ and the simplest model that had the low AIC, high accuracy on test set was chosen. $d = 48$ was chosen for faster computation with minimal loss in representation power.



Comparison of Algorithms

For Follow-The-Leader Algorithm, a small number of rounds was played with uniform exploration (100 rounds). This was done to make V_t invertible, beyond which the algorithm learns incrementally with each new sample. For regret calculation, a "best model in hindsight" was created. For algorithms based on linear models (ETC - Model the World, Follow-the-Leader, UCB and Thompson Sampling), the "best model in hindsight" was trained on 500,000 data points, 10 data points (corresponding to the 10 possible actions) associated with each of the 50,000 samples. For the algorithm based on the logistic regression model (ETC - Model the Bias), a logistic regression model was trained to predict the arm to be played for each context using the 50,000 data points available. Regret relative to the "best model in hindsight" is plotted for each algorithm in the graph below.



Follow-The-Leader algorithm performs better than the trained model possibly because the extra data doesn't help with learning a better representation because the linear model will have a high bias and thus, extra data doesn't guarantee improved performance.

Code

model_selection.py

```
from mnist import MNIST
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
from sklearn.decomposition import PCA
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import log_loss
from utils import load_train_downsampled, preview


def plot_model_selection(d_vals, aic_vals, test_accuracy, variance_expl):
    fig, ax = plt.subplots()
    ax.set_yscale("log")
    ln1 = ax.plot(d_vals, aic_vals, label = "AIC value", color="tab:red")
    ax.set_ylabel("AIC")
    ax.set_xlabel("d")

    ax2 = ax.twinx()
    ln2 = ax2.plot(d_vals, test_accuracy, label = "test set accurac", color="tab:blue")
    ln3 = ax2.plot(d_vals, variance_expl, label = "variance explained", color="tab:orange")
    ax2.set_ylabel("percentage")

    lines = ln1+ln2+ln3
    ax.legend(lines, [l.get_label() for l in lines], loc="upper left")

    plt.show()

train_images, train_labels = load_train_downsampled("train_downsampled.csv")

mndata = MNIST('./python-mnist/data/')
test_images, test_labels = mndata.load_testing()
test_images, test_labels = np.array(test_images), np.array(test_labels)

# preview(np.array(images), labels, 1)

count_labels = Counter(train_labels)

d_vals = [16, 24, 32, 48, 64, 80, 96, 128]
aic_vals = [0]*len(d_vals)
```

```
test_accuracy = [0]*len(d_vals)
variance_expl = [0]*len(d_vals)

for i, d in enumerate(d_vals):
    pca = PCA(n_components=d, svd_solver='full', whiten=True).fit(train_images)

    X_train_pca = pca.transform(train_images)
    variance_expl[i] = sum(pca.explained_variance_ratio_)

    clf = make_pipeline(StandardScaler(), SGDClassifier(loss='log', max_iter=1000, tol=1e-3))
    clf.fit(X_train_pca, train_labels)
    pred = clf.predict_proba(X_train_pca)

    aic_vals[i] = 2*log_loss(train_labels, pred)*X_train_pca.shape[0] + 2*d

    X_test_pca = pca.transform(test_images)
    test_accuracy[i] = clf.score(X_test_pca, test_labels)
    print(d, aic_vals[i], test_accuracy[i])

plot_model_selection(d_vals, aic_vals, test_accuracy, variance_expl)
```

downsample.py

```
import numpy as np
from mnist import MNIST
from utils import append

def balance_downsample(images, labels, N):
    n, d = images.shape
    unique_classes = np.unique(labels)
    n_classes = len(unique_classes)
    class_size = int(N/n_classes)

    data = np.c_[images, labels]
    sampled_data = None

    for c in unique_classes:
        class_indices = np.where(labels == c)[0]
        downsampled_indices = np.random.choice(class_indices, class_size, replace=False)
        sampled_data = append(sampled_data, data[downsampled_indices])

    np.random.shuffle(sampled_data)
    sampled_images, sampled_labels = sampled_data[:,:-1], np.squeeze(sampled_data[:,-1:])

    return sampled_images, sampled_labels
```

```
if __name__ == "__main__":
    mndata = MNIST('./python-mnist/data/')
    images, labels = mndata.load_training()
    images, labels = np.array(images), np.array(labels)

    sampled_images, sampled_labels = balance_downsample(images, labels, 50000)

    np.savetxt("train_downsampled.gz", np.c_[sampled_images, sampled_labels], delimiter=",")
```

reduce_dimensions.py

```
from sklearn.decomposition import PCA
from utils import load_train_downsampled
import numpy as np
import pickle

if __name__=="__main__":
    train_images, train_labels = load_train_downsampled("train_downsampled.csv")

    d = 48

    pca = PCA(n_components=d, svd_solver='full', whiten=True).fit(train_images)
    X_train_pca = pca.transform(train_images)

    # dump state of pca object for reuse
    with open('pca_48.pkl', 'wb') as output:
        pickle.dump(pca, output, pickle.HIGHEST_PROTOCOL)

    np.savetxt("train_downsampled_pca_48.csv", np.c_[X_train_pca, train_labels], delimiter=",")
```

contextual_bandit_algorithms.py

```
import math
import numpy as np
from utils import load_train_downsampled, load_pickle, preview, rescale_norm, append, plot
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
import time

def basis(i, d):
    """
    returns standard basis vector e_i in R^d with 1 at idx i and 0 everywh
    """
    e = np.zeros(d)
    e[i] = 1
    return e

def generate_Phi(C, a, k):
    tau, d = len(a), C.shape[1]
    Phi = np.zeros((tau, d*k))
    for j, action in enumerate(a):
        e_a = basis(action, k).reshape(k, 1)
        c = C[j].reshape(len(C[j]), 1)
        phi_c_a = ( c @ e_a.T ).flatten()
        Phi[j] = phi_c_a

    return Phi

def select_best_arm(c_t, theta_hat, k):
    Phi_c_t = np.einsum("i,jk->kij", c_t, np.eye(k))
    Phi_c_t = Phi_c_t.reshape(k, -1)
    return np.argmax(Phi_c_t @ theta_hat)

def commit(C, theta_hat, tau, k):
    T, d = C.shape
    commit_plays = []
    for t in range(tau, T):
        commit_plays.append( select_best_arm(C[t], theta_hat, k) )

    return commit_plays

def ETC_world(X_train, y_train):
    T, d = X_train.shape
    k = len(np.unique(y_train))

    # calculate tau
    tau = math.ceil(T**(2/3) * (2*k*(k*d+1)*math.log(2))**(1/3) )

    # explore to get theta_hat
    explore_plays = np.random.choice(np.arange(k), size=tau)
    Phi = generate_Phi(X_train_pca_norm, explore_plays, k)
    r = np.where(y_train[:tau] == explore_plays, 1, 0).reshape(tau, 1)
    theta_hat = (np.linalg.pinv(Phi.T @ Phi) @ Phi.T) @ r

    # play arg max based on theta_hat
    commit_plays = commit(X_train_pca_norm, theta_hat, tau, k)

    # calculate regret
    arms_played = np.r_[explore_plays, commit_plays]
    regret = np.where(arms_played == y_train, 0, 1)

    return np.cumsum(regret)

def ETC_bias(X_train, y_train):
    T, d = X_train.shape
    k = len(np.unique(y_train))

    # calculate tau
    tau = math.ceil(T**(2/3) * (2*k*(k*d+1)*math.log(2))**(1/3) )

    # explore to generate data for logistic classifier
    explore_plays = np.random.choice(np.arange(k), size=tau)
    indices = np.where(explore_plays == y_train[:tau])[0]
    X = X_train[indices]
    y = y_train[indices]

    # train logistic classifier on generated data
    clf = make_pipeline(StandardScaler(), SGDClassifier(loss='log', max_iter=1000, tol=1e-3))
    clf.fit(X, y)

    # play arg max based on the trained logistic classifier
    commit_plays = clf.predict(X_train[tau:])

    # calculate regret
    arms_played = np.r_[explore_plays, commit_plays]
    regret = np.where(arms_played == y_train, 0, 1)

    return np.cumsum(regret)

def FTL(C, theta_hat, V_t_inv, y_train, S_t, tau, k, Phi, r):
    T, d = C.shape
    FTL_plays = []
    t1, t2, t3 = [[] for i in range(3)]
    Phi_new = Phi
    for t in range(tau, T):
        if t%500==0: print(t)

        start_time = time.time()
        FTL_plays.append( select_best_arm(C[t], theta_hat, k) )

        e_a = basis(FTL_plays[-1], k).reshape(k, 1)
        c = C[t].reshape(len(C[t]), 1)
        phi_c_a = ( c @ e_a.T ).flatten()
        phi_c_a = phi_c_a.reshape(len(phi_c_a), 1)
        temp = V_t_inv @ phi_c_a
        mul = 1/(1+ phi_c_a.T @ temp )

        V_t_inv = V_t_inv - mul * temp @ (phi_c_a.T @ V_t_inv )
        S_t = S_t + (y_train[t] == FTL_plays[-1]) * phi_c_a
        theta_hat = V_t_inv @ S_t

    return FTL_plays

def Follow_The_Leader(X_train, y_train, tau):
    T, d = X_train.shape
    k = len(np.unique(y_train))
    gamma = 0

    # explore to get theta_hat
    explore_plays = np.random.choice(np.arange(k), size=tau)
    Phi = generate_Phi(X_train_pca_norm, explore_plays, k)
    r = np.where(y_train[:tau] == explore_plays, 1, 0).reshape(tau, 1)
    V_t_inv = np.linalg.inv(Phi.T @ Phi + gamma*np.eye(Phi.shape[1]))
    S_t = Phi.T @ r
    theta_hat = V_t_inv @ S_t

    # play follow the leader strategy
    FTL_plays = FTL(X_train_pca_norm, theta_hat, V_t_inv, y_train, S_t, tau, k, Phi, r)

    # calculate regret
    arms_played = np.r_[explore_plays, FTL_plays]
    regret = np.where(arms_played == y_train, 0, 1)

    return np.cumsum(regret)

def UCB_algorithm(X_train, y_train, gamma = 1):
    T, d = X_train.shape
    k = len(np.unique(y_train))
    delta = 1/T
    #gamma = 1
    U = 1
    V_0, S_0 = gamma * np.eye(k*d), np.zeros((k*d, 1))
    log_det_V_0 = np.log(np.linalg.det(V_0))

    I = np.zeros(T)

    V_t, S_t = V_0, S_0
    V_t_inv = np.linalg.inv(V_t)
    for t in range(T):
        if t%400==0: print(t)
        _, log_det_V_t = np.linalg.slogdet(V_t)
        beta_t = np.sqrt(gamma)*U + np.sqrt(2*np.log(1/delta) + log_det_V_t - log_det_V_0 )
        theta_t = V_t_inv @ S_t

        c_t = X_train[t]
        Phi_c_t = np.einsum("i,jk->kij", c_t, np.eye(k))
        Phi_c_t = Phi_c_t.reshape(k, -1)
        i_t = np.argmax( Phi_c_t @ theta_t + beta_t*np.sqrt(np.sum((Phi_c_t @ V_t_inv) * Phi_c_t, axis=

        # pull arm and observe
        I[t] = i_t

        # update V_t_inv, S_t
        S_t = S_t + (y_train[t] == i_t)*(Phi_c_t[[i_t]].T)
        temp = V_t_inv @ Phi_c_t[[i_t]].T
        mul = 1/(1+ Phi_c_t[[i_t]] @ temp )
        V_t_inv = V_t_inv - mul * temp @ (Phi_c_t[[i_t]] @ V_t_inv )


    regret = np.where(I == y_train, 0, 1)
    return np.cumsum(regret)

def Thompson_sampling(X_train, y_train, gamma=1):
    T, d = X_train.shape
    k = len(np.unique(y_train))
    #gamma = 1
    V_0, S_0 = gamma * np.eye(k*d), np.zeros((k*d, 1))

    Y = np.zeros(T)
    I = np.zeros(T)

    V_t, S_t = V_0, S_0
    V_t_inv = np.linalg.inv(V_t)
    for t in range(T):
        if t%400==0: print(t)

        theta_t = V_t_inv @ S_t
        theta_sample = np.random.multivariate_normal(np.squeeze(theta_t), V_t_inv)

        c_t = X_train[t]
        Phi_c_t = np.einsum("i,jk->kij", c_t, np.eye(k))
        Phi_c_t = Phi_c_t.reshape(k, -1)
        i_t = np.argmax(Phi_c_t @ theta_sample)

        # pull arm and observe
        I[t] = i_t

        # update V_t_inv, S_t
        S_t = S_t + (y_train[t] == i_t)*(Phi_c_t[[i_t]].T)
        temp = V_t_inv @ Phi_c_t[[i_t]].T
        mul = 1/(1+ Phi_c_t[[i_t]] @ temp )
        V_t_inv = V_t_inv - mul * temp @ (Phi_c_t[[i_t]] @ V_t_inv )

    regret = np.where(I == y_train, 0, 1)
    return np.cumsum(regret)

def best_linear_model_predictions(X_train, y_train):
    T, d = X_train_pca_norm.shape
    k = len(np.unique(y_train))

    actions_repeat = np.arange(T*10)%10
    Phi_train_full = generate_Phi(np.repeat(X_train, k, axis=0), actions_repeat, k)
    r_train_full = np.where(np.repeat(y_train, k, axis=0) == actions_repeat, 1, 0)

    theta_opt = (np.linalg.inv(Phi_train_full.T @ Phi_train_full) @ Phi_train_full.T) @ r_train_full

    # play arg max based on theta_opt
    optimal_plays = commit(X_train, theta_opt, 0, k)

    return optimal_plays


def best_logistic_reg_predictions(X_train, y_train):
    # create optimal logreg model
    clf = make_pipeline(StandardScaler(), SGDClassifier(loss='log', max_iter=1000, tol=1e-3))
    clf.fit(X_train, y_train)
    # play optimally in hindsight based on the trained logistic classifier
    optimal_plays = clf.predict(X_train)

    return optimal_plays

if __name__=="__main__":
    T = 50000

    # load downsample, PCA transformed data
    X_train_pca, y_train = load_train_downsampled("train_downsampled_pca_48.csv")
    pca = load_pickle("pca_48.pkl")
    X_train_pca_norm = rescale_norm(X_train_pca)

    X_train_pca_norm, y_train = X_train_pca_norm[:T], y_train[:T]

    n, d = X_train_pca_norm.shape
    k = len(np.unique(y_train))

    sim_types = [
        ('ETCW', "Explore then Commit - Model the World"),
        ('ETCB', "Explore then Commit - Model the Bias"),
        ('FTL', "Follow the Leader"),
        ('UCB', "UCB Algorithm"),
        ('TS', "Thompson Sampling"),
    ]

    mean_result, var_result, labels = [[0 for i in range(len(sim_types))] for _ in range(3)]
    for i, (key, label) in enumerate(sim_types):
        # get optimal predictions (in hindsight) for linear and logreg models
        best_linear_predictions = best_linear_model_predictions(X_train_pca_norm, y_train)
        best_logistic_predictions = best_logistic_reg_predictions(X_train_pca_norm, y_train)
        optimal_linear_regret = np.cumsum(np.where(best_linear_predictions == y_train, 0, 1))
        optimal_logistic_regret = np.cumsum(np.where(best_logistic_predictions == y_train, 0, 1))

        labels[i] = label
        if key == 'ETCW':
            regret = ETC_world(X_train_pca_norm, y_train)
            mean_result[i] = regret - optimal_linear_regret
        if key == 'ETCB':
            regret = ETC_bias(X_train_pca_norm, y_train)
            mean_result[i] = regret - optimal_logistic_regret
        if key == 'FTL':
            regret = Follow_The_Leader(X_train_pca_norm, y_train, 100)
            mean_result[i] = regret - optimal_linear_regret
        if key == 'UCB':
            regret = UCB_algorithm(X_train_pca_norm, y_train, 0.33)
            mean_result[i] = regret - optimal_linear_regret
        if key == 'TS':
            regret = Thompson_sampling(X_train_pca_norm, y_train, 0.4)
            mean_result[i] = regret - optimal_linear_regret

    plot(mean_result, var_result, labels, T)
```

utils.py

```
from copyreg import pickle
import numpy as np
import pickle
import matplotlib.pyplot as plt
import pandas as pd

def append(arr1, arr2):
    if arr1 is None: return arr2
    return np.r_[arr1, arr2]

def load_pickle(filepath):
    with open(filepath, 'rb') as inp:
        pca = pickle.load(inp)
    return pca

def load_train_downsampled(filepath):
    downsampled_data = np.loadtxt(filepath, delimiter=",")
    sampled_images, sampled_labels = downsampled_data[:,:-1], np.squeeze(downsampled_data[:,-1:])

    return sampled_images, sampled_labels

def rescale_norm(X):
    X_norm_vec = np.linalg.norm(X, axis=1, keepdims=True)
    X = X/X_norm_vec
    return X

def preview(data, labels, i):
    img = data[i].reshape((28,28))
    plt.imshow(img, cmap="Greys")
    plt.title(labels[i])
    plt.show()

def plot(mean_aggregate, var_aggregate, labels, T):
```

```
mean_aggregate = [np.array(item) for item in mean_aggregate]
std_aggregate = [np.sqrt(item) for item in var_aggregate]

x_error = np.arange(0, T+1, T/5, dtype=np.int32)
x_error[-1] = x_error[-1]-1

for i in range(len(mean_aggregate)):
    colors = ['b', 'g', 'r', 'm', 'c', 'k', 'tab:grey']
    y_error = mean_aggregate[i][x_error]
    #e = std_aggregate[i][x_error]

    time_series_df = pd.DataFrame(mean_aggregate[i])

    plt.plot(time_series_df, linewidth=1, label=labels[i], color=colors[i]) #mean curve.
    #plt.errorbar(x_error, y_error, e, linestyle='None', fmt='o', color=colors[i], capsize=5, alpha
plt.legend(loc='upper left')
plt.ylabel("Regret at time step t")
plt.xlabel("Time step t")
plt.title("Comparison of Regret at time t, (algorithm run only for one simulation)")
plt.rcParams["figure.figsize"] = (30,6)
plt.savefig('results/plot.png')
plt.show()
```