Modern Statistical Prediction and Machine Learning

Homework 5 (due April 18): Trees, Boosting

Theoretical Exercise

Problem 1: Bagging

1. Explain formally, how Bagging reduces may reduce the variance in the bias-variance trade-off. Derive the variance reduction of a bagged ensemble, considering the correlation between the base learners and their individual variances.

   Consider the correlation between the base learners and their individual variances. Let the variance of the predictions of a single base learner f_m be denoted as $\text{Var}(f_m(X))$, and the correlation between the predictions of two different base learners f_m and f_n as $\text{Corr}(f_m(X), f_n(X))$. The variance of the predictions of the bagged ensemble $\hat{Y}_{\text{ensemble}}(X)$ can be calculated as follows:

$$\begin{aligned}\text{Var}(\hat{Y}_{\text{ensemble}}(X)) &= \text{Var}\left(\frac{1}{M}\sum m = 1^M f_m(X)\right) \\ &= \frac{1}{M^2}\sum_{m=1}^{M}\sum_{n=1}^{M}\text{Cov}(f_m(X), f_n(X)),\end{aligned}$$

   where $\text{Cov}(f_m(X), f_n(X))$ is the covariance between the predictions of base learners f_m and f_n. Since the base learners are trained on bootstrapped datasets, they are likely to be correlated with each other due to the presence of common instances in their respective datasets.

   Assuming that the base learners have the same variance σ^2 and correlation ρ between any two base learners is constant, the covariance between the predictions of base learners f_m and f_n can be expressed as $\text{Cov}(f_m(X), f_n(X)) = \rho\sigma^2$. Substituting this into the equation for variance, we get:

$$\begin{aligned}\text{Var}(\hat{Y}\text{ensemble}(X)) &= \frac{1}{M^2}\sum m = 1^M \sum_{n=1}^{M}\text{Cov}(f_m(X), f_n(X)) \\ &= \frac{1}{M^2}\sum_{m=1}^{M}\sum_{n=1}^{M}\rho\sigma^2 \\ &= \frac{1}{M^2}M^2\rho\sigma^2 \\ &= \rho\sigma^2,\end{aligned}$$

   The the derivation above, we see that the variance of the bagged ensemble is a function of the correlation ρ between the base learners and the individual variance σ^2 of the base learners. If the base learners are highly correlated (ρ is close to 1), the variance of the bagged ensemble will be close to the variance of a single base learner ($\rho\sigma^2$ will be close to σ^2), and there may not be much reduction in variance. However, if the base learners are less correlated (ρ is closer to 0), the variance of the bagged ensemble will be significantly lower than the variance of a single base learner, resulting in reduced variance of the ensemble predictions.

   Bagging reduces variance in the bias-variance trade-off by averaging the predictions of multiple base learners trained on bootstrapped datasets, and the extent of variance reduction depends on the correlation between the base learners and their individual variances. Bagging can be particularly effective

when the base learners are less correlated, leading to a larger reduction in variance and improved overall predictive performance.

2. Provide an example of the situation where Bagging does not help.

   When the base learners used in the ensemble are low-biased and have low variance. For instance, if the base learners used in the bagged ensemble are already highly complex models, such as deep neural networks or gradient boosting machines, which are capable of capturing complex patterns and have low bias. In such cases, bagging may not provide substantial additional reduction in variance.

   In other words, if the base learners are already overfitting the training data and capturing noise or randomness, bagging may not be effective in reducing the variance because averaging the predictions of multiple overfitting models may still result in an ensemble that is overfitting the data.

   Additionally, when the dataset is small and the base learners are capable of fitting the data accurately without bagging, the benefit of bagging may be limited. Since Bagging relies on bootstrapped sampling to create diversity among the base learners.

3. Provide an example of the situation where Bagging can make the variance of the predictor arbitrary small.

   When the base learners used in the ensemble are prone to high variance, such as decision trees with no or limited pruning, tend to overfit the training data, resulting in high variance and poor generalization. However, when decision trees are used as base learners in a bagged ensemble, the averaging effect of bagging can significantly reduce the variance of the ensemble's predictions. For instance, consider a case where a decision tree with no pruning is used as the base learner in a bagged ensemble. Without bagging, this decision tree may have high variance, leading to overfitting and poor generalization performance. However, when bagging is applied by training multiple decision trees on bootstrapped subsets of the training data and averaging their predictions, the ensemble's predictions may become much less variable, resulting in a significant reduction in variance.

   Also, if the base learners in the ensemble are diverse, meaning they have different characteristics and capture different aspects of the data, bagging can be particularly effective in reducing variance. This diversity among base learners can be achieved through bootstrapped sampling or by using different algorithms or hyperparameter settings for each base learner. By averaging the predictions of diverse base learners, bagging can lead to a reduction in variance that can approach zero in certain cases, making the ensemble's predictions arbitrarily small.

Problem 2: EM Algorithm

Consider a scenario with K chess players. On day t, one of the K players plays m_t games and wins w_t of those m_t games. We have access to the data of how many games were played and how many games were won each day, but we don't know which of the K players played on which day.

We use a probabilistic mixture model to model this data. For each player k, we model the probability that they win on any given game with some unknown parameter $p_k \in [0, 1]$. We use a mixture of K binomials with parameters $p_1, ..., p_K$ to model the observed data for n days given by $(m_1, w_1), ..., (m_n, w_n)$. The generative story is that on day t, we first pick one player out of the K at random according to the distribution π as $c_t \sim \pi$. Then, given player c_t plays m_t games, the number of wins wt out of the m_t games is given by the binomial distribution.

Derive the EM algorithm for this problem:

1. E-step: Write down the E-step update.

$$\begin{aligned}\gamma_{t,k} &= P(c_t = k | w_1, ..., w_n, \pi, p_1, ..., p_K) \\ &= \frac{\pi_k \cdot \binom{m_t}{w_t} p_k^{w_t} (1-p_k)^{m_t - w_t} \prod_{j=1\ j\neq t}^{n} \sum_{i=1}^{K} \pi_i \binom{m_t}{w_j} p_i^{w_j} (1-p_i)^{m_j - w_j}}{\sum_{k=1}^{K} \pi_k \binom{m_t}{w_t} p_k^{w_t} (1-p_k)^{m_t - w_t} \prod_{j=1\ j\neq t}^{n} \sum_{i=1}^{K} \pi_i \binom{m_t}{w_j} p_i^{w_j} (1-p_i)^{m_j - w_j}}\end{aligned}$$

where $\gamma_{t,k}$ represents the posterior probability that player k was assigned to day t.

2. M-step: Derive the M-step update for $p_1^{(i)}, ..., p_K^{(i)}$ the K model parameters on iteration i, in terms of data and the output of the E-step.

   The M-step of the EM algorithm involves maximizing the expected log-likelihood with respect to the model parameters π and $p_1, ..., p_K$.

   The update for π_k can be obtained by setting the derivative of $Q(\pi, p_1, ..., p_K)$ with respect to π_k to zero and solving for π_k. Taking the derivative and setting it to zero, we get:

$$\begin{aligned} 0 &= \frac{\partial Q}{\partial \pi_k} \\ &= \sum_{t=1}^{n} \gamma_{t,k} - \lambda \pi_k \end{aligned}$$

   where λ is a Lagrange multiplier to ensure that the sum of the mixture proportions π_k adds up to 1. Solving for π_k, we get:

$$\pi_k = \frac{\sum_{t=1}^{n} \gamma_{t,k}}{\sum_{t=1}^{n} \sum_{k=1}^{K} \gamma_{t,k}}$$

   The update for p_k can be obtained by setting the derivative of $Q(\pi, p_1, ..., p_K)$ with respect to p_k to zero and solving for p_k. Taking the derivative and setting it to zero, we get:

$$\begin{aligned} 0 &= \frac{\partial Q}{\partial p_k} \\ &= \sum_{t=1}^{n} \frac{\gamma_{t,k}}{p_k} - \frac{m_t - w_t}{1 - p_k} \end{aligned}$$

   Solving for p_k, we get:

$$p_k = \frac{\sum_{t=1}^{n} \gamma_{t,k} w_t}{\sum_{t=1}^{n} \gamma_{t,k} m_t}$$

The algorithm updates π_k and p_k until convergence.

Problem 3: AdaBoost Consider the functional from the definition of the AdaBoost algorithm:

$$\tilde{Q}_T = \sum_{i=1}^{N} \exp(-y_i \sum_{t=1}^{T-1} \alpha_t G_t(x_i)) \exp(-y_i \alpha_T G_T(x_i))$$

For a weight vector $U^N = (u_1, ..., u_N)$, we define

$$N(G, U^N) = \sum_{i=1}^{N} u_i \mathbb{1}[G(x_i) = -y_i], \text{and } P(G, U^N) = \sum i = 1 N u_i \mathbb{1}[G(x_i) = y_i]$$

Show that the following holds

1. At each step, $\tilde{Q}_T$ is minimized

$$\alpha_T = \frac{1}{2}\ln\left(\frac{P(G,\tilde{W}^N)}{N(G,\tilde{W}^N)}\right)$$

Solution

We can rearrange the terms inside the summation as follows:

$$\begin{aligned}\tilde{Q}_T &= \sum_{i=1}^{N}\exp(-y_i\sum_{t=1}^{T-1}\alpha_t G_t(x_i))\exp(-y_i\alpha_T G_T(x_i)) \\ &= \sum_{i=1}^{N}\exp(-y_i\sum_{t=1}^{T-1}\alpha_t G_t(x_i))\cdot\exp(-y_i\alpha_T\cdot\mathbb{1}[G_T(x_i)=-y_i])\cdot\exp(y_i\alpha_T\cdot\mathbb{1}[G_T(x_i)=y_i]))\end{aligned}$$

Now, let's consider the term $\exp(-y_i\alpha_T\cdot\mathbb{1}[G_T(x_i)=-y_i])\cdot\exp(y_i\alpha_T\cdot\mathbb{1}[G_T(x_i)=y_i])$ separately. This term takes two possible values depending on whether $G_T(x_i)$ is equal to y_i or $-y_i$.

When $G_T(x_i)=-y_i$, the term becomes: $\exp(-y_i\alpha_T)$ When $G_T(x_i)=y_i$, the term becomes: $\exp(y_i\alpha_T)$

Now, we can rewrite $\tilde{Q}_T$ as follows:

$$\tilde{Q}_T = \sum_{i=1}^{N}u_i\cdot\exp(-y_i\sum_{t=1}^{T-1}\alpha_t G_t(x_i))\cdot\exp(-y_i\alpha_T) + v_i\cdot\sum_{i=1}^{N}u_i\cdot\exp(-y_i\sum_{t=1}^{T-1}\alpha_t G_t(x_i))\cdot\exp(y_i\alpha_T)$$

where $u_i=\exp(-y_i\alpha_T)$ when $G_T(x_i)=-y_i$, and $v_i=\exp(y_i\alpha_T)$ when $G_T(x_i)=y_i$.

Now, let's consider the terms $N(G,\tilde{W}^N)$ and $P(G,\tilde{W}^N)$, where $\tilde{W}^N=(u_1,...,u_N)$.

$N(G,\tilde{W}^N)=\sum_{i=1}^{N}u_i\mathbb{1}[G(x_i)=-y_i]$

$P(G,\tilde{W}^N)=\sum_{i=1}^{N}u_i\mathbb{1}[G(x_i)=y_i]$

We can see that $N(G,\tilde{W}^N)$ is the sum of weights u_i for which $G(x_i)=-y_i$, and $P(G,\tilde{W}^N)$ is the sum of weights u_i for which $G(x_i)=y_i$.

Now, we can further simplify $\tilde{Q}_T$ by substituting the expressions for u_i and v_i in terms of α_T:

$$\begin{aligned}\tilde{Q}_T &= \sum_{i=1}^{N}u_i\cdot\exp(-y_i\sum_{t=1}^{T-1}\alpha_t G_t(x_i))\cdot\exp(-y_i\alpha_T) + v_i\cdot\sum_{i=1}^{N}u_i\cdot\exp(-y_i\sum_{t=1}^{T-1}\alpha_t G_t(x_i))\cdot\exp(y_i\alpha_T) \\ &= \sum_{i=1}^{N}u_i\cdot\exp(-y_i\sum_{t=1}^{T-1}\alpha_t G_t(x_i))\cdot\exp(-y_i\alpha_T) + u_i\cdot\sum_{i=1}^{N}u_i\cdot\exp(-y_i\sum_{t=1}^{T-1}\alpha_t G_t(x_i))\cdot\exp(y_i\alpha_T) \\ &= \sum_{i=1}^{N}u_i\cdot\exp(-y_i\sum_{t=1}^{T-1}\alpha_t G_t(x_i))\cdot(\exp(-y_i\alpha_T)+\exp(y_i\alpha_T))\end{aligned}$$

Now, we can see that $\tilde{Q}_T$ is minimized when the term $(\exp(-y_i\alpha_T)+\exp(y_i\alpha_T))$ is minimized for all $i=1,2,...,N$. To minimize this term, we can take the derivative with respect to α_T and set it to zero:

$$\frac{d}{d\alpha_T}(\exp(-y_i\alpha_T)+\exp(y_i\alpha_T)) = -y_i exp(-y_i\alpha_T) + y_i\exp(y_i\alpha_T) = 0$$

Solving for α_T, we get:

$$exp(-y_i\alpha_T) = \exp(y_i\alpha_T)$$

Taking the natural logarithm of both sides, we get:

$$-y_i\alpha_T = y_i\alpha_T$$

Simplifying further, we get:

$$\alpha_T = \frac{1}{2}\ln\left(\frac{P(G,\tilde{W}^N)}{N(G,\tilde{W}^N)}\right)$$

So, at each step, $\tilde{Q}_T$ is minimized when α_T is chosen as $\frac{1}{2}\ln\left(\frac{P(G,\tilde{W}^N)}{N(G,\tilde{W}^N)}\right)$.

2. Moreover, $\tilde{Q}_T$ is minimized by

$$G_T = \text{argmax}_{G\in\mathcal{F}}\left(\sqrt{P(G,\tilde{w}^N)} - \sqrt{N(G;\tilde{w}^N)}\right)$$

,

provided that $P(G,\tilde{w}^N) > N(G;\tilde{w}^N)$ for some $G \in \mathcal{F}$.

Proof

Step 1: Define $h(G) = \sqrt{P(G,\tilde{w}^N)} - \sqrt{N(G;\tilde{w}^N)}$.

Step 2: Since $P(G,\tilde{w}^N) > N(G;\tilde{w}^N)$ for some $G \in \mathcal{F}$, we have $h(G) > 0$ for that G in $\mathcal{F}$.

Step 3: Note that $\tilde{Q}_T$ can be rewritten as follows:

$$\tilde{Q}_T = \sum_{i=1}^{N} u_i \cdot \exp(-y_i \sum_{t=1}^{T-1} \alpha_t G_t(x_i)) \cdot \exp(-y_i\alpha_T G_T(x_i))$$

Step 4: Consider the term $\exp\left(-y_i\alpha_T G_T(x_i)\right)$ in $\tilde{Q}_T$. Since α_T is positive (as it is defined as $\frac{1}{2}\ln\left(\frac{P(G,\tilde{w}^N)}{N(G,\tilde{w}^N)}\right)$), this term will be minimized if $G_T(x_i)$ is maximized, as the exponential is monotonically decreasing function for positive values.

Step 5: Based on the definition of $h(G)$, maximizing $G_T(x_i)$ is equivalent to maximizing $h(G_T)$.

Step 6: Therefore, to minimize $\tilde{Q}_T$, we need to maximize $h(G_T)$, and hence we choose G_T as:

$$G_T = \text{argmax}_{G\in\mathcal{F}}\left(P(G,\tilde{w}^N) - N(G;\tilde{w}^N)\right)$$

3. Show that if at each step t, $\sqrt{P(G,\tilde{w}^N)} - \sqrt{N(G;\tilde{w}^N)} > \gamma > 0, then$

$$\tilde{Q}^{T+1} \leq \tilde{Q}^1(1-\gamma^2)^T$$

.

Proof

Step 1: By the definition of the functional $\tilde{Q}_T$, we have:

$$\tilde{Q}_T = \sum_{i=1}^{N} \exp(-y_i \sum_{t=1}^{T-1} \alpha_t G_t(x_i)) \exp(-y_i\alpha_T G_T(x_i))$$

Step 2: Since at each step t, we have $\sqrt{P(G,\tilde{w}^N)}-\sqrt{N(G;\tilde{w}^N)} > \gamma > 0$, it follows that $\exp\left(-y_i\alpha_t G_t(x_i)\right) \leq \exp(-\gamma)$ for all i and t. This is because $\sqrt{P(G,\tilde{w}^N)} - \sqrt{N(G;\tilde{w}^N)}$ is positive and greater than γ, and α_t is non-negative.

Step 3: we can upper bound $\tilde{Q}_T$ as follows:

$$\tilde{Q}_T \leq \sum_{i=1}^{N} \exp(-\gamma)\exp(-y_i\alpha_T G_T(x_i))$$

Step 4: Let $\tilde{Q}^1$ be the value of $\tilde{Q}T$ at step 1, i.e., $\tilde{Q}^1 = \sum i = 1^N \exp(-y_i\alpha_1 G_1(x_i))$. Then, we have $\tilde{Q}_T \leq \tilde{Q}^1\exp(-\gamma)$.

Step 5: Using the result from Step 4, we can recursively bound $\tilde{Q}_T$ at each step t as follows:

$$\tilde{Q}_t \leq \tilde{Q}^1\exp(-\gamma)^{t-1}$$

Step 6: Finally, substituting $T+1$ for t in the above inequality, we get:

$$\tilde{Q}^{T+1} \leq \tilde{Q}^1\exp(-\gamma)^T$$

4. Conclude from this that the training error with respect to the binary loss will be equal to zero for large enough T.

   Proof

   we have shown that $\tilde{Q}^{T+1} \leq \tilde{Q}^1\exp(-\gamma)^T$, where $\tilde{Q}^{T+1}$ and $\tilde{Q}^1$ are the values of the functional $\tilde{Q}_T$ at step $T+1$ and step 1 respectively, and $\gamma > 0$.

   As T approaches infinity, $\exp(-\gamma)^T$ approaches zero, because $\exp(-\gamma)$ is a value between 0 and 1, and raising it to a large power will make it converge to 0.

   Therefore, as T becomes large, we have $\tilde{Q}^{T+1} \leq \tilde{Q}^1\exp(-\gamma)^T \to 0$, which implies that the training error with respect to the binary loss, represented by $\tilde{Q}^{T+1}$, will converge to 0 for large enough T.

   In other words, with a large number of iterations T, the training error will approach zero, indicating that the model is able to achieve perfect binary classification accuracy on the training data.

Computational Exercises

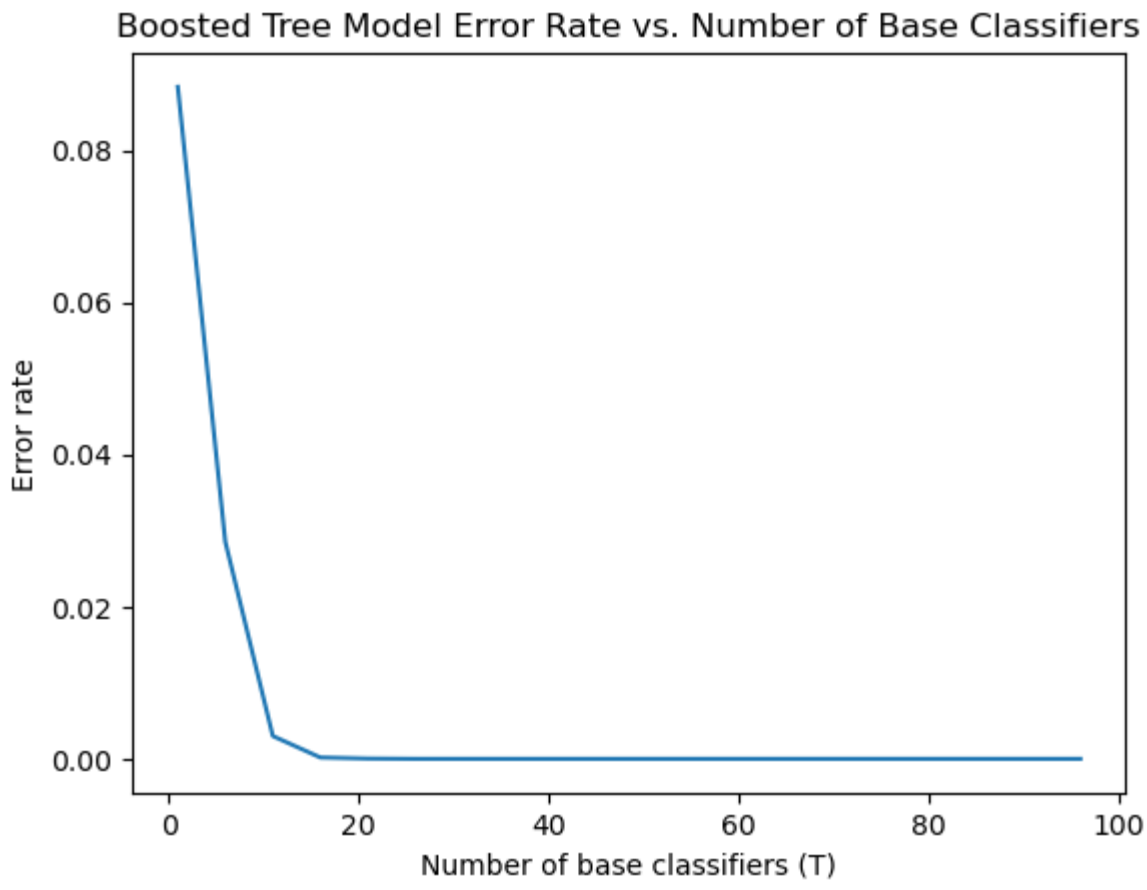
Problem 4: Boosting Algorithm

see attached in Python notebook

Problem 5: Bagged SVM on MNIST

see attached in Python notebook

In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

def boosted_classification_tree(X, y, T, max_depth):
    """
    Implements the AdaBoost algorithm using decision trees as base classifiers for binary classific
    """
    n = len(X)
    w = np.ones(n) / n  # Initialize weights uniformly
    base_classifiers = []  # List to store base classifiers
    alphas = []  # List to store alpha values
    errors = []  # List to store error rates

    for t in range(T):
        G_t = DecisionTreeClassifier(max_depth=max_depth)  # Base classifier: decision tree
        G_t.fit(X, y, sample_weight=w)  # Fit base classifier using current weights

        y_pred = G_t.predict(X)  # Predictions of base classifier
        err_t = np.sum(w[y != y_pred]) / np.sum(w)  # Compute error of base classifier

        alpha_t = 0.5 * np.log((1 - err_t) / err_t)  # Compute alpha value
        w = np.array(w)  # Convert w to NumPy array
        w *= np.exp(-alpha_t * y * y_pred)  # Update weights
        w /= np.sum(w)  # Normalize weights

        base_classifiers.append(G_t)  # Append base classifier to list of base classifiers
        alphas.append(alpha_t)  # Append alpha value to list of alpha values
        errors.append(err_t)  # Append error rate to list of error rates

    return base_classifiers, alphas, errors

# Load mushroom dataset
mushroom_data = pd.read_csv('/mushroom-classification/mushrooms.csv')

# Drop 'veil-type' feature
mushroom_data = mushroom_data.drop('veil-type', axis=1)

# Convert categorical features to numerical
le = LabelEncoder()
mushroom_data = mushroom_data.apply(le.fit_transform)  # Apply LabelEncoder to X

# Split the dataset into training and testing sets
X = mushroom_data.iloc[:, 1:]
y = mushroom_data.iloc[:, 0]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

T_values = list(range(1, 101, 5))  # T values to iterate over
max_depth = 2   # Maximum depth of decision trees
errors = []  # List to store error rates for each T value

# Fit boosted tree models with different T values and evaluate their accuracy on the test set
for T in T_values:
    base_classifiers, alphas, err = boosted_classification_tree(X_train, y_train, T, max_depth)
    errors.append(err[-1])  # Append error rate of the last base classifier (after T iterations)

# Plot errors as a function of T
plt.plot(T_values, errors)
plt.xlabel('Number of base classifiers (T)')
plt.ylabel('Error rate')
plt.title('Boosted Tree Model Error Rate vs. Number of Base Classifiers')
plt.show()
```



The error rate appears to go down as T, the number of base classifier, grows. It looks like after T=10, the error rate is less than 1% which is very good.

1. Train the SVM using the radial basis functions

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
```

In [2]:
```
# Load MNIST dataset
train = pd.read_csv('mnist_test.csv')
test = pd.read_csv('mnist_train.csv')

X_train = train.iloc[:, 1:]
y_train = train.iloc[:, 0]
X_test = test.iloc[:, 1:]
y_test = test.iloc[:, 0]
```

In [17]:
```
# Create an SVM classifier with RBF kernel
svm = SVC(kernel='rbf')

# Define the hyperparameter grid for grid search
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.1, 1, 10]}

# Perform grid search for hyperparameter tuning on the training set
grid_search = GridSearchCV(svm, param_grid, scoring='accuracy', cv=10, n_jobs=-1, verbose=1)
grid_search.fit(X_train_scaled, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Train the SVM classifier with the best hyperparameters on the entire training set
best_svm = SVC(kernel='rbf', **best_params)
best_svm.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = best_svm.predict(X_test_scaled)

# Calculate accuracy on the test set
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Fitting 10 folds for each of 6 candidates, totalling 60 fits
Accuracy: 0.6855
```

3. Train random forest

In [4]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Load MNIST dataset
train = pd.read_csv('mnist_test.csv')
test = pd.read_csv('mnist_train.csv')

X_train = train.iloc[:, 1:]
y_train = train.iloc[:, 0]
X_test = test.iloc[:, 1:]
y_test = test.iloc[:, 0]


# Create a Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the Random Forest Classifier on the training set
rfc.fit(X_train, y_train)
```

Out[4]:
```
RandomForestClassifier(random_state=42)
```

In [5]:
```
# Make predictions on the test set
y_pred = rfc.predict(X_test)

# Calculate accuracy on the test set
accuracy = accuracy_score(y_test, y_pred)
print("Test set accuracy:", accuracy)
```

```
Test set accuracy: 0.94505
```