CSCI567: Machine Learning	USC, Fall 2022
Homework 4	
Instructor: Vatsal Sharan	*Due: November 17 by 2:00 pm PST*

A reminder on collaboration policy and academic integrity: Our goal is to maintain an optimal learning environment. You can discuss the homework problems at a high level with other groups, but you should not look at any other group's solutions. Trying to find solutions online or from any other sources for any homework or project is prohibited, will result in zero grade and will be reported. To prevent any future plagiarism, uploading any material from the course (your solutions, quizzes etc.) on the internet is prohibited, and any violations will also be reported. Please be considerate, and help us help everyone get the best out of this course.

Please remember the Student Conduct Code (Section 11.00 of the USC Student Guidebook). General principles of academic honesty include the concept of respect for the intellectual property of others, the expectation that individual work will be submitted unless otherwise allowed by an instructor, and the obligations both to protect one's own academic work from misuse by others as well as to avoid using another's work as one's own. All students are expected to understand and abide by these principles. Students will be referred to the Office of Student Judicial Affairs and Community Standards for further review, should there be any suspicion of academic dishonesty.

Notes on notation:

- Unless stated otherwise, scalars are denoted by small letter in normal font, vectors are denoted by small letters in bold font and matrices are denoted by capital letters in bold font.
- $\|.\|$ means L2-norm unless specified otherwise, *i.e.*, $\|.\| = \|.\|_2$.

Instructions

We recommend that you use LaTeX to write up your homework solution. However, you can also scan handwritten notes. The homework will need to be submitted on Gradescope.

Theory-based Questions

Problem 1: Decision Trees (12pts)

Consider a binary dataset with 400 examples, where half of them belong to class A and the rest belong to class B. Next, consider two decision stumps (i.e. trees with depth 1) $\mathcal{T}_1$ and $\mathcal{T}_2$, each with two children. For $\mathcal{T}_1$, the left child has 150 examples in class A and 50 examples in class B. For $\mathcal{T}_2$, the left child has 0 examples in class A and 100 examples in class B. (You can infer the number of examples in the right child using the total number of examples.)

1.1 (6 pts) In class, we discussed entropy and Gini impurity as measures of uncertainty at a leaf. Another possible metric is the classification error at the leaf, assuming that the prediction at the leaf is the majority class among all examples that belong to that leaf. For each leaf of $\mathcal{T}_1$ and $\mathcal{T}_2$, compute the entropy (base e), Gini impurity and classification error. You can either exactly express the final numbers in terms of fractions and logarithms, or round them to two decimal places.

Solution:
Knowing that class A has 200 samples and class B has 200 samples, derive that the right child for $\mathcal{T}_1$ has 50 examples in class A and 150 examples in class B, and the right child for $\mathcal{T}_2$ has 200 examples in class A and 100 examples in class B.
Now, we can compute the entropy, Gini impurity and classification error for each leaf.
For $\mathcal{T}_1$,
left child:

$$\text{Entropy} = -\frac{150}{200}\ln(\frac{150}{200}) - \frac{50}{200}\ln(\frac{50}{200}) = \ln(4) - \frac{3}{4}\ln(3) = 0.56$$
$$\text{Gini impurity} = \frac{150}{200}\frac{50}{200} + \frac{50}{200}\frac{150}{200} = \frac{3}{8}$$
$$\text{Classification error} = \frac{50}{200} = \frac{1}{4}$$

right child:

$$\text{Entropy} = -\frac{50}{200}\ln(\frac{50}{200}) - \frac{150}{200}\ln(\frac{150}{200}) = \ln(4) - \frac{3}{4}\ln(3) = 0.56$$
$$\text{Gini impurity} = \frac{50}{200}\frac{150}{200} + \frac{150}{200}\frac{50}{200} = \frac{3}{8}$$
$$\text{Classification error} = \frac{50}{200} = \frac{1}{4}$$

For $\mathcal{T}_2$,
left child:

$$\text{Entropy} = -\frac{0}{100}\ln(\frac{0}{100}) - \frac{100}{100}\ln(\frac{100}{100}) = 0$$
$$\text{Gini impurity} = \frac{0}{100}\frac{100}{100} + \frac{100}{100}\frac{0}{100} = 0$$
$$\text{Classification error} = \frac{0}{100} = 0$$

right child:

$$\text{Entropy} = -\frac{200}{300}\ln(\frac{200}{300}) - \frac{100}{300}\ln(\frac{100}{300}) = \ln(3) - \frac{2}{3}\ln(2) = 0.64$$
$$\text{Gini impurity} = \frac{200}{300}\frac{100}{300} + \frac{100}{300}\frac{200}{300} = \frac{4}{9}$$
$$\text{Classification error} = \frac{100}{300} = \frac{1}{3}$$

1.2 (6 pts) Compare the quality of $\mathcal{T}_1$ and $\mathcal{T}_2$ (that is, the two different splits of the root) based on conditional entropy (base e), weighted Gini impurity and total classification error. Intuitively, which of $\mathcal{T}_1$ or $\mathcal{T}_2$ appears to be a better split to you (there may not necessarily be one correct answer to this)? Based on your conditional entropy, Gini impurity and classification error calculations, which of the metrics appear to be more suitable choices to decide which variable to split on?

Solution:
For $\mathcal{T}_1$,

$$\text{Conditional entropy} = \frac{200}{400}(\ln(4) - \frac{3}{4}\ln(3)) + \frac{200}{400}(\ln(4) - \frac{3}{4}\ln(3)) = \ln(4) - \frac{3}{4}\ln(3) = 0.56$$
$$\text{Weighted Gini impurity} = \frac{200}{400}\frac{3}{8} + \frac{200}{400}\frac{3}{8} = \frac{3}{8}$$
$$\text{Total classification error} = \frac{200}{400}\frac{1}{4} + \frac{200}{400}\frac{1}{4} = \frac{1}{4}$$

For $\mathcal{T}_2$,

$$\text{Conditional entropy} = \frac{100}{400}0 + \frac{300}{400}(\ln(3) - \frac{2}{3}\ln(2)) = \frac{3}{4}\ln(3) - \frac{1}{2}\ln(2) = 0.48$$
$$\text{Weighted Gini impurity} = \frac{100}{400}0 + \frac{300}{400}\frac{4}{9} = \frac{1}{3}$$
$$\text{Total classification error} = \frac{100}{400}0 + \frac{300}{400}\frac{1}{3} = \frac{1}{4}$$

Intuitively, $\mathcal{T}_2$ appears to be a better split because it has less uncertainty for leaves. Based on calculations, conditional entropy and weighted Gini impurity are more suitable to decide split because they measure less uncertainty for $\mathcal{T}_2$, while total classification error gives equal measures for $\mathcal{T}_1$ and $\mathcal{T}_2$.

Problem 2: Gaussian Mixture Model and EM

In class, we applied EM to learn Gaussian Mixture Models (GMMs) and showed the M-Step without a proof. Now, it is time that you prove it.

Consider a GMM with the following PDF of $\mathbf{x}_i$:

$$p(\mathbf{x}_i) = \sum_{j=1}^{k} \pi_j N(\mathbf{x}_i \mid \mu_j, \Sigma_j) = \sum_{j=1}^{k} \frac{\pi_j}{(\sqrt{2\pi})^d |\Sigma_j|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \mu_j)\right)$$

where k is the number of Gaussian components, d is dimension of a data point $\mathbf{x}_i$ and N is the usual Gaussian pdf ($|\Sigma|$ in the pdf denotes the determinant of matrix Σ). This GMM has k tuples of model parameters $\{(\mu_j, \Sigma_j, \pi_j)\}_{j=1}^k$, where the parameters represent the mean vector, covariance matrix, and component weight of the j-th Gaussian component. For simplicity, we further assume that all components are isotropic Gaussian, i.e., $\Sigma_j = \sigma_j^2 I$.

2.1 (10 pts) Find the MLE of *the expected complete log-likelihood*. Equivalently, find the optimal solution to the following optimization problem.

$$\begin{aligned} \underset{\pi_j, \mu_j, \Sigma_j}{\operatorname{argmax}} & \sum_i \sum_j \gamma_{ij} \ln \pi_j + \sum_i \sum_j \gamma_{ij} \ln N(\mathbf{x}_i \mid \mu_j, \Sigma_j) \\ \text{s.t. } & \pi_j \geq 0 \\ & \sum_{j=1}^{k} \pi_j = 1 \end{aligned}$$

where γ_{ij} is the posterior of latent variables computed from the E-Step.

You can use the following fact: Given $a_1, \ldots, a_k \in \mathbb{R}^+$, the solution to the following optimization problem over $q_1, \ldots, q_k$:

$$\begin{aligned} \underset{q_j}{\operatorname{argmax}} & \sum_{j=1}^{k} a_j \ln q_j, \\ \text{s.t. } & q_j \geq 0, \\ & \sum_{j=1}^{k} q_j = 1. \end{aligned}$$

is given by:

$$q_j^* = \frac{a_j}{\sum_{k'} a_{k'}}$$

Solution:
We can separate the left term and right term, then find their maximum respectively because they are not related.
For the left part:

$$\begin{aligned} \underset{\pi_j}{\operatorname{argmax}} & \sum_i \sum_j \gamma_{ij} \ln \pi_j \\ \text{s.t. } & \pi_j \geq 0 \\ & \sum_{j=1}^{k} \pi_j = 1 \end{aligned}$$

,which is similar to the given fact. Thus, we can derive that

$$\pi_j^* = \frac{\sum_i \gamma_{ij}}{\sum_i \sum_j \gamma_{ij}} = \frac{\sum_i \gamma_{ij}}{n}$$

,where n is the number of data points.

For the right part:

$$\underset{\mu_j,\Sigma_j}{\operatorname{argmax}} \sum_i \sum_j \gamma_{ij} \ln N(\mathbf{x}_i \mid \mu_j, \Sigma_j) = \sum_j \underset{\mu_j,\Sigma_j}{\operatorname{argmax}} \sum_i \gamma_{ij} \ln N(\mathbf{x}_i \mid \mu_j, \Sigma_j)$$

Then, for each j,

$$\begin{aligned}
\underset{\mu_j,\Sigma_j}{\operatorname{argmax}} \sum_i \gamma_{ij} \ln N(\mathbf{x}_i \mid \mu_j, \Sigma_j) &= \underset{\mu_j,\Sigma_j}{\operatorname{argmax}} \sum_i \gamma_{ij} \ln \left[\frac{1}{(\sqrt{2\pi})^d |\Sigma_j|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} (\mathbf{x}_i - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \mu_j) \right) \right] \\
&= \underset{\mu_j,\sigma_j}{\operatorname{argmax}} \sum_i \gamma_{ij} \ln \left[\frac{1}{(\sqrt{2\pi}\sigma_j)^d} \exp\left(-\frac{\|\mathbf{x}_i - \mu_j\|^2}{2\sigma_j^2} \right) \right] \\
&= \underset{\mu_j,\sigma_j}{\operatorname{argmax}} \sum_i \gamma_{ij} \left(-d \ln \sigma_j - \frac{\|\mathbf{x}_i - \mu_j\|^2}{2\sigma_j^2} \right)
\end{aligned}$$

First, we get the derivative with respect to μ_j and set it to zero

$$\frac{1}{\sigma_j^2} \sum_i \gamma_{ij} (\mathbf{x}_i - \mu_j) = 0$$

Thus, the optimal solution for μ_j is

$$\mu_j^* = \frac{\sum_i \gamma_{ij} \mathbf{x}_i}{\sum_i \gamma_{ij}}$$

Then, we get the derivative with respect to σ_j and set it to zero

$$\sum_i \gamma_{ij} (-\frac{d}{\sigma_j} + \frac{\|\mathbf{x}_i - \mu_j\|^2}{\sigma_j^3}) = 0$$

Finally, utilized μ_j^* derived above, find the optimal solution for σ_j is

$$(\sigma_j^*)^2 = \frac{\sum_i \gamma_{ij} \|\mathbf{x}_i - \mu_j^*\|^2}{d \sum_i \gamma_{ij}}$$

2.2 (Bonus) (5 pts) The posterior probability of z in GMM can be seen as a *soft* assignment to the clusters; in contrast, k-means assign each data point to one cluster at each iteration (*hard* assignment). Show that if we set $\{\sigma_j, \pi_j\}_{j=1}^k$ in a particular way in the GMM model, then the cluster assignments given by the GMM reduce in the limit to the k-means clusters assignment (where the cluster centers $\{\mu_j\}_{j=1}^k$ remain the same for both the models). To verify your answer, you should derive $p(z_i = j|\boldsymbol{x}_i)$ for your choice.

Solution:
Derive that

$$\begin{aligned}
p(z_i = j \mid \boldsymbol{x}_i) &= \frac{p(\boldsymbol{x}_i \mid z_i = j) p(z_i = j)}{\sum_{j=1}^k p(\boldsymbol{x}_i \mid z_i = j) p(z_i = j)} \\
&= \frac{\frac{\pi_j}{(\sqrt{2\pi}\sigma_j)^d} \exp\left(-\frac{\|\mathbf{x}_i - \mu_j\|^2}{2\sigma_j^2} \right)}{\sum_{j=1}^k \frac{\pi_j}{(\sqrt{2\pi}\sigma_j)^d} \exp\left(-\frac{\|\mathbf{x}_i - \mu_j\|^2}{2\sigma_j^2} \right)}
\end{aligned}$$

Then, set $\sigma_j = \sigma \to 0$ and $\pi_j = \frac{1}{k}$ for all j, we get

$$p(z_i = j \mid \boldsymbol{x}_i) = \lim_{\sigma \to 0} \frac{\exp\left(-\frac{\|\mathbf{x}_i - \mu_j\|^2}{2\sigma^2}\right)}{\sum_{j=1}^{k} \exp\left(-\frac{\|\mathbf{x}_i - \mu_j\|^2}{2\sigma^2}\right)} \to \begin{cases} 1 \text{ , if } j = \operatorname{argmin}_j \|\mathbf{x}_i - \mu_j\|^2 \\ 0 \text{ , others} \end{cases}$$

Now, the GMM cluster assignment reduces to k-means cluster assignment with the posterior is one-hot.

Programming-based Questions

As in previous homeworks, you need to have your coding environment setup for this part. We use python3 (version $\geq$ 3.7) in our programming-based questions. There are multiple ways you can install python3, for example:

- You can use conda to configure a python3 environment for all programming assignments.
- Alternatively, you can also use virtualenv to configure a python3 environment for all programming assignments

After you have a python3 environment, you will need to install the following python packages:

- numpy
- matplotlib (for plotting figures)

Note: You are **not allowed** to use other packages such as *tensorflow, pytorch, keras, scikit-learn, scipy*, etc. for 3.1-3.2. If you have other package requests, please ask first before using them. You are **allowed** to use any packages for 3.3-3.5.

Download the files for the programming part from `https://vatsalsharan.github.io/fall22/hw3.zip`.

Problem 3: Exploring Decision Trees and Random Forests (12pts)

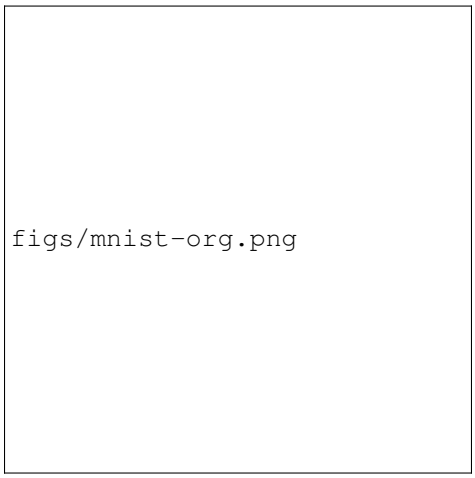
In this question, we will observe the effect of different hyperparameters in training decision trees and random forests, and also visualize what features the random forest model is using to make predictions. We will do this on a Colab notebook `HW4-Exploring-Random-Forests.ipynb`.

Instructions to run the notebook: Upload this file to your USC Google Drive. Then, add Google Colab to your Google App by New $\rightarrow$ More $\rightarrow$ Connect more apps $\rightarrow$ Type in Google Colab and install it, as in Fig. 1. After that, you can run the notebook with your browser.

Figure 1: Screenshots showing how to install Colab.

We use a variant of the MNIST dataset for this problem. As mentioned in HW 3, the MNIST dataset contains images of handwritten digits (0, 1, 2, ..., 9) and is generally used for the (10) digit classification problem. Here, we work with a binary classfication task of predicting whether the digit is less than 5 or not. Fig. 2 shows some samples images from the dataset with original and binary labels, respectively.

You should go through the code we provide and understand what's happening, but for the purpose of answering the questions you will mainly have to run the code and understand the results. All the models (decision trees, random forests, etc.) are imported from the `sklearn` library. You should feel free to explore the role of other hyperparameters



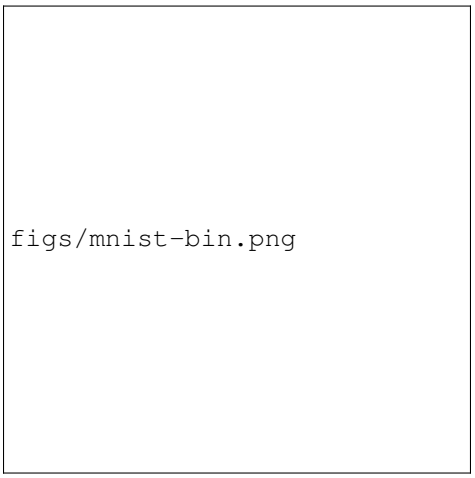


Figure 2: Some samples images from the MNIST dataset with original (left) and binary (right) labels, respectively.

and other methods (such as bagging, boosting), and go through the documentation to better understand these things, but for this question, we will focus on decision trees and random forests.

We will look at the effect of 5 parameters:

- `max_depth`: The maximum depth of the decision tree.
- `n_estimators`: The number of decision trees in the forest.
- `min_samples_leaf`: The minimum number of samples required to be a leaf node.
- `max_samples`: The number of samples to draw from the training set to train each decision tree in the forest.
- `max_features`: The number of features to consider when looking for the best split for any node.

To observe the effect of a parameter, we look at train and test accuracy for different values of that parameter while keeping the rest of the parameters fixed.

3.1 (2 pts) The first set of plots shows the train and test accuracy for different values of `max_depth` using a decision tree and a random forest, respectively. How do the accuracies and the generalization gap vary with `max_depth` in these cases? For a particular value of `max_depth`, how does the generalization gap for the decision tree compare with that of the random forest? Explain your observations.

Solution:
For both decision tree and random forest, the train and test accuracy first increase as the `max_depth` becomes bigger, then at a point, train accuracy keeps going up, while test accuracy stagnates at a level or even starts to go down. Overall, the generalization gap keeps growing as `max_depth` increases. However, the generalization gap of the random forest is smaller than that of the decision tree. For `max_depth` equals 10, the generalization gap of random forest is around 8, while the decision tree is around 14. This is because when depth is too small, underfitting happens leading to low train and test accuracy and small generalization gap; as depth grows, more splits could be made on nodes which improve the performance, while generalization gap is also growing; when depth is too large, overfitting happens which makes train accuracy keeps growing, while test accuracy might stagnate and even go down, and the generalization gap keeps growing. The reason why the generalization gap of random forest is smaller that that of decision tree is that random forest has better generalization ability than decision trees. When depth goes large, a decision tree will overfit the whole training data. However, random forest trains multiple trees with different structures based on bootstrap samplings of original training set, which reduces the overfitting.

3.2 (2 pts) The next plot shows the train and test accuracy for different values of `n_estimators` for a random forest. Comment on your observations regarding the accuracies and the generalization gap. What value(s) (range or approximate values are enough) would you prefer to use for this parameter? Give reasons why. Hint: Are there any drawbacks to using very high values of `n_estimators`?

Solution:
Both train and test accuracy first increase as the `n_estimators` goes up, then reach a maximum and remain almost constant when `n_estimators` is too large. However, the generalization gap remains almost constant when `n_estimators` varies. I prefer to use approximately 25 to 50 `n_estimators`. Because the train and test accuracy reach the maximum at that range and it prevents from using more trees that cost more computing and storage resources while the performance is not improved.

For the next three parts of this question, we will also look at the how the size of the training set influences the accuracy trends for a given parameter. In each case, for the first plot, the training set consists of 4000 samples whereas for the second plot, it contains 1000 samples.

3.3 (2 pts) The next set of plots shows the train and test accuracy for different values of `min_samples_leaf` for a random forest. Taking into account the behaviours for different training set sizes, explain your observations for very low and very high values of `min_samples_leaf`. What do you conclude from this trend? What could be the reasons for such a behaviour?

Solution:
For very low value of `min_samples_leaf`, both train and test accuracy are large and the generalization gap is also big. For very high value of `min_samples_leaf`, both train and test accuracy are small and the generalization gap is small. Notice that for 1000 training set size, when `min_samples_leaf` is very high, the train and test accuracy are almost the same. Conclude that an intermediate value should be chosen for `min_samples_leaf`, which could lead to both good accuracy and good generalization ability. The reason for this behavior might be when the `min_samples_leaf` is very low, more split could be made on data and only a few samples could make up a leaf node, which causes overfitting, therefore, the accuracy goes up and the generalization gap also goes up; when the `min_samples_leaf` is very high, the leaf node contains too many samples, which is not split very well and causes underfitting, so the accuracy and the generalization gap goes down. For training set with 1000 samples, when the `min_samples_leaf` is very high, samples within leaf node might be too many compared to samples within root node, which causes the model doesn't learn anything. Thus, the train and test accuracy are almost the same.

3.4 (2 pts) The next set of plots shows the train and test accuracy for different values of `max_samples` for a random forest. What do you observe for very low and very high values of `max_samples`? Would you prefer to use low, intermediate or high values for this parameter in both the cases? Give reasons why. Hint: How does the size of the training set influence the choice of this parameter?

Solution:
For 4000 training samples, the very low `max_samples` causes low train and test accuracy and small generalization gap, while the very high `max_samples` causes high train and test accuracy and big generalization gap. For 1000 training samples, the very low `max_samples` causes the train and test accuracy to be low and almost the same, while the very high `max_samples` causes high train accuracy and high test accuracy (but not the maximum) and big generalization gap. I prefer to use intermediate values for this parameter in both cases. Because it gives both good accuracy and good generalization ability. However, for smaller training set, this parameter should be relatively larger. Because the number of samples to train each tree might be too small if the parameter is small and it would not learn well.

3.5 (2 pts) The next set of plots shows the train and test accuracy for different values of `max_features` for a random forest. Comment on your observations regarding the accuracies and the generalization gap for the two training set sizes. What is the best range of values for this parameter in both the cases? Is it similar/different? Explain your observations.

Solution:
For both cases, the train and test accuracy first increase as the `max_features` increases, then reach a maximum and finally decrease (The decreasing of train accuracy for 4000 training samples is very small). The decreasing speed of accuracy for 4000 training samples is slower than that for 1000 training samples. The generalization gap is nearly going up continuously as `max_features` increases. This is because when `max_features` is very small, underfitting happens leading to low train and test accuracy and small generalization gap; as `max_features` increases, the trees fit the data better due to more select options of features leading to higher train and test accuracy and bigger generalization gap; when `max_features` is too large, overfitting happens and trees in the forest are less diverse, which leads to bigger generalization gap and lower test accuracy. Train accuracy is also decreasing, which might be caused by that the `max_samples` is set to be 0.5 and trees are trained only depending on a portion of training data. The reason why accuracy decreases faster for 1000 training samples might be smaller training set is easier to overfit and may only depend on a small number of relevant features. The best range of this parameter is similar for both cases, which is about from 25 to 100 that generates both good accuracy and good generalization ability.

3.6 (2 pts) In class, we discussed how ensembles are usually not as interpretable as a single decision tree. While this is true, there are still ways to explore which features are used the most by our ensemble. We will explore one such technique in this part.

We visualize the *feature importances* of a random forest model trained for the binary classification task on the MNIST dataset. Intuitively, features with higher importance are the pixels which are used more often in the decision trees in the forest and which lead to better splits, i.e. which contribute more in improving the performance of the model. For more details, you can see Section 18.6.1 of the PML book. The last plot shows the importances of different pixels/portions of the image for a trained random forest model to make its predictions. What portions of the image does the model seem to be focusing on? In other words, can you think of reasons why the pixels with higher importance are indeed important for the prediction task (classifying whether the digit is smaller than 5 or not)? As is usually true for such open-ended questions, there can be multiple correct answers here and we're looking more for your reasoning than a specific answer.

Solution:
The model seems to focus on the upper-left portion that nears the center of the image. After overlaying digit images with this feature importance images, find that all digits bigger than 5 contain that upper-left portion of pixels in their strokes, while in other digits, only 0 contains it in its stroke and 2, 4 contain it at the end point of their strokes. Thus, the reason why these pixels are important might be that it lies on the strokes of multiple digits, on which the classification depends, and its value could roughly separate the digits bigger than 5 from others, which could help split the nodes.

Problem 4: PCA for Learning Word Embeddings

This question is about *word embeddings*. We saw word embeddings in class in lecture 7. As we discussed then, a word embedding is simply a vector space representation of words which captures some of the semantic and syntactic structures in the language—for example, words similar in meaning have representations which are close to each other in the vector space. Word embeddings have taken natural language processing (NLP) by storm in the past decade or so, and have become the backbone for numerous NLP tasks such as question answering and machine translation. There are neural approaches to learning word embeddings, but in this question we will study a simple PCA-based scheme which does a surprisingly good job at learning word embeddings.

We have created a word co-occurrence matrix $\mathbf{M}$ of the 10000 most frequent words from a Wikipedia corpus with 1.5 billion words. The co-occurrences were obtained by using a sliding window of length 5 across the Wikipedia corpus. Entry M_{ij} of the matrix denotes the number of times words i and j occur in the corpus within the same sliding window. The file `co_occur.csv` contains the symmetric co-occurence matrix. `dictionary.txt` contains the dictionary for interpreting this matrix, the ith row of the dictionary is the word corresponding to the ith row or column of $\mathbf{M}$. The dictionary is sorted according to the word frequencies. Therefore the first word in the dictionary—"the" is the most common word in the corpus and the first row or column of $\mathbf{M}$ contains the co-occurrence counts of "the" with every other word in the dictionary.

We provide some starter code in the file `hw4-pca.py` with some useful functions (e.g. to read files, generate plots, etc.) which can be used directly, and instructions on how to complete the functions required for this problem.

4.1 (6 pts) First, read the co-occurrance matrix and the list of all words from the given files. Let the matrix $\mathbf{M}$ be the $n \times d$ ($n = d = 10000$) matrix of word co-occurrences. As we discussed in class, a suitable normalization or scaling is often very helpful to get PCA to work well. In light of the power law distribution of word occurrences, in this case we will work with the normalized matrix $\tilde{\mathbf{M}}$ such that each entry $\tilde{M}_{ij} = \log(1 + M_{ij})$. We regard the i-th row of $\mathbf{M}$ as the datapoint for the i-th word.

We will use PCA to find the first 100 principal components of the data. Let $\tilde{\mathbf{M}}_c$ be the centered version of $\tilde{\mathbf{M}}$. Use the `PCA` function from the `sklearn` library (refer `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`) to get $\mathbf{V}$, *i.e.* the set of first 100 principal components or eigenvectors of $\tilde{\mathbf{M}}_c$. Note that you can directly use the `fit` method with $\tilde{\mathbf{M}}_c$ as the input. Also get the eigenvalues of the covariance matrix (check the documentation of the function) and plot all the 100 eigenvalues. Do the eigenvalues seem to decay? What percent of the variance in the data is explained by the first 100 eigenvalues we calculated (note that there are 10,000 eigenvalues in total)?

Solution:

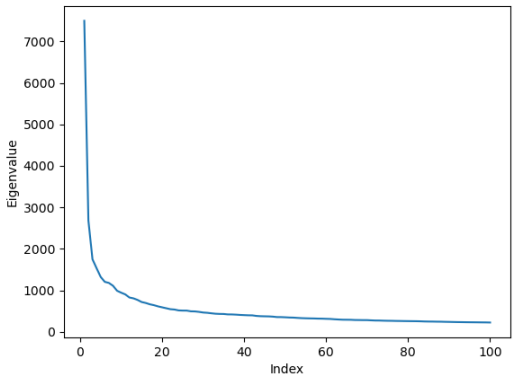


Figure 3: Plot of first 100 eigenvalues.

The plot is shown in Fig. 3. Observe that eigenvalues do decay in the plot. The percent of the variance in the data explained by the first 100 eigenvalues is 75.7%.

4.2 (6 pts) In this question, we find embeddings for all 10,000 words in the dictionary using the princpal components $\mathbf{V}$. Then, we will use word embeddings to find word(s) which are 'similar' to a given word.

To obtain the embeddings of the words, we will project the datapoint corresponding to each word (its co-occurences with every other word) onto the 100 dimensional space spanned by the first 100 principal components, similar to the general approach we laid down in class. Here are the two steps you should follow. Recall that we regard the i^{th} row of $\tilde{\mathbf{M}}_c$ as the datapoint for the i^{th} word. Given the 100 PCs ($\mathbf{V}$), we now project each datapoint (row of $\tilde{\mathbf{M}}_c$) onto these PCs. Denote this $n \times k$ (10000×100) matrix as $\mathbf{P}$ (you should write $\mathbf{P}$ as a matrix operation using $\tilde{\mathbf{M}}_c$ and $\mathbf{V}$). Next, to ensure that each PC gets equal importance, we normalize the vector of the projections of all the words onto the j^{th} PC (*i.e.* the j^{th} column of $\mathbf{P}$) to have unit norm, for all $j = \{1, ..., 100\}$. Denote this $n \times k$ (10000×100) matrix as $\mathbf{E}$. Finally, normalize the rows of $\mathbf{E}$ such that each row has unit ℓ_2 norm, to get a new matrix $\tilde{\mathbf{E}}$.

We regard the ith row of $\tilde{\mathbf{E}}$ as the embedding of the ith word. Next, we will define a similarity metric for the word embeddings. We will use the cosine-similarity as the similarity metric. As all the vectors have unit ℓ_2 norm, the cosine similarity between two words i and j with embeddings $\mathbf{w}_i$ and $\mathbf{w}_j$ is equal to the inner product $\langle \mathbf{w}_i, \mathbf{w}_j \rangle$. Now that we have a similarity metric defined, we can have some fun with these embeddings by querying for the closest word to any word we like! Try finding the closest words to some common words, such as "learning", "university", "california", and comment on your observations.

Solution:
Most similar word of 'learning': teaching
Most similar word of 'university': college
Most similar word of 'california': florida
Observe that by querying word embeddings, it will find words with similar meanings or appearing in similar contexts, which means the word embeddings will embed similar words close to each other in vector space.

4.3 (6 pts) We'll now interpret the principal components/eigenvectors (columns of $\mathbf{V}$). For any i, denote $\mathbf{v}_i$ as the eigenvector corresponding to the ith largest eigenvalue. Note that the entries of this vector correspond to the 10000 words in our dictionary, we'll call these our 10000 variables. By sorting the entries of $\mathbf{v}_i$ by absolute value, and observing what the top 10 variables and their (signed) entries are, we can infer what information the ith eigenvector roughly captures. Can you find 5 interesting eigenvectors, and point out what semantic or syntactic structures they capture? Can you do this for all 100 eigenvectors? Hint: What do you observe about PCs or eigenvectors with small eigenvalues?

Solution:
The words corresponding to the top 10 absolute entries in each eigenvectors are stored in `info_ev.txt` file. Below are 5 interesting eigenvectors found with easy-to-interpret meanings:

10-th eigenvector with 943.9410315110822 eigenvalue captures information about 'Military':
$[ii, squadron, commander, forces, command, aircraft, russian, german, army, soviet]$
27-th eigenvector with 495.0458167751042 eigenvalue captures information about 'Country':
$[dutch, canadian, american, swedish, scottish, italian, german, polish, australian, irish]$
30-th eigenvector with 466.1537359278541 eigenvalue captures information about 'Male Name':
$[mike, richard, tom, robert, george, michael, peter, david, william, john]$
47-th eigenvector with 366.73350487663083 eigenvalue captures information about 'Movie and TV Drama':
$[actor, film, directed, party, drama, tv, starring, actress, television, comedy]$
72-th eigenvector with 273.43875306183565 eigenvalue captures information about 'Large Degree':
$[significant, much, wide, huge, used, can, considerable, massive, greater, growing]$

Observe that PCs or eigenvectors with small eigenvalues are likely to capture the most common words in `dictionary.txt`, which might be caused by that they are easy to co-occur with every other word and have less variations.

4.4 (12 pts) In this question, we will explore a curious property of the word embeddings—that certain directions in the embedded space correspond to specific syntactic or semantic concepts. Let $\mathbf{w}_1$ be the word embedding for "woman" and $\mathbf{w}_2$ be the word embedding for "man". Let $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_2$, and $\hat{\mathbf{w}} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$.

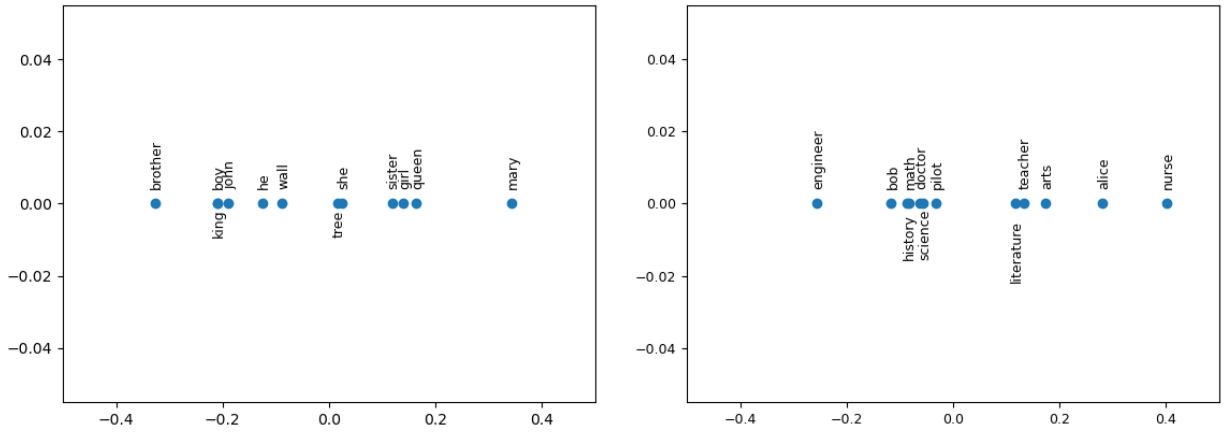
4.4.1 (6 pts) Project the embeddings of the following words onto $\hat{\mathbf{w}}$: *boy, girl, brother, sister, king, queen, he, she, john, mary, wall, tree*. Present a plot of projections of the embeddings of these words marked on a line. For example, if the projection of the embedding for "girl" onto $\hat{\mathbf{w}}$ is 0.1, then you should label 0.1 on the line with "girl". What do you observe?

Solution:
The plot is shown in Fig. 4a. Observe that words with female meanings, such as 'Mary', 'queen', 'girl' and 'sister', have positive values. Words with male meanings, such as 'brother', 'John', 'boy', 'king' and 'he', have negative values. Gender-neutral words, like 'tree', are near 0. The cause of 'she' being near zero might be 'she' is often to represent people's favorite objects. 'Wall' is towards the man side, which might be because 'wall' is often used to represent a man's hardness. The projections captures the gender meanings of words.

4.4.2 (6 pts) Present a similar plot of the projections of the embeddings of the following words onto $\hat{\mathbf{w}}$: *math, history, nurse, doctor, pilot, teacher, engineer, science, arts, literature, bob, alice.* What do you observe? Why do you think this is the case? Do you see a potential problem with this? Remember that word embeddings are extensively used across NLP. Suppose LinkedIn used such word embeddings to find suitable candidates for a job or to find candidates who best match a search term or job description. What might be the result of this?

If you want to learn more about this, you might find it interesting to read the original paper[1] which pointed out this issue in word embeddings.

Solution:
The plot is shown in Fig. 4b. Observe that words without a gender meaning are also divided by the projections. This might be caused by that certain words, such as 'engineer' and 'math', appear more frequently with male-meaning words, while others, such as 'nurse' and 'arts', appear more frequently with female-meaning words in our data. The potential problem is that the model is biased by capturing the biases in data. If Linkedin used this embedding, it might likely to find men for engineer and math-related jobs and find women for nurse and art-related jobs as candidates without considering other aspects of a person.



(a) Embedding projections on 'man' to 'woman'

(b) Embedding projections on 'man' to 'woman'

Figure 4: Plots of embedding projections on certain direction.

[1] https://proceedings.neurips.cc/paper/2016/file/a486cd07e4ac3d270571622f4f316ec5-Paper.pdf

4.5 (Bonus) (10 pts) In this question, we will explore the property that that directions in the embedded space correspond to semantic or syntactic concepts in more depth.

Because word embeddings capture semantic and syntactic concepts, they can be used to solve word analogy tasks. For example, consider an analogy question— *"man is to woman as king is to ___"*, where the goal is to fill in the blank space. This can be solved by finding the word whose embedding is closest to $\mathbf{w}_{\text{woman}} - \mathbf{w}_{\text{man}} + \mathbf{w}_{\text{king}}$ in cosine similarity. You can do this by a nearest neighbor search across the entire dictionary—excluding the three words *man, woman, king* which already appear in the analogy as they cannot be valid answers to the question. Here $\mathbf{w}_i$ represents the word embedding for the word i. Refer to Fig. 5 for why this makes sense in light of the fact that directions in the embedded space correspond to semantic/syntactic concepts.

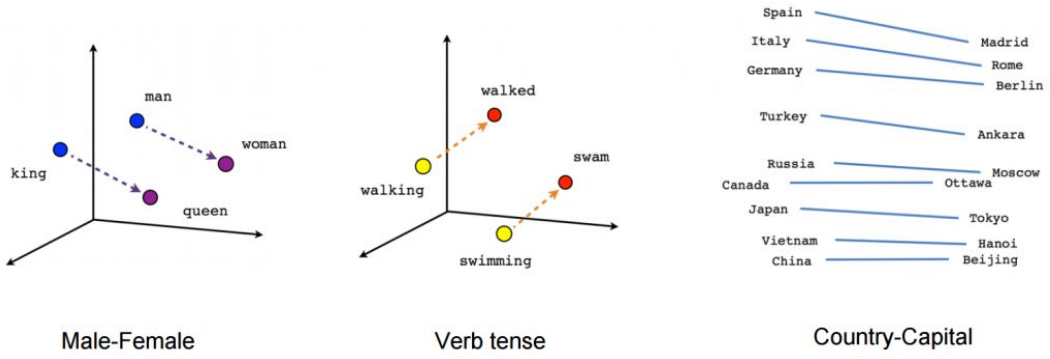


Figure 5: Figure denoting how word embeddings might encode gender, tense or country-capital relationships. Image source.

We have provided a dataset, `analogy_task.txt`, which tests the ability of word embeddings to answer analogy questions, such as those represented in Fig. 5. Using the cosine similarity metric, find and report the accuracy of the word embeddings you have constructed on the word analogy task. Look at the incorrect/correct answers of the approach and comment on the results. For example, what types of analogy questions seem to be harder to answer correctly for this approach?

Solution:
The accuracy of word analogy task is 55.3%. By looking at the correct predictions and incorrect predictions stored in `correct_pred.txt` and `incorrect_pred.txt` files, found that the embeddings works relatively good on captial-country analogy, gender analogy and familial relationship, but badly on city-state analogy, country-currency analogy and grammar rules, which seem to be hard for this approach.

Deliverables: Discussions for all the parts. Plots for parts **4.1**, **4.4.1**, **4.4.2**. Code for all the parts as a separate Python file `hw4-pca.py`.

Problem 5: Project Launchpad