Machine Learning Exam Notes

Tomáš Jelínek

January 22, 2024

Contents

Disclaimer

These notes are based on the lecture slides from NPFL129 course: Introduction to Machine Learning with Python in winter semester 2023/24 which are under CC-BY-SA-4.0 license. The notes are not guaranteed to be correct and are not a substitute for the lecture. They are intended to be used as a study aid and should not be used as the only source of information for the exam.

Artificial Intelligence such as GPT-4, GitHub Copilot and possibly others were used in the proccess of writing this document.

License

Part I
Lecture 1

Q1.1 Explain why we need separate train and test data? What is generalization and how the concept relates to underfitting and overfitting? [10]

Why Separate Train and Test Data:

- To evaluate the performance of a machine learning model reliably.
- Training data is used to fit the model, while test data assesses its performance on unseen data.
- Prevents overfitting, ensuring the model generalizes well to new, unseen data.

Generalization:

- The ability of a model to perform well on new, unseen data.
- Indicates how well the model learns the underlying patterns, not just memorizing the training data.

Relation to Underfitting and Overfitting:

- **Underfitting:** Model is too simple, fails to capture underlying patterns in data, leading to poor performance on both training and test data.
- **Overfitting:** Model is too complex, captures noise along with patterns in the training data, leading to poor generalization on test data.

Q1.2 Define prediction function of a linear regression model and write down L2-regularized mean squared error loss. [10]

Prediction Function: Given an input vector $x \in \mathbb{R}^D$, the prediction function f for linear regression is defined as:

$$f(x; w, b) = w^T x + b$$

where w is the weight vector, b is the bias term, and T denotes the transpose of w.

L_2-Regularized Mean Squared Error Loss: The L_2-regularized mean squared error loss (also known as Ridge Regression) for a dataset with N samples is defined as:

$$L(w, b) = \frac{1}{2N} \sum_{i=1}^{N} (f(x_i; w) - t_i)^2 + \lambda \|w\|^2$$

where t_i is the true target value for the i-th sample, λ is the regularization parameter, and $\|w\|^2$ denotes the L_2 norm of the weight vector, which is the sum of the squares of its components.

Q1.3 Starting from unregularized sum of squares error of a linear regression model, show how the explicit solution can be obtained, assuming $X^T X$ is regular. [20]

In order to find a minimum of $\frac{1}{2} \sum_{i=1}^{N} (x_i^T w - t_i)^2$, we can inspect values where the derivative of the error function is zero, with respect to all weights w_j.

$$\frac{\partial}{\partial w_j} \frac{1}{2} \sum_{i=1}^{N} (x_i^T w - t_i)^2 = \frac{1}{2} \sum_{i=1}^{N} 2(x_i^T w - t_i) x_{ij} = \sum_{i=1}^{N} x_{ij} (x_i^T w - t_i)$$

Therefore, we want for all j that $\sum_{i=1}^{N} x_{ij} (x_i^T w - t_i) = 0$. We can rewrite the explicit sum into $X_{*,j}^T (Xw - t) = 0$, then write the equations for all j together using matrix notation as $X^T (Xw - t) = 0$, and finally, rewrite to

$$X^T X w = X^T t.$$

The matrix $X^T X$ is of size $D \times D$. If it is regular, we can compute its inverse and therefore

$$w = (X^T X)^{-1} X^T t.$$

Part II
Lecture 2

Q2.1 Describe standard gradient descent and compare it to stochastic (i.e., online) gradient descent and minibatch stochastic gradient descent. [10]

Standard gradient descent, also known as batch gradient descent, computes the gradient of the cost function with respect to the parameters (w) for the entire training dataset:

$$w \leftarrow w - \alpha \nabla_w E(w)$$

where α is the learning rate.

Stochastic Gradient Descent (SGD), or online gradient descent, on the other hand, updates the parameters for each training example:

$$\nabla_w E(w) \approx \nabla_w L(y(x_i; w), t_i)$$

This method is noisier but can converge faster for large datasets.

Minibatch SGD is a compromise between the two, updating the parameters for a small subset of the training data:

$$\nabla_w E(w) \approx \frac{1}{B} \sum_{i=1}^{B} \nabla_w L(y(x_i; w), t_i)$$

This approach aims to balance the computational efficiency of standard gradient descent with the faster convergence of SGD.

Q2.2 Write an L_2-regularized minibatch SGD algorithm for training a linear regression model, including the explicit formulas of the loss function and its gradient. [10]

The loss function for L_2-regularized linear regression is given by:

$$E(w) = \frac{1}{2}\mathbb{E}_{(x,t)\sim p_{\text{data}}}[(x^T w - t)^2] + \frac{\lambda}{2}\|w\|^2$$

where w are the weights, x is the input, t is the target, and λ is the regularization parameter.

The gradient of the loss function with respect to the weights is:

$$\nabla_w E(w) \approx \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} ((x_i^T w - t_i) x_i) + \lambda w$$

where $\mathcal{B}$ is a minibatch of examples.

Pseudocode of the minibatch SGD algorithm

Require: Dataset $\{X \in \mathbb{R}^{N \times D}, t \in \mathbb{R}^N\}$, learning rate $\alpha \in \mathbb{R}_+$, L_2 strength $\lambda \in \mathbb{R}$
Ensure: Weights $w \in \mathbb{R}^D$ minimizing the regularized MSE of a linear regression model.

1: Initialize w randomly
2: **repeat**
3: Sample a minibatch $\mathcal{B}$ of examples with indices $\mathcal{B}$
4: Compute gradient g according to $\nabla_w E(w)$ using $\mathcal{B}$
5: Update w: $w \leftarrow w - \alpha \cdot g$
6: **until** convergence or maximum number of iterations is reached

---

Q2.3 Does the SGD algorithm for linear regression always find the best solution on the training data? If yes, explain under what conditions it happens, if not explain why it is not guaranteed to converge. [20]

Stochastic Gradient Descent (SGD) for linear regression does not always guarantee finding the best solution on the training data. It converges to the global optimum if the following conditions are met:

- The loss function is convex and continuous.
- The learning rate α_i meets the Robbins-Monro conditions, which are:
  - $\alpha_i > 0$
  - $\sum_{i=1}^{\infty} \alpha_i = \infty$

- $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$
- The third condition ensures that $\alpha_i \to 0$ as $i \to \infty$.

When these conditions are satisfied, SGD converges to the unique optimum of convex problems. However, for non-convex loss functions, SGD is not guaranteed to find the global minimum; it may converge to a local minimum instead. The noise in the gradient estimation due to the stochastic nature of the algorithm can also affect convergence. Thus, while SGD can perform well in practice, especially for large datasets, it doesn't always find the best solution due to these factors.

Q2.4 After training a model with SGD, you ended up with a low training error and a high test error. Using the learning curves, explain what might have happened and what steps you might take to prevent this from happening. [10]

The learning curves might indicate that while the training loss decreases over time, the test loss decreases initially but then starts to increase. This scenario suggests that the **model is overfitting** to the training data. Overfitting occurs when a model learns the training data too well, including noise and details that do not generalize to unseen data. Consequently, the model performs well on the training data but poorly on the test data.

To prevent overfitting, you can take the following steps:

1. Use regularization techniques such as L_1 (LASSO) or L_2 (Ridge) to penalize large weights in the model.
2. Implement early stopping based on validation performance to halt training before overfitting occurs.
3. Increase the size of the training set if possible, to provide the model with more generalizable examples.
4. Simplify the model by reducing its complexity to prevent it from capturing noise in the data. (Make less features, use less layers, etc.)

These methods can help in guiding the model to generalize better to unseen data and thus improve its test performance.

Another reason might be that the model **failed to converge**. In this case, you can try to increase the number of iterations or decrease the learning rate to improve convergence.

Q2.5 You were provided with a fixed training set and a fixed test set and you are supposed to report model performance on that test set. You need to decide what hyperparameters to use. How will you proceed and why?. [5]

To determine the best hyperparameters for a model given a fixed training and test set, the following procedure should be employed:

1. **Split the training set:** Divide the training set into a smaller training set and a validation set.

2. **Hyperparameter tuning:** Use the smaller training set to train different models with various hyperparameter configurations. (Grid Search, Random Search, Hyperband, SMAC, etc.)
3. **Validation:** Evaluate the performance of each model on the validation set.
4. **Selection:** Choose the hyperparameters that yield the best performance on the validation set.
5. **Final Model:** Train a new model on the full training set using the selected hyperparameters.
6. **Testing:** Report the model's performance on the fixed test set.

This procedure is crucial because it helps to estimate the model's performance on unseen data and prevents overfitting to the training set. The validation set acts as a proxy for the test set, allowing for an unbiased evaluation of hyperparameter choices.

Q2.6 What method can be used for normalizing feature values? Explain why it is useful. [5]

Feature normalization can be achieved through methods such as Min-Max normalization and Z-score standardization. These methods are useful for several reasons:

- **Min-Max Normalization:** Scales the features to a fixed range, typically [0, 1]. It is given by the formula:
$$x'_{i,j} = \frac{x_{i,j} - \min_k x_{k,j}}{\max_k x_{k,j} - \min_k x_{k,j}}$$
This method is beneficial when we need to bound our features within a specific scale without distorting differences in the ranges of values.
- **Z-score Standardization:** Transforms the features to have a mean of zero and a standard deviation of one. The formula is:
$$x'_{i,j} = \frac{x_{i,j} - \bar{x}_j}{\sigma_j}$$
This is particularly useful in optimization algorithms that require features on a comparable scale for efficient learning.

Additionally, techniques similar to PCA, such as Principal Component Analysis itself, can be used for feature scaling and reduction. PCA transforms the data into a new coordinate system, reducing dimensionality and potentially improving model performance by removing noise and redundancy in the data.

Part III
Lecture 3

Q3.1 Define binary classification, write down the perceptron algorithm and show how a prediction is made for a given example. [10]

Binary classification is the task of classifying the elements of a given set into two groups based on a classification rule. In binary classification, the output variable can take only two values, typically denoted as 0 and 1, or -1 and 1 in some contexts.

The perceptron algorithm is a binary classifier that linearly separates these two classes. The algorithm iteratively adjusts the weights based on the training data. Given a set of features x and a target t, the perceptron rule updates the weights w as follows:

if $t_i(x_i^T w) \leq 0$ **then**
  $w \leftarrow w + t_i x_i$
end if

To make a prediction $\hat{y}$ for a new example with feature vector x, the perceptron uses the sign of the dot product between the features and weights:

$$\hat{y} = \text{sign}(x^T w)$$

where sign is an activation function that maps positive values to +1 and non-positive values to -1.

Prediction Example

Given a new input x and trained weights w, the perceptron prediction is computed as:

$$\hat{y} = \text{sign}(x^T w + b)$$

where b is the bias term of the perceptron. If $\hat{y}$ is positive, the input is classified into one class, and if it is negative, it is classified into the other class.

Q3.2 Define entropy, cross-entropy, Kullback-Leibler divergence, and prove the Gibbs inequality. [20]

Entropy $H(P)$ for a discrete random variable with probability distribution P is defined as:

$$H(P) = -\sum_x P(x) \log P(x)$$

It measures the expected level of 'surprise' or uncertainty inherent in the variable's possible outcomes.

Cross-entropy $H(P, Q)$ between two discrete probability distributions P and Q is defined as:

$$H(P, Q) = -\sum_x P(x) \log Q(x)$$

It measures the expected number of bits (if the log is in base 2) required to identify an event from a set of possibilities if a wrong distribution Q is used instead of the true distribution P.

Kullback-Leibler divergence $D_{KL}(P||Q)$ from Q to P is defined as:

$$D_{KL}(P||Q) = H(P, Q) - H(P) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

It measures how one probability distribution diverges from a second, expected probability distribution.

Proof of Gibbs Inequality: We want to prove that $D_{KL}(P||Q) \geq 0$, with equality if and only if $P = Q$.

Using the log sum inequality $\log \frac{a}{b} \leq \frac{a}{b} - 1$ with equality only if $a = b$, we have:

$$
\begin{aligned}
H(P) - H(P,Q) &= \sum_x P(x) \log \frac{Q(x)}{P(x)} \\
&\leq \sum_x P(x) \left(\frac{Q(x)}{P(x)} - 1 \right) \\
&= \sum_x Q(x) - \sum_x P(x) \\
&= 0
\end{aligned}
$$

since $\sum_x P(x) = 1$ and $\sum_x Q(x) = 1$. The inequality is strict unless $P(x) = Q(x)$ for all x, which proves Gibbs Inequality.

Q3.3 Explain the notion of likelihood in maximum likelihood estimation. [5]

Likelihood in the context of maximum likelihood estimation (MLE) is a function that measures the probability of observing the given data under different parameter values of a statistical model. For a set of independent and identically distributed (i.i.d) data points $X = \{x_1, x_2, \ldots, x_N\}$, the likelihood of a parameter w is defined as:

$$
L(w) = \prod_{i=1}^{N} P_{\text{model}}(x_i; w)
$$

where $P_{\text{model}}(x_i; w)$ is the probability of observing the specific data point x_i under the model parameterized by w.

In MLE, we seek the parameter w that maximizes this likelihood function, which is equivalent to maximizing the probability of observing the given data. While the likelihood itself is not a probability distribution, it serves as a scoring function that indicates how well the model with a particular set of parameters explains the observed data. Maximizing the likelihood function leads to finding the parameter values that make the observed data most probable under the assumed model.

Q3.4 Describe maximum likelihood estimation, as minimizing NLL, cross-entropy, and KL divergence. [20]

Let $X = \{x_1, x_2, \ldots, x_N\}$ be training data drawn independently from the data-generating distribution p_{data}. We denote the empirical data distribution as $\hat{p}_{\text{data}}$, where $\hat{p}_{\text{data}}(x) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[x_i = x]$. Let $p_{\text{model}}(x; w)$ be a family of distributions.

The maximum likelihood estimation of w is:

$$
w_{\text{MLE}} = \arg\max_w p_{\text{model}}(X; w) = \arg\max_w \prod_{i=1}^{N} p_{\text{model}}(x_i; w)
$$

$$
= \arg\min_w - \sum_{i=1}^{N} \log p_{\text{model}}(x_i; w)
$$

$$
= \arg\min_w \mathbb{E}_{x \sim \hat{p}_{\text{data}}} [- \log p_{\text{model}}(x; w)]
$$

$$= \arg\min_w H(\hat{p}_{\text{data}}(x), p_{\text{model}}(x; w))$$

$$= \arg\min_w D_{KL}(\hat{p}_{\text{data}}(x)||p_{\text{model}}(x; w)) + H(\hat{p}_{\text{data}}(x))$$

For MLE generalized to the conditional case, where the goal is to predict t given x:

$$w_{\text{MLE}} = \arg\max_w p_{\text{model}}(t|x; w) = \arg\max_w \prod_{i=1}^{N} p_{\text{model}}(t_i|x_i; w)$$

$$= \arg\min_w -\sum_{i=1}^{N} \log p_{\text{model}}(t_i|x_i; w)$$

$$= \arg\min_w \mathbb{E}_{(x,t)\sim\hat{p}_{\text{data}}}[-\log p_{\text{model}}(t|x; w)]$$

$$= \arg\min_w H(\hat{p}_{\text{data}}(t|x), p_{\text{model}}(t|x; w))$$

$$= \arg\min_w D_{KL}(\hat{p}_{\text{data}}(t|x)||p_{\text{model}}(t|x; w)) + H(\hat{p}_{\text{data}}(t|x))$$

Where $H(\hat{p}_{\text{data}})$ is the entropy of the empirical data distribution and D_{KL} is the Kullback-Leibler divergence. The terms are defined such that the conditional entropy is $H(\hat{p}_{\text{data}}) = \mathbb{E}_{(x,t)\sim\hat{p}_{\text{data}}}[-\log(\hat{p}_{\text{data}}(t|x))]$ and the conditional cross-entropy is $H(\hat{p}_{\text{data}}, p_{\text{model}}) = \mathbb{E}_{(x,t)\sim\hat{p}_{\text{data}}}[-\log(p_{\text{model}}(t|x;$ The negative log-likelihood (NLL) is equivalent to cross-entropy or Kullback-Leibler divergence in the context of MLE.

Q3.5 Considering binary logistic regression model, write down its parameters (including their size) and explain how prediction is performed (including the formula for the sigmoid function). Describe how we can interpret the outputs of the linear part of the model as logits. [10]

In a binary logistic regression model, the prediction $\hat{y}$ is based on the probability that a given input x belongs to a particular class C_1, which is modeled using the logistic function σ. The parameters of the model include:

- Weight vector $w \in \mathbb{R}^D$, where D is the number of features.
- Bias $b \in \mathbb{R}$.

The logistic regression model makes predictions using the sigmoid function σ applied to the linear combination of the input features and the model weights:

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{e^z + 1}$$

Given an input x, the linear part of the logistic regression model computes z as:

$$z = x^T w + b$$

The final prediction $\hat{y}$ is given by:

$$\hat{y}(x; w) = \sigma(z) = \sigma(x^T w + b)$$

The output of the linear part $x^T w + b$ can be interpreted as logits, which are the log odds of the probability that x belongs to class C_1 before the sigmoid transformation. Logits can take any real value, and transforming them through the sigmoid function maps them to the $(0, 1)$ interval, representing probabilities.

Q3.6 Write down an L2-regularized minibatch SGD algorithm for training a binary logistic regression model, including the explicit formulas of the loss function and its gradient. [20]

To train the logistic regression, we use MLE (maximum likelihood estimation). The loss for a minibatch $\mathcal{X} = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$ is given by

$$E(w) = \frac{1}{N} \sum_i -\log(p(C_{t_i}|x_i; w)) + \frac{\lambda}{2}\|w\|^2$$

The logistic regression model uses the sigmoid function as the activation function, which is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

The parameters are updated using the SGD algorithm as follows:

1. Initialize the weight vector $\vec{w} \leftarrow \vec{0}$ or randomly.
2. Repeat until convergence:
   - Compute the gradient for a minibatch $\mathcal{B}$:
   $$\vec{g} \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\vec{w}} \left(-\log(p(C_{t_i}|\vec{x}_i; \vec{w}))\right) + \lambda \vec{w}$$
   - Update the weights:
   $$\vec{w} \leftarrow \vec{w} - \alpha \vec{g}$$

Logistic Regression Gradient

Consider the log-likelihood of logistic regression $\log p(t|\vec{x}; \vec{w})$. For brevity, we denote $\bar{y}(\vec{x}; \vec{w}) = \vec{x}^T \vec{w}$ simply as $\bar{y}$ in the following computation.

Given that for $t \sim \text{Ber}(\phi)$ we have $p(t) = \phi^t (1 - \phi)^{1-t}$, we can rewrite the log-likelihood as:

$$\log p(t|\vec{x}; \vec{w}) = \log \sigma(\bar{y})^t (1 - \sigma(\bar{y}))^{1-t}$$

This simplifies to:

$$t \cdot \log(\sigma(\bar{y})) + (1 - t) \cdot \log(1 - \sigma(\bar{y}))$$

The gradient of the logistic regression's likelihood with respect to the weights $\vec{w}$ is derived as follows:

$$\begin{aligned}
\nabla_{\vec{w}} - \log p(t|\vec{x}; \vec{w}) &= \nabla_{\vec{w}} \left(-t \log(\sigma(\hat{y})) - (1 - t) \log(1 - \sigma(\hat{y}))\right) \\
&= \nabla_{\vec{w}} \left(-t \log(\sigma(\vec{x}^T \vec{w})) - (1 - t) \log(1 - \sigma(\vec{x}^T \vec{w}))\right) \\
&= -t \cdot \frac{1}{\sigma(\hat{y})} \cdot \nabla_{\vec{w}} \sigma(\hat{y}) + (1 - t) \cdot \frac{1}{1 - \sigma(\hat{y})} \cdot \nabla_{\vec{w}} (-\sigma(\hat{y})) \\
&= \left(-t + t\sigma(\hat{y}) + \sigma(\hat{y}) - t\sigma(\hat{y})\right) \vec{x} \\
&= \left(\sigma(\vec{x}^T \vec{w}) - t\right) \vec{x}
\end{aligned}$$

where $\hat{y} = \vec{x}^T\vec{w}$ and the gradient of the sigmoid function $\sigma(x)$ is $\sigma(x)(1-\sigma(x))$. The resulting gradient is used to update the weights in the SGD algorithm.

Therefor the gradient of the loss function is:

$$\nabla_{\vec{w}} E(\vec{w}) = \frac{1}{N}\sum_i \left(\sigma(\vec{x}_i^T\vec{w}) - t_i\right)\vec{x}_i + \lambda\vec{w}$$

Part IV
Lecture 4

Q4.1 Define mean squared error and show how it can be derived using MLE. [10]

Mean Squared Error (MSE) is commonly used as a loss function for regression problems and can be derived from Maximum Likelihood Estimation (MLE) when we assume that the target variables, t, conditioned on the inputs, $\vec{x}$, are normally distributed with a mean equal to the output of the model, $y(\vec{x};\vec{w})$, and variance σ^2. Under this assumption, the probability distribution for t is given by $p(t|\vec{x};\vec{w}) = \mathcal{N}(t; y(\vec{x};\vec{w}), \sigma^2)$.

Applying MLE, we look for the parameters $\vec{w}$ that maximize the likelihood of the observed data, which is equivalent to minimizing the negative log-likelihood. This leads to the MSE as follows:

$$\begin{aligned}
\vec{w}_{\text{MLE}} &= \arg\max_{\vec{w}} p(\vec{t}|\vec{X};\vec{w}) = \arg\min_{\vec{w}} \sum_{i=1}^{N} -\log p(t_i|\vec{x}_i;\vec{w}) \\
&= \arg\min_{\vec{w}} -\sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(t_i - y(\vec{x}_i;\vec{w}))^2}{2\sigma^2}\right)\right) \\
&= \arg\min_{\vec{w}} -N\log\left((2\pi\sigma^2)^{-\frac{1}{2}}\right) - \sum_{i=1}^{N}\frac{(t_i - y(\vec{x}_i;\vec{w}))^2}{2\sigma^2} \\
&= \arg\min_{\vec{w}} \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y(\vec{x}_i;\vec{w}) - t_i)^2.
\end{aligned}$$

Ignoring the constant $\frac{1}{2\sigma^2}$, we obtain the familiar form of the MSE:

$$E(\vec{w}) = \frac{1}{N}\sum_{i=1}^{N}(y(\vec{x}_i;\vec{w}) - t_i)^2.$$

This derivation shows that when we assume a normal distribution for the model errors, the MLE approach naturally leads to the MSE as the loss function to be minimized.

Q4.2 Considering K-class logistic regression model, write down its parameters (including their size) and explain how prediction is performed (including the formula for the softmax function). Describe how we can interpret the outputs of the linear part of the model as logits. [10]

To extend the binary logistic regression to a K-class case, we define the model parameters as a weight matrix $W \in \mathbb{R}^{D\times K}$, where D is the number of features and K is the number of classes.

Each column $W_{*,i}$ corresponds to the weights associated with class i.

Predictions are made using the softmax function applied to the linear outputs, known as logits. For an input vector $\vec{x}$, the logits are given by $\vec{y}(\vec{x}; W) = W^T\vec{x}$, and the softmax function is defined as:

$$\text{softmax}(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

for each class i, where $\vec{z}$ is the vector of logits.

Therefore, the probability that $\vec{x}$ belongs to class i is:

$$p(C_i|\vec{x}; W) = \text{softmax}(\vec{y}(\vec{x}; W))_i = \frac{e^{\vec{x}^T W_{*,i}}}{\sum_{j=1}^{K} e^{\vec{x}^T W_{*,j}}}$$

The linear part of the model $\vec{x}^T W$ can be interpreted as logits because they represent the log odds before passing through the softmax function. The softmax function normalizes these log odds to probabilities that sum to one across all classes.

Training a K-class logistic regression model typically involves using the cross-entropy loss function, which for a given data point (x_i, t_i) is:

$$E(W) = -\sum_{i=1}^{N} \log p(C_{t_i}|\vec{x}_i; W)$$

This loss function is minimized using optimization algorithms such as minibatch stochastic gradient descent (SGD).

Q4.3 Explain the relationship between the sigmoid function and softmax. [5]

The softmax function is a generalization of the sigmoid function to the case where there are multiple classes. For binary classification ($K = 2$), the softmax function simplifies to the sigmoid function. Specifically, the sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

which is the probability of a single class (e.g., class 1 in binary classification).

The softmax function, which is used for multinomial logistic regression with K classes, is defined as:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

for each class i. When $K = 2$, this reduces to:

$$\text{softmax}([x, 0]) = \frac{e^x}{e^x + e^0} = \frac{1}{1 + e^{-x}}$$

which is identical to the sigmoid function. Therefore, the softmax function can be seen as an extension of the sigmoid function from binary to multiclass classification, where the output for each class is the normalized exponential function of the logits, ensuring that the class probabilities sum to one.

The sigmoid function thus can be represented as a softmax function applied to a vector with two elements, where one element is the logit x and the other is zero. This connection shows the versatility of softmax as a multi-class sigmoid function.

Q4.4 Write down an L2-regularized minibatch SGD algorithm for training a K-class logistic regression model, including the explicit formulas of the loss function and its gradient. [20]

To train a K-class logistic regression model, we use the minibatch stochastic gradient descent (SGD) with L2 regularization. The loss function is the regularized negative log-likelihood, and the gradient takes into account the regularization term.

Algorithm

1. **Input:** Input dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$, target labels $\mathbf{t} \in \{0, 1, \ldots, K-1\}^N$, learning rate $\alpha \in \mathbb{R}^+$, and regularization parameter $\lambda \in \mathbb{R}^+$.
2. **Model Parameters:** Initialize weight matrix $\mathbf{W} \in \mathbb{R}^{D \times K}$ and bias vector $\mathbf{b} \in \mathbb{R}^K$ either to zero or with random values.
3. **Optimization:** Repeat until convergence:
   - Compute the gradient of the loss with respect to $\mathbf{W}$ for a minibatch $\mathcal{B}$:
   $$\mathbf{g} \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\mathbf{W}} \left(-\log(p(C_{t_i} | \mathbf{x}_i; \mathbf{W})) \right) + \lambda \mathbf{W}$$
   - Update the weights:
   $$\mathbf{W} \leftarrow \mathbf{W} - \alpha \mathbf{g}$$

Loss Function

The regularized loss function for a minibatch $\mathcal{B}$ is:

$$E(\mathbf{W}) = -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \log(p(C_{t_i} | \mathbf{x}_i; \mathbf{W})) + \frac{\lambda}{2} \|\mathbf{W}\|_F^2$$

where $\|\mathbf{W}\|_F^2$ is the Frobenius norm of $\mathbf{W}$, representing the L2 regularization term.

Gradient

The gradient of the loss function with L2 regularization is:

$$\nabla_{\mathbf{W}} E(\mathbf{W}) = -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (p(C_{t_i} | \mathbf{x}_i; \mathbf{W}) - 1_t) \mathbf{x}_i^T + \lambda \mathbf{W}$$

Note that 1_t is a one-hot vector with a 1 in the position corresponding to the target class t.

Q4.5 Prove that decision regions of a multiclass logistic regression are convex. [10]

To prove the convexity of decision regions in multiclass logistic regression, consider two points x_A and x_B in the same decision region R_k. The decision criterion for logistic regression is based on the linear functions $x^T W$, where W is the weight matrix. A point x is in region R_k if and only if

$$\hat{y}(x)_k = x^T W_k$$

is the largest among all class scores. For two points $x_A, x_B \in R_k$, and any $\lambda \in [0, 1]$, their convex combination $x = \lambda x_A + (1 - \lambda)x_B$ also satisfies

$$\hat{y}(x)_k = \lambda \hat{y}(x_A)_k + (1 - \lambda)\hat{y}(x_B)_k$$

Given that both $\hat{y}(x_A)_k$ and $\hat{y}(x_B)_k$ are the largest scores for their respective points, $\hat{y}(x)_k$ will also be the largest score for the convex combination, placing x in R_k. This holds for any convex combination of points in R_k, thus R_k is convex.

Q4.6 Considering a single-layer MLP with D input neurons, H hidden neurons, K output neurons, hidden activation f, and output activation a, list its parameters (including their size) and write down how the output is computed. [10]

A single-layer Multilayer Perceptron (MLP) with D input neurons, H hidden neurons, and K output neurons, uses the following parameters:

- Hidden layer weights $W^{(h)} \in \mathbb{R}^{D \times H}$
- Hidden layer biases $b^{(h)} \in \mathbb{R}^{H}$
- Output layer weights $W^{(y)} \in \mathbb{R}^{H \times K}$
- Output layer biases $b^{(y)} \in \mathbb{R}^{K}$

The activation functions for the hidden and output layers are denoted by f and a respectively. The output is computed as follows:

For a single input $x \in \mathbb{R}^D$:

1. Hidden layer activations: $h = f(x^T W^{(h)} + b^{(h)})$
2. Output predictions: $y = a(h^T W^{(y)} + b^{(y)})$

For a batch of inputs $X \in \mathbb{R}^{N \times D}$:

1. Batch hidden layer activations: $H = f(XW^{(h)} + 1b^{(h)})$
2. Batch output predictions: $Y = a(HW^{(y)} + 1b^{(y)})$

where 1 is a column vector of ones for bias broadcasting over the batch.

Q4.7 List the definitions of frequently used MLP output layer activations (the ones producing parameters of a Bernoulli distribution and a categorical distribution). Then write down three commonly used hidden layer activations (sigmoid, tanh, ReLU). [10]

Output Layer Activations

- **Identity (Regression):** $\text{identity}(x) = x$
- **Sigmoid (Binary Classification):** $\sigma(x) = \frac{1}{1+e^{-x}}$
- **Softmax (K-class Classification):** $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$

Hidden Layer Activations

- **Sigmoid:** $\sigma(x) = \frac{1}{1+e^{-x}}$
- **Tanh:** $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$
- **ReLU:** $\text{ReLU}(x) = \max(0, x)$

Part V
Lecture 5

Q5.1 Considering a single-layer MLP with D input neurons, a ReLU hidden layer with H units and a softmax output layer with K units, write down the explicit formulas of the gradient of all the MLP parameters (two weight matrices and two bias vectors), assuming input x, target t, and negative log likelihood loss. [20]

Assuming an MLP with D input neurons, a ReLU hidden layer with H units, and a softmax output layer with K units, we compute the gradients of the loss L with respect to the weight matrices $W^{(h)}, W^{(y)}$ and bias vectors $b^{(h)}, b^{(y)}$ given an input x, a target t, and using the negative log likelihood loss.

Let $x \in \mathbb{R}^D$ be the input vector, $h \in \mathbb{R}^H$ be the output of the hidden layer, and $y \in \mathbb{R}^K$ be the output of the network. The negative log likelihood loss for a correct class c is given by $L = -\log(y_c) = -\log(p(C|x))$.

Forward Pass:

$$
\begin{aligned}
h^{(in)} &= x^T W^{(h)} + b^{(h)} \\
h &= \text{ReLU}(h^{(in)}) \\
y^{(in)} &= h^T W^{(y)} + b^{(y)} \\
y &= \text{softmax}(y^{(in)})
\end{aligned}
$$

Backward Pass (Gradients):

$$
\begin{aligned}
\frac{\partial L}{\partial y_k} &= -\frac{t_k}{y_k} \\
\frac{\partial L}{\partial \mathbf{y}^{(in)}} &= \mathbf{y} - \mathbf{t} \quad (\text{since } \sum_k t_k = 1) \\
\frac{\partial L}{\partial \mathbf{W}^{(y)}} &= \mathbf{h} \left(\frac{\partial L}{\partial \mathbf{y}^{(in)}} \right)^\top \\
\frac{\partial L}{\partial \mathbf{b}^{(y)}} &= \frac{\partial L}{\partial \mathbf{y}^{(in)}} \\
\frac{\partial L}{\partial \mathbf{h}} &= \mathbf{W}^{(y)} \frac{\partial L}{\partial \mathbf{y}^{(in)}}^\top \\
\frac{\partial L}{\partial \mathbf{h}^{(in)}} &= \begin{cases} \frac{\partial L}{\partial \mathbf{h}} & \text{if } \mathbf{h}^{(in)} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (\text{ReLU gradient}) \\
\frac{\partial L}{\partial \mathbf{W}^{(h)}} &= \mathbf{x} \left(\frac{\partial L}{\partial \mathbf{h}^{(in)}} \right)^\top \\
\frac{\partial L}{\partial \mathbf{b}^{(h)}} &= \frac{\partial L}{\partial \mathbf{h}^{(in)}}
\end{aligned}
$$

Gradient of Loss with respect to Output Probabilities y

$$\frac{\partial L}{\partial \mathbf{y}} = -\frac{\mathbf{t}}{\mathbf{y}}$$

Gradient of Output Probabilities with respect to Logits $\mathbf{y^{(in)}}$

The softmax function for a class k is given by $y_k = \frac{e^{y_k^{(\text{in})}}}{\sum_j e^{y_j^{(\text{in})}}}$. Its derivative with respect to the logits $y_i^{(\text{in})}$ is:

$$\frac{\partial y_k}{\partial y_i^{(\text{in})}} = \begin{cases} y_k(1 - y_i) & \text{if } i = k, \\ -y_k y_i & \text{if } i \neq k. \end{cases}$$

Chain Rule Application for Loss Gradient with respect to Logits

$$\begin{aligned}
\frac{\partial L}{\partial y_i^{(\text{in})}} &= \sum_k \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial y_i^{(\text{in})}} \\
&= \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial y_i^{(\text{in})}} + \sum_{k \neq i} \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial y_i^{(\text{in})}} \\
&= -\frac{t_i}{y_i} y_i (1 - y_i) - \sum_{k \neq i} \frac{t_k}{y_k} (-y_k y_i) \\
&= -t_i + t_i y_i + \sum_{k \neq i} t_k y_i \\
&= -t_i + y_i \left(t_i + \sum_{k \neq i} t_k \right) \\
&= -t_i + y_i \sum_k t_k \\
&= y_i - t_i \quad (\text{since } \sum_k t_k = 1 \text{ for one-hot encoded targets})
\end{aligned}$$

Gradient of Loss with respect to Output Layer Weights $\mathbf{W}^{(y)}$

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{W}^{(y)}} &= \frac{\partial L}{\partial \mathbf{y}^{(\text{in})}} \frac{\partial \mathbf{y}^{(\text{in})}}{\partial \mathbf{W}^{(y)}} \\
&= (\mathbf{y} - \mathbf{t})^\top \mathbf{h}
\end{aligned}$$

Gradient of Loss with respect to Output Layer Biases $\mathbf{b}^{(y)}$

$$\frac{\partial L}{\partial \mathbf{b}^{(y)}} = \mathbf{y} - \mathbf{t}$$

Gradient of Loss with respect to Hidden Layer Outputs $\mathbf{h}$

$$\frac{\partial L}{\partial \mathbf{h}} = \mathbf{W}^{(y)} (\mathbf{y} - \mathbf{t})$$

Gradient of Loss with respect to Hidden Layer Pre-Activation $\mathbf{h}^{(\text{in})}$

$$\frac{\partial L}{\partial \mathbf{h}^{(\text{in})}} = \frac{\partial L}{\partial \mathbf{h}} \cdot \mathbb{I}(\mathbf{h}^{(\text{in})} > 0)$$

where $\mathbb{I}$ is the indicator function, yielding 1 for elements where the condition is true and 0 otherwise, which corresponds to the derivative of the ReLU activation function.

Gradient of Loss with respect to Hidden Layer Weights $\mathbf{W}^{(h)}$

$$\frac{\partial L}{\partial \mathbf{W}^{(h)}} = \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{h}^{(\text{in})}}$$

Gradient of Loss with respect to Hidden Layer Biases $\mathbf{b}^{(h)}$

$$\frac{\partial L}{\partial \mathbf{b}^{(h)}} = \frac{\partial L}{\partial \mathbf{h}^{(\text{in})}}$$

Update Rules:

$$W^{(h)} = W^{(h)} - \alpha \frac{\partial L}{\partial W^{(h)}}$$
$$b^{(h)} = b^{(h)} - \alpha \frac{\partial L}{\partial b^{(h)}}$$
$$W^{(y)} = W^{(y)} - \alpha \frac{\partial L}{\partial W^{(y)}}$$
$$b^{(y)} = b^{(y)} - \alpha \frac{\partial L}{\partial b^{(y)}}$$

In these equations, α represents the learning rate, and the derivatives with respect to $h^{(in)}$ take into account the ReLU activation, which is zero for negative inputs and equal to the derivative of the loss with respect to h otherwise. The derivative with respect to $y^{(in)}$ is computed as the difference between the output probabilities y and the one-hot encoded target vector t.

Q5.2 Formulate Universal Approximation Theorem ('89). [10]

Let $\phi(x) : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded and nondecreasing continuous function. (Later a proof was given also for $\phi = \text{ReLU}$ and even for any nonpolynomial function.)

For any $\epsilon > 0$ and any continuous function $f : [0,1]^D \to \mathbb{R}$, there exists $H \in \mathbb{N}$, $\mathbf{v} \in \mathbb{R}^H$, $\mathbf{b} \in \mathbb{R}^H$ and $\mathbf{W} \in \mathbb{R}^{D \times H}$, such that if we denote

$$F(\mathbf{x}) = \mathbf{v}^T \phi(\mathbf{W}^T \mathbf{x} + \mathbf{b}) = \sum_{i=1}^{H} v_i \phi(\mathbf{x}^T \mathbf{W}_{*,i} + b_i),$$

where ϕ is applied elementwise, then for all $\mathbf{x} \in [0,1]^D$:

$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon.$$

Q5.3 How do we search for a minimum of a function $f(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}$ subject to equality constraints $g_1(\mathbf{x}) = 0, \ldots, g_m(\mathbf{x}) = 0$? [10]

We search for a minimum of a function $f(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}$ subject to equality constraints $g_1(\mathbf{x}) = 0, \ldots, g_m(\mathbf{x}) = 0$ using the method of Lagrange multipliers. This involves finding a point $\mathbf{x} \in \mathbb{R}^D$ and a set of multipliers $\lambda_1, \ldots, \lambda_m \in \mathbb{R}$ such that the gradient of the Lagrangian function $\mathcal{L}(\mathbf{x}, \lambda)$ with respect to both $\mathbf{x}$ and λ is zero.

The Lagrangian is defined as:

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}),$$

where $\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda) = 0$ and $\nabla_{\lambda} \mathcal{L}(\mathbf{x}, \lambda) = 0$. This gives us a system of equations which, when solved, gives the points $\mathbf{x}$ that minimize $f(\mathbf{x})$ subject to the constraints.

Q5.4 Prove which categorical distribution with N classes has maximum entropy. [10]

We want to find a categorical distribution $\mathbf{p} = (p_1, \ldots, p_N)$ that maximizes entropy, subject to the constraints:

- $p_i \geq 0$ for all i,
- $\sum_{i=1}^{N} p_i = 1$.

The entropy $H(\mathbf{p})$ for a categorical distribution is given by:

$$H(\mathbf{p}) = -\sum_{i=1}^{N} p_i \log(p_i)$$

We form the Lagrangian $\mathcal{L}$ to include the equality constraint:

$$\mathcal{L}(\mathbf{p}, \lambda) = -\sum_{i=1}^{N} p_i \log(p_i) + \lambda \left(\sum_{i=1}^{N} p_i - 1 \right)$$

Taking the derivative of $\mathcal{L}$ with respect to p_i and setting it to zero:

$$0 = \frac{\partial \mathcal{L}}{\partial p_i} = -\log(p_i) - 1 + \lambda$$

Solving for p_i, we get:

$$p_i = e^{\lambda - 1}$$

Since all p_i must satisfy the equality constraint $\sum_{i=1}^{N} p_i = 1$, substituting p_i gives:

$$\sum_{i=1}^{N} e^{\lambda - 1} = 1$$

$$N e^{\lambda - 1} = 1$$

$$e^{\lambda - 1} = \frac{1}{N}$$

$$p_i = \frac{1}{N}$$

Therefore, each p_i is $\frac{1}{N}$, indicating that the distribution with maximum entropy is the uniform distribution.

Q5.5 Consider derivation of softmax using maximum entropy principle, assuming we have a dataset of N examples (x_i, t_i), $x_i \in \mathbb{R}^D$, $t_i \in \{1, 2, \ldots, K\}$. Formulate the three conditions we impose on the searched $\pi : \mathbb{R}^D \to \mathbb{R}^K$, and write down the Lagrangian to be minimized. [20]

Given a dataset of N examples (x_i, t_i) where $x_i \in \mathbb{R}^D$ and $t_i \in \{1, 2, \ldots, K\}$, we want to derive a softmax function using the maximum entropy principle. The softmax function $\pi(x)$ must satisfy the following conditions:

1. For all $x \in \mathbb{R}^D$ and each class k, the predicted probability $\pi(x)_k \geq 0$.
2. For each input x, the probabilities must sum up to 1: $\sum_{k=1}^{K} \pi(x)_k = 1$.
3. The expected value of the predicted distribution should match the empirical distribution: $\frac{1}{N} \sum_{i=1}^{N} \pi(x_i)_k = \frac{1}{N} \sum_{i=1}^{N} [t_i = k]$ for each class k.

The Lagrangian $\mathcal{L}$, incorporating these constraints with Lagrange multipliers λ and μ_k. We want to minimize $-\sum_{i=1}^{N} H(\pi(x_i))$ given

- for $1 \leq i \leq N$, $1 \leq k \leq K$: $\pi(x_i)_k \geq 0$,
- for $1 \leq i \leq N$: $\sum_{k=1}^{K} \pi(x_i)_k = 1$,
- for $1 \leq j \leq D$, $1 \leq k \leq K$: $\sum_{i=1}^{N} \pi(x_i)_k x_{i,j} = \sum_{i=1}^{N} [t_i = k] x_{i,j}$.

We therefore form a Lagrangian (ignoring the first inequality constraint):

$$\mathcal{L} = \sum_{i=1}^{N}\sum_{k=1}^{K} \pi(x_i)_k \log(\pi(x_i)_k) - \sum_{j=1}^{D}\sum_{k=1}^{K} \lambda_{j,k} \left(\sum_{i=1}^{N} \pi(x_i)_k x_{i,j} - [t_i = k] x_{i,j} \right) - \sum_{i=1}^{N} \beta_i \left(\sum_{k=1}^{K} \pi(x_i)_k - 1 \right).$$

Q5.6 Define precision (including true positives and others), recall, F_1 score, and F_β score (we stated several formulations for F_1 and F_β scores; any one of them will do). [10]

The confusion matrix is a table used to describe the performance of a classification model:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

Table 1: Confusion Matrix

Precision quantifies the number of correct positive predictions made. It is defined as:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{TP} + \text{False Positives (FP)}}$$

Recall measures the proportion of actual positives correctly identified. It is defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{False Negatives (FN)}}$$

The F_1 score is the harmonic mean of precision and recall. It is defined as:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F_β score generalizes the F_1 score by weighing recall more heavily than precision. It is defined as:

$$F_\beta = (1 + \beta^2) \times \frac{\text{Precision} \times \text{Recall}}{(\beta^2 \times \text{Precision}) + \text{Recall}}$$

Q5.7 Explain the difference between micro-averaged and macro-averaged F_1 scores. [10]

The micro-averaged F_1 score (F_1^μ) aggregates the contributions of all classes to compute the average score. It is given by:

$$F_1^\mu = 2 \times \frac{Precision_{micro} \times Recall_{micro}}{Precision_{micro} + Recall_{micro}} = \frac{2\sum TP}{2\sum TP + \sum FP + \sum FN}$$

The macro-averaged F_1 score (F_1^M) is the arithmetic mean of the F_1 scores per class, treating all classes equally:

$$F_1^M = \frac{1}{K} \sum_{k=1}^{K} F_{1k}$$

where F_{1k} is the F_1 score of the k-th class.

Q5.8 Explain (using examples) why accuracy is not a suitable metric for unbalanced target classes, e.g., for a diagnostic test for a contagious disease. [5]

Accuracy is defined as the ratio of correctly predicted observations to the total observations. In the context of unbalanced datasets, particularly in disease diagnosis where the disease prevalence is low, a model might predict "no disease" for all patients and still achieve high accuracy. For example:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Observations}}$$

Consider a dataset with 1000 individuals where only 10 have the disease. A model that predicts "no disease" for everyone would have an accuracy of:

$$\frac{0 + 990}{10 + 990} = \frac{990}{1000} = 99\%$$

Despite the high accuracy, the model fails to detect any true disease cases, demonstrating the inadequacy of accuracy as a performance metric in such scenarios. It is more informative to look at metrics such as precision, recall, and the F_1 score in cases of class imbalance.

Part VI
Lecture 6

Q6.1 Explain how is the TF-IDF weight of a given document-term pair computed. [5]

The TF-IDF weight of a document-term pair is given by:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

where:

- $\text{TF}(t, d)$ is the term frequency, defined as the number of times term t appears in document d, normalized or not.
- $\text{IDF}(t, D)$ is the inverse document frequency, calculated as:

  $$\text{IDF}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|(optionaly + 1))}$$

  In this formula, N is the total number of documents in the corpus D, and $|\{d \in D : t \in d\}|$ is the number of documents where the term t appears (i.e., df_t, the document frequency of t).

The TF-IDF score increases with the number of times a word appears in the document but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

Q6.2 Define conditional entropy, mutual information, write down the relation between them, and finally prove that mutual information is zero if and only if the two random variables are independent (you do not need to prove statements about D_{KL}). [10]

Conditional entropy $H(Y|X)$ quantifies the expected value of the entropy of Y given that the value of X is known. It is defined as:

$$H(Y|X) = -\sum_{x\in X, y\in Y} P(x,y)\log P(y|x).$$

Mutual information $I(X;Y)$ measures the amount of information that one random variable contains about another random variable. It is defined as:

$$I(X;Y) = \sum_{x\in X, y\in Y} P(x,y)\log\frac{P(x,y)}{P(x)P(y)}.$$

The relationship between conditional entropy and mutual information can be expressed as:

$$I(X;Y) = H(Y) - H(Y|X).$$

This equation implies that mutual information is the reduction in uncertainty about Y due to the knowledge of X.

The mutual information is symmetrical, so

$$I(X;Y) = I(Y;X) = H(X) - H(X|Y) = H(Y) - H(Y|X).$$

Therefore

$$I(X;Y) = D_{KL}(P(X,Y)||P(X)P(Y))$$

Q6.3 Show that TF-IDF terms can be considered portions of suitable mutual information. [10]

Let $\mathcal{D}$ be a collection of documents and $\mathcal{T}$ a collection of terms. We can express the mutual information between a document d and a term t using their probabilities and the TF-IDF measure.

- The probability of selecting a document d uniformly at random from $\mathcal{D}$ is $P(d) = \frac{1}{|\mathcal{D}|}$.
- The information content of a document $I(d) = H(\mathcal{D}) = \log|\mathcal{D}|$.
- The probability of a term t occurring in a document d is $P(t|d) = \frac{|\{d\in\mathcal{D}: t\in d\}|}{|\mathcal{D}|}$.
- The information content of a term t in a document d is $I(t|d) = \log|\{d\in\mathcal{D} : t\in d\}|$.
- The difference in information content of a document with and without a term is the IDF: $I(d) - I(t|d) = \log|\mathcal{D}| - \log|\{d\in\mathcal{D} : t\in d\}| = \mathrm{IDF}(t)$.

The mutual information $I(\mathcal{D};\mathcal{T})$ is calculated as:

$$I(\mathcal{D};\mathcal{T}) = \sum_{d,t\in d} P(d)\cdot P(t|d)\cdot(I(d) - I(t|d)).$$

Given the definitions of TF and IDF, we can write:

$$I(\mathcal{D};\mathcal{T}) = \frac{1}{|\mathcal{D}|}\sum_{d,t\in d}\mathrm{TF}(t,d)\cdot\mathrm{IDF}(t).$$

Thus, we can interpret the TF-IDF weight as a portion of the mutual information between the collection of documents $\mathcal{D}$ and the collection of terms $\mathcal{T}$, where each TF-IDF value corresponds to a "bit of information" for a document-term pair.

Q6.4 Explain the concept of word embedding in the context of MLP and how it relates to representation learning. [5]

Word embedding is a technique in representation learning where words from a vocabulary are associated with vectors of real numbers, effectively capturing their semantic meanings in a continuous vector space. Semantically similar words are positioned closely in this space.

In the realm of Multilayer Perceptrons (MLPs), these embeddings serve as the input layer. Each word is represented by a unique, learnable vector, rather than a high-dimensional, sparse one-hot vector. Throughout the training phase, the MLP fine-tunes these embeddings via back-propagation, based on the context in which words appear.

This approach is a key part of representation learning because it enables MLPs to internalize language subtleties directly from data, surpassing the need for manual feature engineering. It allows the network to capture complex syntactic and semantic word relationships, enhancing its performance on Natural Language Processing (NLP) tasks such as sentiment analysis, translation, and text categorization.

Word embeddings are the cornerstone of modern NLP models and are integrated into more advanced neural architectures like Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and Transformers, driving forward the state-of-the-art in various NLP applications.

Q6.5 Describe the skip-gram model trained using negative sampling. [10]

The skip-gram model aims to learn word embeddings that predict the context words given a target word. For a given word in the vocabulary, the model outputs a probability distribution over all words to be the 'context' words.

The Skip-gram model with negative sampling (SGNS) enhances training efficiency by altering the objective function. Instead of predicting the presence of context words among all words in the vocabulary, SGNS focuses on distinguishing the actual context words from a number of randomly sampled 'negative' words.

Given a pair of words (target word w and context word c), the model learns to assign high probabilities to the actual context words and low probabilities to the negative samples. This is achieved by maximizing the log probability of the sigmoid function σ, which represents the model's estimated probability that a given pair (target-context) is valid:

$$\log \sigma(\mathbf{v}_c^\top \mathbf{e}_w) \tag{1}$$

Here, $\mathbf{e}_w$ is the embedding of the target word, and $\mathbf{v}_c$ is the embedding of the context word. The negative samples are then introduced in the loss function, where the model maximizes the log probability of the sigmoid of the negative of the dot product between the target word embedding and the embedding of a negative sample c_i:

$$-\log \sigma(-\mathbf{v}_{c_i}^\top \mathbf{e}_w) \tag{2}$$

The overall objective combines these terms for the positive and negative samples, summing across all positive pairs observed in the training data and a set of negative samples drawn according to a noise distribution, often related to word frequency.

By focusing only on a small subset of negative samples rather than the entire vocabulary, SGNS reduces computational complexity significantly, allowing for faster training over large datasets.

Algorithm 1 Word2Vec Training Algorithm

```
Initialize word embeddings for the target words and context words
for each epoch do
    for each word w in the corpus do
        context = GetContextWords(w, window_size)
        for each context word c in context do
            // Positive sample training
            z = DotProduct(embedding[w], embedding[c])
            Update embeddings using gradient ascent with the objective: log(σ(z))
            // Negative sampling training
            for i = 1 to k do // k is the number of negative samples
                c_i = SampleNegativeWord()
                z_i = DotProduct(embedding[w], embedding[c_i])
                Update embeddings using gradient ascent with the objective: log(σ(−z_i))
            end for
            // Update the embeddings
            embedding[w] = embedding[w] - learning_rate * gradient(w)
            embedding[c] = embedding[c] - learning_rate * gradient(c)
            for i = 1 to k do
                embedding[c_i] = embedding[c_i] - learning_rate * gradient(c_i)
            end for
        end for
    end for
end for
```

Q6.6 How would you proceed to train a part-of-speech tagger (i.e., you want to assign each word with its part of speech) if you only could use pre-trained word embeddings and MLP classifier?. [5]

I don't know mucb about this question, so this is what GPT-4 came up with:

Data Preparation

- Acquire a dataset with words annotated with POS tags.
- Tokenize and embed words using pre-trained vectors.
- Construct input features from contextual embeddings.
- Encode POS tags with one hot vectors.

MLP Architecture

- Input layer: Size equal to concatenated embedding vectors.
- Hidden layers: One or more, with activation functions like ReLU.
- Output layer: Softmax with a neuron for each POS tag.

Training Process

- Divide data into training, validation, and test sets.

- Train with categorical cross-entropy and optimizers like Adam.
- Use validation set for hyperparameter tuning and early stopping.

Evaluation

- Assess performance on the test set.
- Use classification metrics such as accuracy and F1 score.

Post-Processing

- Optionally apply sequence modeling techniques like CRFs.

Part VII
Lecture 7

Q7.1 Describe k-nearest neighbors prediction, both for regression and classification. Define Lp norm and describe uniform, inverse, and softmax weighting. [10]

Regression

For regression, k-NN predicts the target value by a weighted average of the targets of the k nearest neighbors:

$$t = \frac{\sum_i w_i \cdot t_i}{\sum_j w_j}$$

Classification

For classification, k-NN uses voting among the k nearest neighbors. For uniform weights:

$$\text{class} = \text{mode}\{t_1, t_2, \ldots, t_k\}$$

With non-uniform weights, the predicted class maximizes the weighted sum of targets:

$$\text{class} = \arg\max \sum_i w_i \cdot t_{i,k}$$

Lp-norm

The L_p-norm is defined as:

$$\|x - y\|_p = \left(\sum_{i=1}^{D} |x_i - y_i|^p \right)^{1/p}$$

Weighting Methods

- Uniform: $w_i = 1$
- Inverse: $w_i = \frac{1}{\text{distance}(x, x_i)}$
- Softmax: $w_i = \frac{\exp(-\text{distance}(x, x_i))}{\sum_j \exp(-\text{distance}(x, x_j))}$

Q7.2 Show that L2-regularization can be obtained from a suitable prior by Bayesian inference (from the MAP estimate). [10]

Assuming a Gaussian prior for model parameters $\mathbf{w}$ with zero mean and variance σ^2, the prior distribution is given as $p(\mathbf{w}_i) = \mathcal{N}(\mathbf{w}_i; 0, \sigma^2)$. Consequently, the prior over all weights $\mathbf{w}$ is $p(\mathbf{w}) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{w}_i; 0, \sigma^2) = \mathcal{N}(\mathbf{w}; 0, \sigma^2\mathbf{I})$. The maximum a posteriori (MAP) estimation is then:

$$\begin{aligned}
\mathbf{w}_{\text{MAP}} &= \arg\max_{\mathbf{w}} p(\mathbf{X}|\mathbf{w})p(\mathbf{w}) \\
&= \arg\max_{\mathbf{w}} \prod_{i=1}^{N} p(\mathbf{x}_i|\mathbf{w})p(\mathbf{w}) \\
&= \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \left(-\log p(\mathbf{x}_i|\mathbf{w}) - \log p(\mathbf{w})\right).
\end{aligned}$$

Incorporating the Gaussian prior probability, we get the L2-regularized objective:

$$\mathbf{w}_{\text{MAP}} = \arg\min_{\mathbf{w}} \left[\sum_{i=1}^{N} -\log p(\mathbf{x}_i|\mathbf{w}) + \frac{D}{2}\log(2\pi\sigma^2) + \frac{\|\mathbf{w}\|^2}{2\sigma^2} \right],$$

which is the L2-regularization term.

Q7.3 Write down how $\mathbf{p(C_k|x)}$ is approximated in a Naive Bayes classifier, explicitly state the Naive Bayes assumption, and show how is the prediction performed. [10]

The Naive Bayes classifier approximates the conditional probability $p(C_k|x)$ using Bayes' theorem and the naive independence assumption. This assumption states that all features x_d are independent given the class C_k. Therefore, the joint probability of the feature vector x given the class C_k can be expressed as the product of individual probabilities:

$$p(x \mid C_k) = \prod_{d=1}^{D} p(x_d \mid C_k).$$

Using Bayes' theorem, the posterior probability for class C_k given the feature vector x is then:

$$p(C_k \mid x) = \frac{p(x \mid C_k)p(C_k)}{p(x)},$$

where $p(x)$ is the evidence term, typically ignored during prediction as it remains constant across all classes.

The prediction for a new sample x is performed by choosing the class C_k that maximizes this posterior probability:

$$\hat{C} = \arg\max_{k} p(C_k \mid x) = \arg\max_{k} \left(\prod_{d=1}^{D} p(x_d \mid C_k) \right) p(C_k).$$

This approach allows for efficient computation and prediction in high-dimensional feature spaces.

Q7.4 Considering a Gaussian naive Bayes, describe how are $p(x_d|C_k)$ modeled (what distribution and which parameters does it have) and how we estimate it during fitting. [10]

In Gaussian Naive Bayes, the conditional probability $p(x_d \mid C_k)$ for a continuous feature x_d given a class C_k is modeled by a normal distribution:

$$p(x_d \mid C_k) = \mathcal{N}(x_d \mid \mu_{d,k}, \sigma^2_{d,k}).$$

The parameters $\mu_{d,k}$ and $\sigma^2_{d,k}$ of this distribution are estimated from the training data using maximum likelihood estimation (MLE). For each feature d and class k, the MLE of the mean $\mu_{d,k}$ is computed as:

$$\mu_{d,k} = \frac{1}{N_k} \sum_{i=1}^{N_k} x_{i,d},$$

where $x_{i,d}$ is the i-th training sample of feature d that belongs to class C_k, and N_k is the number of samples in class C_k.

The variance $\sigma^2_{d,k}$ is estimated as:

$$\sigma^2_{d,k} = \frac{1}{N_k} \sum_{i=1}^{N_k} (x_{i,d} - \mu_{d,k})^2.$$

In practice, to avoid the issue of zero variance, a smoothing term α is often added to the variance estimate:

$$\sigma^2_{d,k} = \frac{1}{N_k + \alpha} \sum_{i=1}^{N_k} (x_{i,d} - \mu_{d,k})^2 + \alpha.$$

The smoothing term α is a hyperparameter that can be tuned using cross-validation.

Q7.5 Considering a Bernoulli naive Bayes, describe how are $p(x_d|C_k)$ modeled (what distribution and which parameters does it have) and how we estimate it during fitting. [10]

In Bernoulli Naive Bayes, the probability of a binary feature x_d given a class C_k, denoted as $p(x_d|C_k)$, is modeled using a Bernoulli distribution with parameter $p_{d,k}$. This parameter represents the probability of feature d being present in a sample of class C_k.

$$p(x_d \mid C_k) = p_{d,k}^{x_d} \cdot (1 - p_{d,k})^{(1-x_d)}.$$

The likelihood of class C_k given the feature vector x is then:

$$p(C_k \mid x) \propto \left(\prod_{d=1}^{D} p_{d,k}^{x_d} \cdot (1 - p_{d,k})^{(1-x_d)} \right) p(C_k),$$

where D is the number of binary features.

Taking the logarithm, we obtain:

$$\log p(C_k \mid x) + c = \log p(C_k) + \sum_d \left(x_d \log p_{d,k} + \log(1 - p_{d,k}) \right) = b_k + x^T w_k,$$

where c is a constant that does not depend on C_k and is not needed for prediction.

The prediction is made by:

$$\arg\max_k \log p(C_k \mid x) = \arg\max_k b_k + x^T w_k.$$

The parameter $p_{d,k}$ is estimated during training as the relative frequency of the feature d in samples of class C_k:

$$p_{d,k} = \frac{1}{N_k} \sum_{i=1}^{N_k} x_{i,d},$$

where $x_{i,d}$ is the presence or absence of feature d in the i-th sample, and N_k is the total number of samples in class C_k.

To prevent zero probabilities and to handle unseen features in the training data, smoothing is applied. The smoothed estimate for $p_{d,k}$ with Laplace smoothing is:

$$p_{d,k} = \frac{\sum_{i=1}^{N_k} x_{i,d} + \alpha}{N_k + 2\alpha},$$

where α is a smoothing parameter, typically set to 1.

Part VIII
Lecture 8

Q8.1 Prove that independent discrete random variables are uncorrelated. [10]

Given two discrete random variables X and Y, they are said to be independent if the joint probability distribution can be expressed as the product of their marginal distributions:

$$P(X = x, Y = y) = P(X = x)P(Y = y) \quad \text{for all } x \text{ and } y.$$

The covariance of X and Y is defined as:

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])].$$

For independent variables, the expectation of the product is the product of the expectations:

$$\begin{aligned}\text{Cov}(X, Y) &= \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] - \mathbb{E}[X]\mathbb{E}[Y] + \mathbb{E}[X]\mathbb{E}[Y] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y].\end{aligned}$$

Since X and Y are independent:

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y].$$

Substituting this into the covariance formula gives:

$$\text{Cov}(X, Y) = \mathbb{E}[X]\mathbb{E}[Y] - \mathbb{E}[X]\mathbb{E}[Y] = 0.$$

Therefore, if X and Y are independent, their covariance is zero, implying that they are uncorrelated.

Q8.2 Write down the definition of covariance and Pearson correlation coefficient ρ , including its range. [10]

The covariance between two random variables X and Y is a measure of the joint variability of X and Y. It is defined as:

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$

where $\mathbb{E}$ denotes the expected value.

The Pearson correlation coefficient, denoted as ρ or r, is defined as:

$$\rho = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}}$$

$$r = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

where:

- ρ is used when the full expectation is computed (population Pearson correlation coefficient);
- r is used when estimating the coefficient from data (sample Pearson correlation coefficient);
- $\bar{x}$ and $\bar{y}$ are sample estimates of the respective means.

The range of the Pearson correlation coefficient is from -1 to 1, inclusive. A value of 1 implies a perfect positive linear relationship between variables, -1 implies a perfect negative linear relationship, and 0 implies no linear relationship.

Q8.3 Explain how are the Spearman's rank correlation coefficient and the Kendall rank correlation coefficient computed (no need to describe the Pearson correlation coefficient). [10]

Spearman's Rank Correlation Coefficient

Spearman's rank correlation coefficient, denoted by ρ, is a nonparametric measure of rank correlation (statistical dependence between the rankings of two variables). It assesses how well the relationship between two variables can be described using a monotonic function. The formula for Spearman's rank correlation coefficient is:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} = \frac{\text{cov}(R(X), R(Y))}{\sigma_{R(X)} \sigma_{R(Y)}}$$

where d_i is the difference between the ranks of corresponding variables and n is the number of observations. Hence $d_i = rank(X_i) - rank(X_i)$

Kendall Rank Correlation Coefficient

Kendall rank correlation coefficient, denoted by τ, measures the ordinal association between two quantities. It is defined as:

$$\tau = \frac{\sum_{i<j} \text{sign}(x_j - x_i)\text{sign}(y_j - y_i)}{\frac{1}{2}n(n-1)}$$

where $\text{sign}(x)$ is the sign function, which returns -1, 0, or 1 depending on the sign of x, and n is the number of observations.

Both coefficients range from -1 to 1. A coefficient of 1 implies a perfect agreement, -1 implies perfect disagreement, and 0 implies the absence of association.

Q8.4 Describe setups where a correlation coefficient might be a good evaluation metric. [5]

Correlation coefficients are powerful statistical tools used to measure the strength and direction of a linear relationship between two variables. They are particularly useful in various setups, including:

1. **Inter-Annotator Agreement (IAA):** In tasks where annotators provide ratings on a continuous scale, Pearson's or Spearman's correlation coefficients can measure the agreement level between annotators. High correlation indicates strong agreement and consistency in the ratings.
2. **Predictive Model Evaluation:** In regression analysis, correlation coefficients help to assess the predictive performance of a model. For example, a high Pearson correlation between predicted and actual values indicates a model with good predictive capabilities.
3. **Feature Selection:** Correlation coefficients can identify the relationships between features in a dataset. Features with high correlation may carry redundant information, and one of them can be removed to reduce dimensionality.
4. **Time Series Analysis:** In financial or economic studies, correlation coefficients can analyze the relationship between different time series data, like stock prices or interest rates, helping to identify trends and co-movements.
5. **Psychometrics:** They are used to establish the reliability of psychometric tests, ensuring that the test measures what it is supposed to measure consistently over time.
6. **Usability Studies:** They can evaluate the relationship between different usability metrics, such as time on task and error rates, to understand user behavior better.

It's important to note that while correlation coefficients can indicate the presence of a relationship, they do not imply causation. Additionally, they are sensitive to outliers, which can disproportionately influence the results. Thus, they should be used alongside other statistical methods to provide a comprehensive evaluation.

Q8.5 Describe under what circumstance correlation can be used to assess validity of evaluation metrics. [5]

Correlation is particularly useful for assessing the validity of evaluation metrics in tasks lacking a clear ground truth. It is applicable in situations such as:

- SoTA - benchmarking in competitions, where metrics should correlate with expert evaluations to rank participants accurately.
- Grammar checking. (β value)
- Validating new metrics against established ones to ensure they measure similar constructs.
- Tasks with subjective assessments, like sentiment analysis, where metrics should correlate with human intuition.
- Ensuring consistency of a metric across different datasets or conditions to confirm its reliability.

In these cases, high correlation with human judgment or between different evaluation metrics indicates that the metrics are capturing aspects of performance that are consistent with human evaluation or other reliable metrics.

Q8.6 Define Cohen's κ and explain what it is used for when preparing data for machine learning. [10]

Cohen's kappa coefficient (κ) is a statistical measure used to evaluate the inter-annotator agreement for qualitative (categorical) items. It is defined as:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

where p_o is the relative observed agreement among raters, and p_e is the hypothetical probability of chance agreement.

This metric is utilized in machine learning to assess the consistency of annotations provided by different human experts. It serves multiple purposes:

- **Data Reliability:** Ensures that the data labels used for training machine learning models are consistent and reliable.
- **Annotator Performance:** Helps in evaluating the performance of annotators and can be used to filter out unreliable annotations.
- **Cultural Insights:** Low values of κ might indicate cultural differences or subjectivity in the data, providing insights into potential biases.
- **Model Benchmarking:** Sets a benchmark for machine learning performance, as achieving high accuracy beyond IAA is often unrealistic and might indicate overfitting or data leakage.

By quantifying the agreement level, Cohen's kappa allows for more informed decisions in the data curation process, ultimately leading to the development of more robust machine learning models.

Q8.7 Considering an averaging ensemble of M models, prove the relation between the average mean squared error of the ensemble and the average error of the individual models, assuming the model errors have zero means and are uncorrelated. [20]

Let $y_i(x)$ be the prediction of model i for an input x with true target t, and $\varepsilon_i(x)$ be the error of model i such that $y_i(x) = t + \varepsilon_i(x)$. The mean squared error (MSE) of model i is:

$$\mathbb{E}\left[(y_i(x) - t)^2\right] = \mathbb{E}\left[\varepsilon_i^2(x)\right].$$

For an ensemble of M models, the ensemble prediction is the average of individual predictions:

$$y_{\text{ensemble}}(x) = \frac{1}{M}\sum_{i=1}^{M} y_i(x).$$

The MSE of the ensemble is then:

$$\mathbb{E}\left[(y_{\text{ensemble}}(x) - t)^2\right] = \mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M}\varepsilon_i(x)\right)^2\right].$$

Assuming that errors $\varepsilon_i(x)$ are uncorrelated and have zero means, we have:

$$\mathbb{E}\left[\varepsilon_i(x)\varepsilon_j(x)\right] = 0 \quad \text{for} \quad i \neq j.$$

Therefore, the MSE of the ensemble simplifies to:

$$\mathbb{E}\left[\left(\frac{1}{M}\sum_{i=1}^{M}\varepsilon_i(x)\right)^2\right] = \frac{1}{M^2}\mathbb{E}\left[\left(\sum_{i=1}^{M}\varepsilon_i(x)\right)^2\right] + \frac{1}{M^2}\sum_{i,j}\mathbb{E}\left[\varepsilon_i(x)\varepsilon_j(x)\right] = \frac{1}{M}\mathbb{E}\left[\frac{1}{M}\sum_{i}\varepsilon_i^2(x)\right],$$

Hence, the average MSE of the ensemble is $\frac{1}{M}$ times the average MSE of the individual models.

Q8.8 Explain knowledge distillation: what it is used for, describe how it is done. [10]

Knowledge distillation is a process of transferring knowledge from a large or complex model (teacher) to a smaller model (student). It is especially useful for deploying deep learning models on devices with limited computational power.

The algorithm for knowledge distillation is as follows:

1. Train a large model or ensemble of models on a dataset to create the teacher model.
2. Use the trained teacher model to generate soft target distributions (also known as pseudolikelihood) for each instance in the dataset.
3. Train the student model to mimic the output distribution of the teacher model by using a loss function that compares the soft targets with the predictions of the student model.

The intuition behind knowledge distillation is that the probability distribution output by the teacher model provides a richer signal than a one-hot encoded target, as it contains information about the relationships between different classes. This rich signal can be used to train the student model more effectively.

Part IX
Lecture 9

Q9.1 In a regression decision tree, state what values are kept in internal nodes, define the squared error criterion and describe how is a leaf split during training (without discussing splitting constraints). [10]

In a regression decision tree, the internal nodes represent the feature and the corresponding threshold used to split the data, while the leaves store the predicted value for the target variable. The predicted value at each leaf is the average of the target values of the samples that fall into that leaf.

The squared error criterion C_{SE} for a node T is defined as:

$$C_{SE}(T) = \sum_{i \in I_T}(t_i - \bar{t}_T)^2,$$

where $\bar{t}_T$ is the average of the target values t_i for the samples at node T, and I_T is the set of sample indices that belong to node T.

During training, a leaf is split by selecting the feature and the threshold that minimize the summed squared error criterion of the resulting child nodes. This is achieved by:

1. Iterating over all features and all possible thresholds for each feature.
2. Splitting the node into two child nodes T_L and T_R based on the selected feature and threshold.
3. Calculating the decrease in the squared error criterion due to the split.
4. Choosing the split that results in the largest decrease in the squared error criterion.

The process continues recursively until a stopping criterion is met, which could be a maximum depth of the tree, a minimum number of samples in a leaf, or a minimal improvement to the criterion.

Q9.2 In a K-class classification decision tree, state what values are kept in internal nodes, define the Gini index and describe how is a node split during training (without discussing splitting constraints). [10]

In a classification decision tree for K classes, internal nodes represent the decision criteria, which consist of a feature and a threshold that is used to split the data into two child nodes.

The Gini index, also known as Gini impurity, is a measure used to quantify the purity of a node. It is defined for a node T as:

$$C_{\text{Gini}}(T) = |I_T| \sum_{k=1}^{K} p_T(k)(1 - p_T(k)),$$

where $p_T(k)$ is the proportion of samples of class k in node T, and I_T is the set of indices of samples that fall into node T.

During training, a node is split by selecting the feature and the threshold that produce child nodes with the lowest combined Gini index. The process is as follows:

1. Iterate over all features and thresholds.
2. Calculate the Gini index for child nodes resulting from each potential split.
3. Select the feature and threshold that minimize the weighted sum of the Gini indices of the child nodes.

This process recursively continues down the tree until a stopping criterion is reached, such as a maximum depth or a minimum node size.

Q9.3 In a K-class classification decision tree, state what values are kept in internal nodes, define the entropy criterion and describe how is a node split during training (without discussing splitting constraints). [10]

In a K-class classification decision tree, the internal nodes hold the decision rules, typically a feature and a threshold that partitions the dataset into subsets.

The entropy criterion for a node T, often used as a measure of impurity or disorder within the node, is defined as:

$$C_{\text{entropy}}(T) = -|I_T| \sum_{\substack{k=1 \\ p_T(k)\neq 0}}^{K} p_T(k) \log p_T(k),$$

where $p_T(k)$ represents the proportion of class k instances within node T, and $|I_T|$ is the number of instances in node T.

The process of splitting a node during training involves:

1. For each feature, calculate the potential splits and the resulting entropy.
2. The split that results in the largest decrease in entropy (highest information gain) is chosen.
3. This process is recursively applied to each new node until the stopping criteria are met.

Q9.4 For binary classification, derive the Gini index from a squared error loss. [20]

Consider a binary classification setting with a set of training examples belonging to a leaf node T. Let $n_T(0)$ denote the number of examples with target value 0, $n_T(1)$ the number of examples with target value 1, and p_T the proportion of examples with target value 1 in T, i.e., $p_T = \frac{n_T(1)}{n_T(0)+n_T(1)}$.

The squared error loss $L(p)$ for a prediction p is defined as:

$$L(p) = \sum_{i\in I_T} (p - t_i)^2,$$

where t_i is the target value for the i-th example.

Minimizing the squared error loss, we find that the optimal prediction p is the average target value in T, i.e., $p = p_T$. The loss for this prediction is:

$$L(p_T) = \sum_{i\in I_T} (p_T - t_i)^2 = n_T(0)(p_T - 0)^2 + n_T(1)(p_T - 1)^2.$$

Expanding the terms, we get:

$$\begin{aligned} &= \frac{n_T(0)n_T(1)^2}{(n_T(0) + n_T(1))^2} + \frac{n_T(1)n_T(0)^2}{(n_T(0) + n_T(1))^2} \\ &= \frac{(n_T(1)) + n_T(0))n_T(0)n_T(1)}{(n_T(0) + n_T(1))(n_T(0) + n_T(1))} \\ &= (n_T(0) + n_T(1))(1 - p_T)p_T = |T| \cdot p_T(1 - p_T). \end{aligned} \tag{3}$$

which is proportional to the Gini impurity measure $G(T) = 2p_T(1 - p_T)$ for a binary classification.

Q9.5 For K-class classification, derive the entropy criterion from a non-averaged NLL loss. [20]

Given a set of training examples I_T corresponding to a leaf node T in a decision tree for K-class classification, let $n_T(k)$ denote the count of examples in T with target class k. The probability of class k in T is given by $p_T(k) = \frac{n_T(k)}{|I_T|}$.

The non-averaged negative log-likelihood loss for a distribution p over K classes is defined as:

$$L(p) = \sum_{i \in I_T} -\log p_{t_i},$$

where p_{t_i} is the predicted probability of the true class t_i for the i-th example.

To minimize the NLL loss, we take its derivative with respect to p_k and set it to zero, subject to the constraint $\sum_k p_k = 1$. This yields the minimizing condition $p_k = p_T(k)$.

The value of the loss with respect to p_T then simplifies to:

$$L(p_T) = \sum_{i \in I_T} -\log p_{t_i} = -\sum_{k:p_T(k)\neq 0} n_T(k) \log p_T(k).$$

Using the definition of entropy $H(p_T)$ for the distribution p_T, we have:

$$H(p_T) = -\sum_{k:p_T(k)\neq 0} p_T(k) \log p_T(k),$$

which implies that the NLL loss is equal to the size of I_T times the entropy of p_T:

$$L(p_T) = |I_T| \cdot H(p_T).$$

This concludes the derivation, showing that minimizing the non-averaged NLL loss is equivalent to minimizing the entropy of the predicted class distribution within a leaf node of a decision tree.

Q9.6 Describe how is a random forest trained (including bagging and a random subset of features) and how is prediction performed for regression and classification. [10]

A random forest is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Training

The training process involves the following steps:

1. **Bootstrap Aggregating (Bagging):** For each tree, a bootstrap sample is drawn from the training data. This means that for a training set of size N, M samples are drawn with replacement to form a training set for the tree.

2. **Random Feature Selection:** When splitting nodes during the construction of the trees, instead of searching for the best split among all features, a random subset of features is selected, and the best split is found within this subset. This introduces diversity among the trees and is key to the success of random forests.

3. **Tree Construction:** Decision trees are constructed to the maximum depth without pruning. Each tree is grown on a different bootstrap sample of the data, and at each node, a different random subset of features is considered for splitting.

4. **Ensemble Creation:** Steps 1 to 3 are repeated to create a forest of decision trees, typically ranging from tens to hundreds of trees.

Prediction

For prediction, the responses from all trees in the forest are aggregated:

- **Regression:** The final prediction is the average of the predictions from all individual trees.
- **Classification:** Each tree gives a vote for the class, and the class receiving the majority of votes becomes the model's prediction. In the case of a tie, one class may be randomly selected, or the tie may be broken based on the class distributions.

The random forest algorithm leverages the power of multiple decision trees to reduce variance and avoid overfitting, providing robust predictions for both regression and classification tasks.

Part X
Lecture 10

Q10.1 Write down the loss function which we optimize in gradient boosted decision trees during the construction of t^{th} tree. Then define g_i and h_i and show the value w_T of optimal prediction in node T and the criterion used during node splitting. [20]

During the construction of the t^{th} tree in gradient boosting, we optimize the following loss function:

$$\begin{aligned} E^{(t)}(w_t; w_{1..t-1}) &= \sum_i \left[\ell(t_i, y^{(t-1)}(x_i; w_{1..t-1}) + y_t(x_i; w_t)) \right] + \frac{1}{2}\lambda \|w_t\|^2 \\ &\approx \sum_i \left[\ell(t_i, y^{(t-1)}(\mathbf{x}_i)) + g_i \cdot y_t(\mathbf{x}_i) + \frac{1}{2} h_i \cdot y_t^2(\mathbf{x}_i) \right] + \frac{\lambda}{2} \|\mathbf{w}\|^2. \end{aligned}$$

Here, g_i and h_i are defined as the first and second order gradients of the loss function with respect to the prediction at step $t-1$, specifically:

$$g_i = \left. \frac{\partial \ell(t_i, y)}{\partial y} \right|_{y=y^{(t-1)}(\mathbf{x}_i)} \tag{4}$$

$$h_i = \left. \frac{\partial^2 \ell(t_i, y)}{\partial y^2} \right|_{y=y^{(t-1)}(\mathbf{x}_i)}. \tag{5}$$

The optimal prediction value w_T for a node T is given by:

$$w_T^* = -\frac{\sum_{i \in I_T} g_i}{\lambda + \sum_{i \in I_T} h_i}, \tag{6}$$

where I_T denotes the set of instance indices in node T. During node splitting, we seek to maximize the gain from the split, which is computed using these gradients.

Criterion for Node Splitting

The splitting criterion in gradient boosting is based on maximizing the gain G, which is calculated as follows:

$$G = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\lambda + \sum_{i\in I_L} h_i} + \frac{(\sum_{i\in I_R} g_i)^2}{\lambda + \sum_{i\in I_R} h_i} - \frac{(\sum_{i\in I_T} g_i)^2}{\lambda + \sum_{i\in I_T} h_i}\right] - \gamma, \tag{7}$$

where I_L and I_R are the sets of instance indices for the left and right splits from node T, and γ is a regularization parameter to penalize the complexity of the model.

Q10.2 For a K-class classification, describe how to perform prediction with a gradient boosted decision tree trained for T time steps (how the individual trees perform prediction and how are the KT trees combined to produce the predicted categorical distribution). [10]

In a K-class classification task using gradient boosted decision trees, prediction after T timesteps is performed as follows. Each timestep t involves training K trees, denoted as $w_{t,k}$, with each tree predicting a value for one of the K classes. Prediction for an instance $\mathbf{x}_i$ is made by summing the outputs of all trees corresponding to each class across all timesteps and then applying the softmax function to these sums to obtain the predicted probabilities for each class:

$$\text{softmax}(\mathbf{y}(\mathbf{x}_i)) = \text{softmax}\left(\sum_{t=1}^{T} y_{t,1}(\mathbf{x}_i; w_{t,1}), \ldots, \sum_{t=1}^{T} y_{t,K}(\mathbf{x}_i; w_{t,K})\right), \tag{8}$$

where $y_{t,k}(\mathbf{x}_i; w_{t,k})$ is the output of the k-th tree at timestep t for instance $\mathbf{x}_i$. The predicted categorical distribution for instance $\mathbf{x}_i$ is then the output of the softmax function, which provides a probability distribution over the K classes.

Q10.3 What type of data are gradient boosted decision trees good for as opposed to multilayer perceptron? Explain the intuition why it is the case. [5]

Gradient boosted decision trees (GBDTs) are optimal for structured, tabular data where input features have high predictive power and clear interpretability. They can capture non-linear relationships and feature interactions without extensive preprocessing.

- Good for lower-dimensional data with meaningful features.
- Handles mixed data types (continuous, categorical).
- Requires less data preprocessing.

Multilayer perceptrons (MLPs), or neural networks, are better suited for high-dimensional data such as images or text. They excel in learning hierarchical feature representations, essential in domains where raw features are not individually informative.

- When one feature does not mean much alone.
- Ideal for high-dimensional data (images, text).
- Capable of complex feature extraction.
- Benefits from pre-trained networks.

The choice of model depends on the dataset characteristics and the problem at hand.

Part XI
Lecture 11

Q11.1 Formulate SVD decomposition of matrix X, describe properties of individual parts of the decomposition. Explain what the reduced version of SVD is. [10]

Given a matrix $X \in \mathbb{R}^{m \times n}$, the singular value decomposition (SVD) of X is a factorization of the form:

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

where:

- $U \in \mathbb{R}^{m \times m}$ is an orthogonal matrix whose columns are the left singular vectors of X,
- $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix with non-negative real numbers on the diagonal known as singular values, sorted in descending order,
- $V \in \mathbb{R}^{n \times n}$ is an orthogonal matrix whose columns are the right singular vectors of X.

Properties:

- The columns of U and V are orthonormal bases for the column space and row space of X, respectively.
- The non-zero singular values in Σ are the square roots of the non-zero eigenvalues of both $X^T X$ and XX^T.

Reduced SVD:
The reduced version of SVD is used when we want to approximate X by a matrix of lower rank k, which is less than the original rank r. It can be expressed as:

$$\tilde{X} = U_k \Sigma_k V_k^T$$

where U_k and V_k contain only the first k columns of U and V, and Σ_k contains only the top k singular values. This approximation minimizes the Frobenius norm $\|X - \tilde{X}\|_F$ among all rank-k approximations.

Q11.2 Formulate the Eckart-Young theorem. [10]

The Eckart-Young theorem states that given a matrix $X \in \mathbb{R}^{n \times m}$ and its Singular Value Decomposition (SVD), the best rank-k approximation X_k of X in terms of the Frobenius norm is obtained by retaining the first k singular values and corresponding singular vectors. Formally:

$$X_k = \sigma_1 u_1 v_1^T + \ldots + \sigma_k u_k v_k^T$$

where σ_i are the singular values, and u_i, v_i are the left and right singular vectors, respectively. This approximation minimizes the Frobenius norm of the difference between X and X_k:

$$\|X - X_k\|_F \leq \|X - B\|_F$$

for any $B \in \mathbb{R}^{n \times m}$ of rank k. The Frobenius norm is the square root of the sum of the absolute squares of its elements, and it can also be expressed as the square root of the trace

of $X^T X$. It has the important property that multiplying by an orthonormal matrix does not change the norm:

$$\|UA\|_F = \sqrt{\text{trace}((UA)^T UA)} = \sqrt{\text{trace}(A^T U^T UA)} = \sqrt{\text{trace}(A^T A)} = \|A\|_F$$

The norm is invariant under orthogonal transformations, and the best strategy to approximate X while preserving most of its norm is to remove the smallest singular values.

Q11.3 Explain how to compute the PCA of dimension M using the SVD decomposition of a data matrix X, and why it works. [10]

Principal Component Analysis (PCA) can be computed using Singular Value Decomposition (SVD) of the data matrix X. For a data matrix $X \in \mathbb{R}^{n \times m}$, where each row is a data point and each column is a feature, PCA is performed as follows:

1. Center the data by subtracting the mean of each feature from the data matrix, resulting in the mean-centered matrix $\tilde{X} = X - \bar{x}$.
2. Compute the SVD of $\tilde{X}$, which is given by $\tilde{X} = U\Sigma V^T$.
3. The columns of V (right singular vectors) correspond to the principal components of X.
4. To reduce the dimensionality to M, select the first M columns of V, and the first M singular values from Σ.
5. The projection of X onto the M-dimensional subspace is given by $X_M = XV_M$, where V_M is the matrix containing the first M columns of V.

This works because the singular values in Σ represent the amount of variance captured by each principal component, and the columns of V are the directions along which the variance is maximized. By taking the first M components, we retain the features that capture the most variance in the data.

$$\|\mathbf{X} - \bar{\mathbf{x}}\|_F^2 = \text{trace}\left((\mathbf{X} - \bar{\mathbf{x}})^T (\mathbf{X} - \bar{\mathbf{x}})\right) = N \sum_{i=1}^{D} \text{Var}(\mathbf{X}_{:,i})$$

Approximating the matrix in terms of Frobenius norm means keeping the most variance from the data. Components are ordered by how much variability in the data they capture.

$$\text{Let } \mathbf{S} = \frac{1}{N}(\mathbf{X} - \bar{\mathbf{x}})^T (\mathbf{X} - \bar{\mathbf{x}}),$$
$$\text{then PCA of } \mathbf{X} \text{ involves the eigenvectors of } \mathbf{S},$$
$$\text{denoted by the } \mathbf{V} \text{ matrix in the SVD of } \mathbf{X} - \bar{\mathbf{x}}.$$

Q11.4 Given a data matrix X, write down the algorithm for computing the PCA of dimension M using the power iteration algorithm. [20]

Given a data matrix X, the PCA of dimension M can be computed using the power iteration algorithm as follows:

1. Center the data matrix X by subtracting the mean μ of each feature.

2. Compute the covariance matrix S as $S = \frac{1}{N}(X - \mu)^T(X - \mu)$.

3. For each dimension i from 1 to M:

   (a) Initialize a random vector v_i.

   (b) Repeat until convergence (or a fixed number of iterations):

      i. Compute $v_i = Sv_i$.

      ii. Normalize $\lambda_i = \|v_i\|$.

      iii. Compute $v_i = \frac{v_i}{\lambda_i}$.

   (c) Deflate S by subtracting the outer product of v_i with itself, scaled by its eigenvalue: $S = S - \lambda_i v_i v_i^T$.

4. Construct the matrix V with columns $v_1, v_2, \ldots, v_M$ as the principal components.

5. Project the data onto the new subspace: $X_{\text{PCA}} = XV$.

This algorithm computes the top M principal components by iteratively finding the direction of maximum variance (eigenvector) and deflating the covariance matrix to remove this variance before finding the next component.

Q11.5 Describe the K-means algorithm, including the kmeans++ initialization. [20]

K-Means is a clustering algorithm that partitions N data points into K clusters. Each cluster is defined by its centroid μ_k, which is the mean of the points assigned to the cluster. The objective is to minimize the within-cluster sum of squares.

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ be a collection of N input examples, each being a D-dimensional vector $\mathbf{x}_i \in \mathbb{R}^D$. Let K, the number of target clusters, be given.

Let $z_{i,k} \in \{0, 1\}$ be binary indicator variables describing whether an input example $\mathbf{x}_i$ is assigned to cluster k, and let each cluster be specified by a point $\mu_1, \ldots, \mu_K$, usually called the cluster center.

Our objective function J, which we aim to minimize, is

$$J = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{i,k} \|\mathbf{x}_i - \mu_k\|^2.$$

Algorithm

1. **Initialization: K-means++**

   (a) Choose one data point uniformly at random as the first cluster center μ_1.

   (b) For each data point x_i, compute $D(x_i)$, the distance between x_i and the nearest cluster center that has already been chosen.

   (c) Choose one new data point at random as a new cluster center, using a weighted probability distribution where a point x_i is chosen with probability proportional to $D(x_i)^2$.

   (d) Repeat steps ii and iii until K cluster centers have been chosen.

2. **Assignment step:** Assign each data point to the nearest centroid.

$$z_{i,k} = \begin{cases} 1 & \text{if } k = \arg\min_j \|x_i - \mu_j\|^2 \\ 0 & \text{otherwise} \end{cases}$$

3. **Update step:** Recalculate centroids as the mean of all points assigned to that centroid.

$$\mu_k = \frac{\sum_{i=1}^{N} z_{i,k} x_i}{\sum_{i=1}^{N} z_{i,k}}$$

4. Repeat steps 2 and 3 until the centroids no longer change significantly or a maximum number of iterations is reached.

The K-means++ initialization helps to ensure that the initial centroids are as far from each other as possible, leading to better and faster convergence of the algorithm.

Part XII
Lecture 12

Q12.1 Considering statistical hypothesis testing, define type I errors and type II errors (in terms of the null hypothesis). Finally, define what a significance level is. [10]

In hypothesis testing, we begin with a null hypothesis, H_0, which is a statement we assume to be true until evidence suggests otherwise. An alternative hypothesis, H_1, is considered if there is sufficient evidence against H_0.

Type I and Type II Errors

- **Type I Error**: Also known as a false positive, occurs when H_0 is true, but we incorrectly reject it.
- **Type II Error**: Also known as a false negative, occurs when H_0 is false, but we fail to reject it.

The probability of making a Type I error is denoted by α, and is known as the significance level of the test. It is the threshold below which we reject H_0, commonly set at 5%.

Significance Level

The significance level α is the probability of rejecting the null hypothesis H_0 when it is actually true. It defines the sensitivity of our hypothesis test.

$$\alpha = P(\text{Type I Error}) = P(\text{Reject } H_0 | H_0 \text{ is true})$$

The lower the value of α, the less likely we are to make a Type I error, but the higher the chance of a Type II error.

Confusion Matrix

A confusion matrix helps to visualize the performance of a statistical test:

	H_0 True	H_1 True
Reject H_0	Type I Error (False Positive)	Correct Decision (True Positive)
Fail to Reject H_0	Correct Decision (True Negative)	Type II Error (False Negative)

Q12.2 Explain what a test statistic and a p-value are. [10]

Test Statistic

A **test statistic** is a standardized value derived from sample data during a hypothesis test. It is calculated to assess the strength of the evidence against the null hypothesis. Mathematically, it can be represented as:

$$T = \frac{\text{Sample Estimate} - \text{Population Parameter}}{\text{Standard Error}}$$

It quantifies how far our sample statistic deviates from what we would expect if the null hypothesis H_0 were true, under the assumption of the null hypothesis.

P-value

The **p-value** is the probability of obtaining a test statistic at least as extreme as the one actually observed, given that the null hypothesis is true. It is a measure of the evidence against the null hypothesis provided by the sample. A smaller p-value indicates stronger evidence against H_0. It is given by:

$$p\text{-value} = P(T \geq t | H_0 \text{ is true})$$

for a right-tailed test, or

$$p\text{-value} = P(T \leq t | H_0 \text{ is true})$$

for a left-tailed test, or

$$p\text{-value} = P(|T| \geq |t| | H_0 \text{ is true})$$

for a two-sided symmetric test. where t is the observed value of the test statistic.

A p-value less than the chosen significance level α (commonly 0.05) leads to the rejection of the null hypothesis.

Q12.3 Write down the steps of a statistical hypothesis test, including a definition of a p-value. [10]

The procedure for performing a statistical hypothesis test is as follows:

1. Formulate the null hypothesis H_0, which is a statement of no effect or no difference, and optionally the alternative hypothesis H_1, which is what you aim to support.
2. Choose the appropriate test statistic that will measure the degree of agreement between the sample data and the null hypothesis.
3. Compute the observed value of the test statistic from the sample data.
4. Calculate the p-value, which is the probability of observing a test statistic as extreme as, or more extreme than, the observed value, assuming the null hypothesis is true.
5. Decide whether to reject or not reject the null hypothesis by comparing the p-value to the chosen significance level α. Common choices for α include 5%, 1%, 0.5%, or 0.1%. If the p-value is less than or equal to α, reject H_0; otherwise, do not reject H_0.

P-value Definition

The p-value is defined as the probability of obtaining a test statistic at least as extreme as the one that was actually observed, under the assumption that the null hypothesis is true. It is used as a tool to decide whether to reject or not reject the null hypothesis. More detailed explanation in Q12.2.

Q12.4 Explain the differences between a one-sample test, two-sample test, and a paired test. [10]

- **One-sample test:** This test is used when we want to compare the sample mean from a single group to a known value or theoretical expectation. Common applications include testing whether the mean of a single distribution is equal to, greater than, or less than a certain value, or performing goodness of fit tests to determine if the data come from a specified distribution.
- **Two-sample test:** This type of test is applicable when two independent samples are taken from different populations, and we want to compare their means. The key assumption is that the two samples are independent of each other.
- **Paired test:** Paired tests are used when the samples are not independent but are paired in some meaningful way. For example, measurements before and after a treatment on the same subjects, or the same subjects measured under two different conditions. The test involves computing the differences between the paired measurements and then performing a one-sample test on these differences.

In a one-sample test, the null hypothesis typically states that the mean of the distribution is equal to a target value. For two-sample tests, the null hypothesis commonly states that the means of the two populations are equal. In a paired test, the null hypothesis usually states that the mean difference between the paired observations is zero.

Q12.5 When considering multiple comparison problem, define the family-wise error rate, and prove the Bonferroni correction, which allows limiting the family-wise error rate by a given α. [10]

Family-Wise Error Rate (FWER)

The family-wise error rate (FWER) is the probability of making one or more Type I errors when performing multiple hypotheses tests. Formally, for a family of m hypotheses tests, the FWER is defined as:

$$FWER = P\left(\bigcup_{i=1}^{m}(p_i \leq \alpha)\right)$$

where p_i is the p-value for the i-th hypothesis test and α is the significance level.

Bonferroni Correction

The Bonferroni correction is a method to control the FWER. It involves adjusting the significance level α by the number of hypotheses m. The corrected significance level is α/m.

Proof: Using the union bound (Boole's inequality), we have:

$$FWER = P\left(\bigcup_{i=1}^{m}(p_i \leq \alpha)\right) \leq \sum_{i=1}^{m} P(p_i \leq \alpha)$$

For independent tests, assuming a uniform distribution under the null hypothesis, the probability of a single test yielding a p-value less than α is exactly α. With the Bonferroni correction, we have:

$$P(p_i \leq \frac{\alpha}{m}) = \frac{\alpha}{m}$$

Therefore, the sum of probabilities for m independent tests is:

$$\sum_{i=1}^{m} P(p_i \leq \frac{\alpha}{m}) = m \cdot \frac{\alpha}{m} = \alpha$$

This proves that the Bonferroni correction ensures that the FWER is at most α.

Q12.6 For a trained model and a given test set with N examples and metric E, write how to estimate 95% confidence intervals using bootstrap resampling. [10]

Given a test set $\{(x_1, t_1), \ldots, (x_N, t_N)\}$, a trained model with predictions $\{y(x_1), \ldots, y(x_N)\}$, and a performance metric E, we estimate the 95% confidence intervals of the model's performance using bootstrap resampling as follows:

1. Initialize an empty list called *performances* to store the sampled performances.
2. Repeat R times, where R is a large number (commonly 1000 or more for better approximation):
   (a) Generate a bootstrap sample by sampling N examples from the test set *with replacement*.
   (b) For each sampled example, obtain the corresponding model prediction.
   (c) Compute the performance metric E for the bootstrap sample.
   (d) Append the computed performance to the *performances* list.
3. Sort the list of *performances*.
4. The 95% confidence interval is estimated by selecting the 2.5th percentile as the lower bound and the 97.5th percentile as the upper bound from the sorted *performances* list.

Formally, the 95% confidence interval CI is given by:

$$CI = \left(performances\left[\frac{R}{40}\right], performances\left[\frac{39R}{40}\right] \right)$$

This interval estimates the range within which the true performance metric of the model lies with 95% confidence.

Q12.7 For two trained models and a given test set with N examples and metric E, explain how to perform a paired bootstrap test that the first model is better than the other. [10]

Given a test set $\{(x_1, t_1), \ldots, (x_N, t_N)\}$ and predictions from two models $\{y(x_1), \ldots, y(x_N)\}$ and $\{z(x_1), \ldots, z(x_N)\}$, we aim to test the hypothesis that the first model y performs better than the second model z using the paired bootstrap test with the following steps:

1. Initialize an empty list called *differences* to store the differences in performances.
2. For a number of resamplings R, repeat:
   (a) Generate a bootstrap sample by sampling N examples from the test set *with replacement*.

(b) For each sampled example, obtain the corresponding predictions from both models y and z.

(c) Calculate the performance of both models on the sampled data using the metric E.

(d) Compute the difference in performance between the two models for each bootstrap sample and append it to the *differences* list.

3. The significance of the test is determined by the proportion of differences that are less than or equal to zero (indicating no improvement of model y over model z).

4. This proportion represents the estimated probability that model y is not better than model z. If this proportion is small (typically less than a chosen significance level such as 0.05), we reject the null hypothesis and conclude that model y is significantly better than model z.

Unfortunately, the value returned by the algorithm is not really a p-value. The reason is that the distribution of differences was obtained **under the true distribution**. However, to perform the statistical test, we require the distribution of the test statistic **under the null hypothesis**. Nevertheless, you can encounter such paired bootstrap tests *"in the wild"*.

Q12.8 For two trained models and a given test set with N examples and metric E, explain how to perform a random permutation test that the first model is better than the other with a significance level of α. [10]

Given two trained models and a test set $\{(x_1, t_1), \ldots, (x_N, t_N)\}$, with model predictions $\{y(x_1), \ldots, y(x_N)\}$ and $\{z(x_1), \ldots, z(x_N)\}$, we perform a random permutation test to determine if the first model is significantly better than the second at a significance level α with the following algorithm:

1. Initialize an empty list called *differences* to store the differences in performance.

2. Repeat for a number of resamplings R:

   (a) For each test set example, randomly permute the model predictions, effectively assigning the prediction of model y or model z to each test case.

   (b) Compute the performance of the permuted predictions using the metric E.

   (c) Record the performance and append it to the list *performances*.

3. Return the ratio of the performances which are greater than or equal to the performance of the model y.

4. The p-value is then given by this proportion. If the p-value is less than or equal to the significance level α, we reject the null hypothesis that there is no difference in performance, suggesting that the first model is significantly better.

(The calculation of the p-value is not exactly as I say here, because the algorithm actually returns β)

Part XIII
Lecture 13

Q13.1 Explain the difference between deontological and utilitarian ethics. List examples on how these theoretical frameworks can be applied in machine learning ethics. [10]

Deontological Ethics focuses on the inherent nature of actions rather than their consequences. It emphasizes adherence to predefined rules and principles, such as the Universal Declaration of Human Rights, the Ten Commandments, or Kant's Categorical Imperative. In the context of machine learning (ML), it translates to principles like beneficence, non-malevolence, privacy, non-discrimination, autonomy, and informed consent.

Utilitarian Ethics, on the other hand, is an ethical theory that emphasizes the maximization of overall happiness or well-being, thus focusing on the consequences of actions. It promotes actions that lead to the greatest overall positive impact.

In ML ethics, **deontological frameworks** might lead to ethical problems in:

- Problem definition: Some tasks may not align with fundamental ethical principles.
- Data collection: Issues like privacy invasion or non-consensual data usage.
- Model development: Ensuring models do not discriminate or violate user autonomy.

Utilitarian frameworks in ML ethics might consider:

- Model evaluation: Metrics should account for overall happiness or harm reduction.
- Model deployment: Using models in ways that maximize social good while minimizing potential harm or feedback loops that might disadvantage certain groups.

Examples:

- A deontological approach might reject any form of user data exploitation, even if it improves the performance of a recommendation system, on the principle of user autonomy.
- A utilitarian approach may justify the use of personal data if the resulting system significantly benefits a large number of users, thereby increasing overall utility.

Both approaches have their merits and challenges when applied to ML ethics. Deontological ethics provide clear guidelines but can be rigid and may lead to conflicts between principles. Utilitarian ethics offer flexibility and quantifiability but may overlook individual rights and face difficulties in defining collective well-being.

Q13.2 List a few examples of potential ethical problems related to data collection. [5]

1. **Representation Bias**: Data may not be representative of the entire population, often excluding minorities or economically disadvantaged groups.
2. **Internet Data Misrepresentation**: Data collected from the internet might disproportionately represent the views and behaviors of those who have access and are more vocal online, skewing perceptions of the general population.
3. **Historical Bias**: Data reflecting past inequalities may perpetuate these biases when used to train modern machine learning systems.

4. **Exploitation in Crowdsourcing**: Individuals hired to collect or label data, often in low-income countries, may be underpaid and work under poor conditions, which could lead to monotonous work that causes psychological harm.

5. **Non-transparent Data Collection**: Users may unknowingly provide personal data when using online services, without a clear understanding or explicit consent of how their data will be used or the implications of its use.

Q13.3 List a few examples of potential ethical problems that can originate in model evaluation. [5]

1. **Incomplete Metrics:** Evaluation metrics may not fully capture the desired outcomes. For instance, while translation fluency metrics may seem adequate, they might overlook ingrained gender biases.

2. **Macro-averaging Oversights:** Using macro-averaging in evaluation can obscure poor performance for specific user groups, often minorities, thus perpetuating discrimination.

3. **Human Resource Algorithms:** Employment recommendation systems optimized for precision may inadvertently discriminate based on gender, age, ethnicity, etc., since the system's recall—indicating the full scope of candidates, including potentially overlooked qualified individuals—is not visible.

4. **Data Mismatch:** When training and testing data do not align, minority languages could be disproportionately classified as hate speech or not safe for work, as highlighted in the paper "The Risk of Racial Bias in Hate Speech Detection" (Sap et al., ACL 2019).

5. **Feedback Loops:** Recommender systems can create feedback loops where predictions influence user behavior, which then feeds back into the training data. This can lead to echo chambers and self-affirmative groups. A notable example is how YouTube's recommendation algorithms facilitated the discovery of a category of home videos of scantily clad children by pedophiles.

Part XIV
The End

Contributing

Github repo: `https://github.com/Desperadus/mff-ml-exam-prep`

If you find any errors in this document, please create a pull request. Contributions are welcome! If you decide to do so feel free to add your name to the list of contributors below.

Contributors