DAMA61 Exams Prep

Model Overfitting and Model Selection

Overfitting & Underfitting

- **Overfitting**: When a model learns the noise in the training data, leading to poor generalization to new data.
    - **Key Functions**: `EarlyStopping`, `Dropout`
    - **Models**: Any model can overfit; commonly observed in deep neural networks and complex decision trees.
- **Underfitting**: When a model is too simple to capture the underlying structure of the data.
    - **Key Functions**: `Regularizers` (like L1, L2)
    - **Models**: Linear models, overly pruned decision trees.

Model Selection

- **Using Validation Set**: Splitting data into training, validation, and test sets.
    - **Key Functions**: `train_test_split` from `sklearn.model_selection`, `validation_split` in Keras `model.fit`
- **Incorporating Model Complexity**: Balancing model complexity to avoid overfitting and underfitting.
    - **Key Functions**: `model.add(Dense(..., kernel_regularizer=l2(0.01)))`
- **Pre-pruning (Early Stopping Rule)**: Stopping the training process before the model becomes overly complex.
    - **Key Functions**: `EarlyStopping` callback in Keras
- **Post-pruning**: Removing unnecessary nodes from a decision tree after it has been fully grown.
    - **Key Functions**: Not directly in TensorFlow/Keras, but available in `sklearn.tree.DecisionTreeClassifier` with parameters `ccp_alpha`.

Classification Algorithms and Metrics

Base Classifiers

- **Decision Trees**: A tree structure where each internal node represents a test on an attribute, each branch represents an outcome, and each leaf node represents a class label.
    - **Key Functions**: `DecisionTreeClassifier` from `sklearn.tree`
    - **Models**: `DecisionTreeClassifier()`
- **Rule-Based**: Classifiers that use a set of "if-then" rules for classification.
    - **Key Functions**: Not directly in TensorFlow/Keras; use libraries like `sklearn.tree.DecisionTreeClassifier` and convert to rules.

- **Nearest Neighbor**: Classification based on the closest training examples in the feature space.
  - **Key Functions**: `KNeighborsClassifier` from `sklearn.neighbors`
  - **Models**: `KNeighborsClassifier()`
- **Naive Bayes**: A probabilistic classifier based on Bayes' theorem with the assumption of feature independence.
  - **Key Functions**: `GaussianNB`, `MultinomialNB` from `sklearn.naive_bayes`
  - **Models**: `GaussianNB()`, `MultinomialNB()`
- **SVMs (Support Vector Machines)**: A classifier that finds the hyperplane that maximizes the margin between classes.
  - **Key Functions**: `SVC` from `sklearn.svm`
  - **Models**: `SVC()`
- **Neural Networks & Deep NNs**: Networks of nodes (neurons) that learn complex patterns in data through layers of transformations.
  - **Key Functions**: `Sequential`, `Dense`, `Conv2D`, `MaxPooling2D`, `Flatten`, `compile`, `fit` from Keras
  - **Models**: `Sequential([...])`

Ensemble Classifiers

- **Boosting**: Combines multiple weak learners to create a strong learner.
  - **Key Functions**: `AdaBoostClassifier`, `GradientBoostingClassifier` from `sklearn.ensemble`
  - **Models**: `AdaBoostClassifier()`, `GradientBoostingClassifier()`
- **Bagging (Bootstrap Aggregating)**: Reduces variance by averaging predictions from multiple models trained on different samples of the data.
  - **Key Functions**: `BaggingClassifier` from `sklearn.ensemble`
  - **Models**: `BaggingClassifier()`
- **Random Forests**: An ensemble of decision trees trained on random subsets of features and data samples.
  - **Key Functions**: `RandomForestClassifier` from `sklearn.ensemble`
  - **Models**: `RandomForestClassifier()`

Metrics for Performance Evaluation

- **Accuracy**: Measures the fraction of correct predictions.
  - **Formula**: `Accuracy = (TP + TN) / (TP + TN + FP + FN)`
  - **Key Functions**: `accuracy_score` from `sklearn.metrics`
- **Precision**: Measures the fraction of true positive instances among the retrieved instances.
  - **Formula**: `Precision = TP / (TP + FP)`
  - **Key Functions**: `precision_score` from `sklearn.metrics`

- **Recall (Sensitivity)**: Measures the fraction of true positive instances among all relevant instances.
  - **Formula**: `Recall = TP / (TP + FN)`
  - **Key Functions**: `recall_score` from `sklearn.metrics`
- **F1-score (F-score)**: Harmonic mean of precision and recall.
  - **Formula**: `F1 = 2 * (Precision * Recall) / (Precision + Recall)`
  - **Key Functions**: `f1_score` from `sklearn.metrics`
- **ROC Curve**: A graph showing the performance of a classification model at all classification thresholds.
  - **Key Functions**: `roc_curve` from `sklearn.metrics`
- **AUC (Area Under the ROC Curve)**: A single scalar value to measure the overall performance of a classifier.
  - **Key Functions**: `roc_auc_score` from `sklearn.metrics`
- **Specificity**: Measures the fraction of true negative instances among all negative instances.
  - **Formula**: `Specificity = TN / (TN + FP)`
  - **Key Functions**: `recall_score` with `pos_label=0` from `sklearn.metrics`
- **Error Rate**: Measures the fraction of incorrect predictions.
  - **Formula**: `Error Rate = 1 - Accuracy`
  - **Key Functions**: `1 - accuracy_score`
- **FP Rate**: Measures the fraction of false positive instances among all negative instances.
  - **Formula**: `FP Rate = FP / (FP + TN)`
  - **Key Functions**: Derived from confusion matrix
- **FN Rate**: Measures the fraction of false negative instances among all positive instances.
  - **Formula**: `FN Rate = FN / (FN + TP)`
  - **Key Functions**: Derived from confusion matrix
- **Power**: The probability of correctly rejecting a false null hypothesis.
  - **Key Functions**: Not directly implemented; use statistical libraries for power analysis.

Clustering Algorithms and Analysis

Clustering Algorithms

- **K-means (Partitional Clustering)**: A method of vector quantization that partitions n observations into k clusters.
  - **Key Functions**: `KMeans` from `sklearn.cluster`
  - **Models**: `KMeans(n_clusters=k)`
  - **Centroid**: The center of a cluster.
  - **SSE (Sum of Squared Errors)**: Measures the variability within clusters.
- **Hierarchical Clustering**: Builds a hierarchy of clusters.
  - **Key Functions**: `AgglomerativeClustering` from `sklearn.cluster`

    - **Models**: `AgglomerativeClustering()`
    - **Agglomerative**: Merges clusters iteratively.
    - **Divisive**: Splits clusters iteratively.
- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**: Clusters based on the density of points.
    - **Key Functions**: `DBSCAN` from `sklearn.cluster`
    - **Models**: `DBSCAN()`
    - **Density**: The number of points within a specified radius.
    - **Points: Border, Noise, Core**: Different types of points in DBSCAN clustering.
    - **Euclidean Distance**: `sqrt(sum((x_i - y_i)^2))`

Cluster Evaluation

- **Cohesion**: Measures how closely related the items in a cluster are.
    - **Key Functions**: Not directly implemented; use custom functions.
- **Separation**: Measures how distinct a cluster is from other clusters.
    - **Key Functions**: Not directly implemented; use custom functions.
- **Silhouette Coefficient**: Measures the quality of a clustering.
    - **Formula**: `Silhouette = (b - a) / max(a, b)`, where `a` is the average intra-cluster distance and `b` is the average nearest-cluster distance.
    - **Key Functions**: `silhouette_score` from `sklearn.metrics`

Dimensionality Reduction

- **PCA (Principal Component Analysis)**: A method to reduce the dimensionality of data by transforming to a new set of variables (principal components) that are uncorrelated.
    - **Key Functions**: `PCA` from `sklearn.decomposition`
    - **Models**: `PCA(n_components=k)`
- **MDS (Multidimensional Scaling)**: A means of visualizing the level of similarity of individual cases of a dataset.
    - **Key Functions**: `MDS` from `sklearn.manifold`
    - **Models**: `MDS(n_components=k)`

Association Rule Mining

- **Frequent Itemsets**: Sets of items that appear frequently together.
    - **Key Functions**: `apriori` from `mlxtend.frequent_patterns`
- **Support Count**: The number of transactions containing a particular itemset.
    - **Key Functions**: Derived from transaction data.
- **Support**: `Support(X) = Number of transactions containing X / Total number of transactions`
    - **Key Functions**: `apriori` from `mlxtend.frequent_patterns`

- **Confidence**: `Confidence(X => Y) = Support(X union Y) / Support(X)`
    - **Key Functions**: `association_rules` from `mlxtend.frequent_patterns`
- **Apriori Principle**: If an itemset is frequent, then all of its subsets must also be frequent.
    - **Key Functions**: `apriori` from `mlxtend.frequent_patterns`

Prediction Models and Neural Networks

Neural Networks

- **Perceptron**: A simple linear classifier.
    - **Key Functions**: `Perceptron` from `sklearn.linear_model`
    - **Models**: `Perceptron()`
- **Activation Functions**: Functions that introduce non-linearity to the model.
    - **Key Functions**: `Dense(..., activation='sigmoid')`, `Dense(..., activation='relu')` in Keras
    - **Sigmoid**: `sigmoid(x) = 1 / (1 + exp(-x))`
    - **ReLU (Rectified Linear Unit)**: `ReLU(x) = max(0, x)`
- **Loss Functions**: Functions that measure the error of the model.
    - **Key Functions**: `compile(loss='mean_squared_error')`, `compile(loss='categorical_crossentropy')` in Keras
    - **Regression Loss (MSE)**: `MSE = (1/n) * sum((y_i - y_pred_i)^2)`
    - **Classification Loss (Cross-Entropy)**: `Cross-Entropy = -sum(y_i * log(y_pred_i) + (1 - y_i) * log(1 - y_pred_i))`
- **Backpropagation**: An algorithm for training neural networks through gradient descent.
    - **Key Functions**: `fit` in Keras
- **GANs (Generative Adversarial Networks)**: Networks consisting of a generator and a discriminator that compete against each other.
    - **Key Functions**: Custom implementation; use `Sequential` for both generator and discriminator in Keras.

Data Streams and Map Reduce

Data Streams

- **Sampling Data**: Selecting a subset of data for analysis.
    - **Key Functions**: `sample` from Pandas, or custom functions.
- **Sliding Windows**: A method for managing data streams by considering only the most recent data points.
    - **Key Functions**: Custom implementation; use deque from collections.
- **Filtering**: Removing unwanted elements from the data stream.
    - **Key Functions**: `filter` from Python

- **Counting Distinct Elements**: Estimating the number of distinct elements in a data stream.
    - **Key Functions**: `count` from Python collections
- **Estimating/Finding Frequent Moments**: Identifying frequent elements in the data stream.
    - **Key Functions**: Custom implementation
- **Bloom Filters**: A space-efficient probabilistic data structure for membership testing.
    - **Key Functions**: Use `bloom-filter` package in Python.
- **Flajolet-Martin Algorithm**: An algorithm for approximating the number of distinct elements in a data stream.
    - **Key Functions**: Custom implementation

Link Analysis

- **PageRank**: An algorithm for ranking web pages based on their link structure.
    - **Key Functions**: `PageRank` from `networkx`
    - **Random Walk**: A stochastic process that describes a path consisting of a succession of random steps.
    - **Key Functions**: `random_walk` from `networkx`
- **HITS (Hyperlink-Induced Topic Search)**: An algorithm that rates web pages by analyzing the link structure of the web.
    - **Key Functions**: `hits` from `networkx`
    - **Hubs and Authorities**: Two types of web pages identified by the HITS algorithm.
    - **Key Functions**: `hits` from `networkx`

Distance Measures and Similarity

Distance Metrics

- **L1 Norm (Manhattan Distance)**: `d(x, y) = sum(|x_i - y_i|)`
    - **Key Functions**: `manhattan_distances` from `sklearn.metrics.pairwise`
- **L2 Norm (Euclidean Distance)**: `d(x, y) = sqrt(sum((x_i - y_i)^2))`
    - **Key Functions**: `euclidean_distances` from `sklearn.metrics.pairwise`
- **L∞ Norm (Maximum Distance)**: `d(x, y) = max(|x_i - y_i|)`
    - **Key Functions**: Custom implementation
- **Mahalanobis Distance**: `Mahalanobis(x, y) = sqrt((x - y)^T * S^(-1) * (x - y))`, where S is the covariance matrix.
    - **Key Functions**: `mahalanobis` from `scipy.spatial.distance`

Jaccard Similarity and Distance

- **Jaccard Similarity**: `J(A, B) = |A ∩ B| / |A ∪ B|`
    - **Key Functions**: `jaccard_score` from `sklearn.metrics`

- **Jaccard Distance**: `Jaccard Distance = 1 - Jaccard Similarity`
    - **Key Functions**: Derived from `jaccard_score`

Graph Theory and Algorithms

Graph Theory

- **Adjacency Matrix**: A square matrix used to represent a finite graph.
    - **Key Functions**: `adjacency_matrix` from `networkx`
- **Adjacency List**: A collection of unordered lists used to represent a finite graph.
    - **Key Functions**: `adjacency_list` from `networkx`
- **Directed/Undirected Graphs**: Graphs where edges have a direction (directed) or do not have a direction (undirected).
    - **Key Functions**: `DiGraph` and `Graph` from `networkx`
- **Weighted/Unweighted Graphs**: Graphs where edges have weights (weighted) or do not have weights (unweighted).
    - **Key Functions**: `Graph` from `networkx` with `weight` attribute
- **Graph Walks**
    - **Random Walk**: A path consisting of a succession of random steps.
    - **Key Functions**: `random_walk` from `networkx`
    - **Shortest Path**: The minimum path length between two nodes in a graph.
    - **Key Functions**: `shortest_path` from `networkx`
- **Community Detection**: Methods to discover groups of nodes that are more densely connected internally than with the rest of the network.
    - **Key Functions**: `community` from `networkx` or `python-louvain`

Additional Concepts

Online Algorithms

- **Bipartite Matching**: Matching elements in two disjoint sets.
- **Greedy Algorithm**: An algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most immediate benefit.
- **Competitive Ratio**: A measure of the performance of an online algorithm compared to an optimal offline algorithm.
- **Web Advertising**: Methods for placing ads on the web.
    - **Ad Words**: Keywords that trigger ads.
    - **Complications (Budget, CTR)**: Issues in online advertising such as budget constraints and click-through rates.

Recommendation Systems

- **Content-Based Recommendation**: Recommending items based on user preferences for item features.

    - **TF-IDF**: Term Frequency-Inverse Document Frequency, a numerical statistic that reflects the importance of a word in a document.
- **Collaborative Filtering**: Recommending items based on the preferences of similar users.
    - **User-Based Collaborative Filtering**: Recommendations based on the preferences of similar users.
    - **Item-Based Collaborative Filtering**: Recommendations based on the similarity of items.
    - **Pearson Correlation**: A measure of the linear correlation between two variables.
    - **Cosine Similarity**: A measure of similarity between two non-zero vectors.

Graph Theory Continued

- **Node Degree and Reach**: Measures of a node's connectivity in a graph.
- **Node Proximity**: Measures of how close nodes are to each other.
- **Betweenness Centrality**: A measure of centrality in a graph based on shortest paths.
- **Closeness Centrality**: A measure of centrality in a graph based on the length of the shortest paths.

Advanced Neural Network Concepts

- **Feedforward Networks**: A type of neural network where connections between the nodes do not form a cycle.
- **Convolutional Neural Networks (CNNs)**: A class of deep neural networks, most commonly applied to analyzing visual imagery.
    - **Key Functions**: `Conv2D`, `MaxPooling2D`, `Flatten`, `Dense` from Keras
- **Recurrent Neural Networks (RNNs)**: A class of neural networks where connections between nodes form a directed graph along a sequence.
    - **Key Functions**: `SimpleRNN`, `LSTM`, `GRU` from Keras
- **GANs (Generative Adversarial Networks)**: Networks consisting of a generator and a discriminator that compete against each other.
    - **Key Functions**: Custom implementation; use `Sequential` for both generator and discriminator in Keras.
- **Autoencoders**: A type of artificial neural network used to learn efficient representations of data.
    - **Key Functions**: `Dense`, `Input` from Keras

Optimization Algorithms

- **Gradient Descent**: An optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent.
    - **Stochastic Gradient Descent (SGD)**: A variant of gradient descent where the gradient is estimated using a single sample.

    - **Mini-Batch Gradient Descent**: A variant of gradient descent where the gradient is estimated using a small batch of samples.
    - **Batch Gradient Descent**: A variant of gradient descent where the gradient is estimated using the entire dataset.
- **Adam**: An optimization algorithm that can handle sparse gradients on noisy problems.
    - **Key Functions**: `Adam` from Keras

Diffusion Models

- **Noise/Denoise**: Techniques for adding and removing noise from data.
    - **Key Functions**: Custom implementations
- **Reinforcement Learning**: A type of machine learning where an agent learns to make decisions by performing actions and receiving rewards.
    - **Key Functions**: Custom implementations
- **Decision Processes**: Models that describe the decision-making process of an agent.
- **Reward/Penalty**: The feedback an agent receives in reinforcement learning.

Association Rule Mining (Continued)

- **FP-Growth**: A method for mining frequent itemsets without candidate generation.
    - **Key Functions**: `fpgrowth` from `mlxtend.frequent_patterns`
- **FP Tree**: A compact representation of the database that provides the essential information for mining frequent patterns.
- **Prefix Path**: A path in an FP-tree that shares a common prefix.

Neural Networks and Deep Learning Models

Special Attributes and Techniques

- **Weights Initialization**: Proper initialization of weights can significantly affect the training process.
    - **Key Functions**: `kernel_initializer`, `bias_initializer` in Keras layers.
- **Learning Rate**: The step size at each iteration while moving toward a minimum of the loss function.
    - **Key Functions**: `learning_rate` parameter in optimizers like `Adam`, `SGD` in Keras.
- **Dropout**: A technique to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time.
    - **Key Functions**: `Dropout(rate)` in Keras.
- **Batch Normalization**: Normalizes the input of each layer to improve training speed and stability.
    - **Key Functions**: `BatchNormalization()` in Keras.
- **Early Stopping**: Stops training when a monitored quantity has stopped improving.

- **Key Functions**: `EarlyStopping(monitor='val_loss', patience=5)` in Keras.
- **Callbacks**: Utilities to customize the behavior of a model during training.
  - **Key Functions**: `callbacks` parameter in `model.fit` in Keras.

Gradient Descent Variants

- **SGD (Stochastic Gradient Descent)**: Performs parameter updates for each training example.
  - **Key Functions**: `SGD()` from Keras.
- **Momentum**: Accelerates SGD by adding a fraction of the previous update vector to the current update vector.
  - **Key Functions**: `SGD(momentum=0.9)` from Keras.
- **RMSprop**: Adapts the learning rate for each parameter by dividing the learning rate by an exponentially decaying average of squared gradients.
  - **Key Functions**: `RMSprop()` from Keras.
- **Adam**: Combines the advantages of two other extensions of stochastic gradient descent, AdaGrad and RMSProp.
  - **Key Functions**: `Adam(learning_rate=0.001)` from Keras.

Ensemble Methods

- **Random Forest Specifics**:
  - **n_estimators**: Number of trees in the forest.
  - **max_features**: The number of features to consider when looking for the best split.
  - **Key Functions**: `RandomForestClassifier(n_estimators=100, max_features='sqrt')` from scikit-learn.
- **Boosting Specifics**:
  - **learning_rate**: Shrinks the contribution of each tree.
  - **n_estimators**: Number of boosting stages to be run.
  - **Key Functions**: `GradientBoostingClassifier(n_estimators=100, learning_rate=0.1)` from scikit-learn.

Clustering

- **K-means Specifics**:
  - **n_clusters**: The number of clusters to form.
  - **init**: Method for initialization (`'k-means++'` is a popular choice).
  - **max_iter**: Maximum number of iterations of the k-means algorithm for a single run.

- Key Functions: `KMeans(n_clusters=8, init='k-means++', max_iter=300)` from scikit-learn.

- **DBSCAN Specifics**:
  - **eps**: The maximum distance between two samples for one to be considered as in the neighborhood of the other.
  - **min_samples**: The number of samples (or total weight) in a neighborhood for a point to be considered as a core point.
  - **Key Functions**: `DBSCAN(eps=0.5, min_samples=5)` from scikit-learn.

Evaluation Metrics

- **Confusion Matrix**: Summarizes the performance of a classification algorithm.
  - **Key Functions**: `confusion_matrix(y_true, y_pred)` from scikit-learn.
- **Precision-Recall Curve**: Shows the trade-off between precision and recall for different threshold settings.
  - **Key Functions**: `precision_recall_curve(y_true, probas_pred)` from scikit-learn.
- **F1 Score**: Combines precision and recall into a single metric.
  - **Key Functions**: `f1_score(y_true, y_pred)` from scikit-learn.

Regularization

- **L1 and L2 Regularization**: Techniques to prevent overfitting by adding a penalty to the loss function.
  - **L1 Regularization**: Adds the absolute value of the magnitude of coefficients as penalty term.
  - **L2 Regularization**: Adds the squared magnitude of coefficients as penalty term.
  - **Key Functions**: `kernel_regularizer=regularizers.l2(0.01)` in Keras layers.

Data Preprocessing

- **Standardization and Normalization**: Techniques to rescale feature values.
  - **StandardScaler**: Standardize features by removing the mean and scaling to unit variance.
  - **MinMaxScaler**: Transforms features by scaling each feature to a given range.
  - **Key Functions**: `StandardScaler()`, `MinMaxScaler()` from scikit-learn.

Advanced Neural Network Concepts

- **Transfer Learning**: Using pre-trained models on a new task.
  - **Key Functions**: Models like `VGG16`, `ResNet50` from `tensorflow.keras.applications`

- **Hyperparameter Tuning**: Adjusting the hyperparameters of a model to optimize performance.
    - **GridSearchCV**: Exhaustive search over specified parameter values for an estimator.
    - **RandomizedSearchCV**: Random search on hyperparameters.
    - **Key Functions**: `GridSearchCV`, `RandomizedSearchCV` from scikit-learn.