Evaluation, Lab Part 1a

Evaluation

Carry out an evaluation of the systems, do at least the following:

- Quantitative evaluation: Evaluate your system based on one or more evaluation metrics. Choose and motivate which metrics you use.
- Error analysis: Are there specific dialog acts that are more difficult to classify? Are there particular utterances that are hard to classify (for all systems)? And why?
- Difficult cases: Come up with two types of 'difficult instances', for example utterances that are not fluent (e.g. due to speech recognition issues) or the presence of negation (I don't want an expensive restaurant). For each case, create test instances and evaluate how your systems perform on these cases.
- System comparison: How do the systems compare against the baselines, and against each other? Which one would you choose for your dialog system?

Deliverables

- Python code that implements a majority class baseline and a keyword matching baseline
- Python code that implements two or more machine learning classifiers

In evaluating our current models, we printed plots for each. They are as follows:

- Majority classifier
- Rule-based
- Decision tree
- Feed forward neural network
- Stochastic gradient descent

Quantitative evaluation and Error analysis:

First we plotted for accuracy, but those results proved not to be too informative. For the majority classifier, for example, the "false positives" had too high an impact on the accuracy score. As there are no false positives for all classes but the majority (as everything is assigned 'inform'), the accuracy is very high.

Recall and precision are better suited. However, some of the labels that occur rarely might not be present in the test set (such as 'reqmore'), which means the recall score is unrealistically high.

It seems that sentences that are classes bye and thank you were problematic for classifiers, and also sentences that had only one word (for example christmas), this might be because bye and thank you are often both in a bye classified sentence, and that one word is too little data. As stated above reqmore was also hard to classify correctly because it does not occur often.

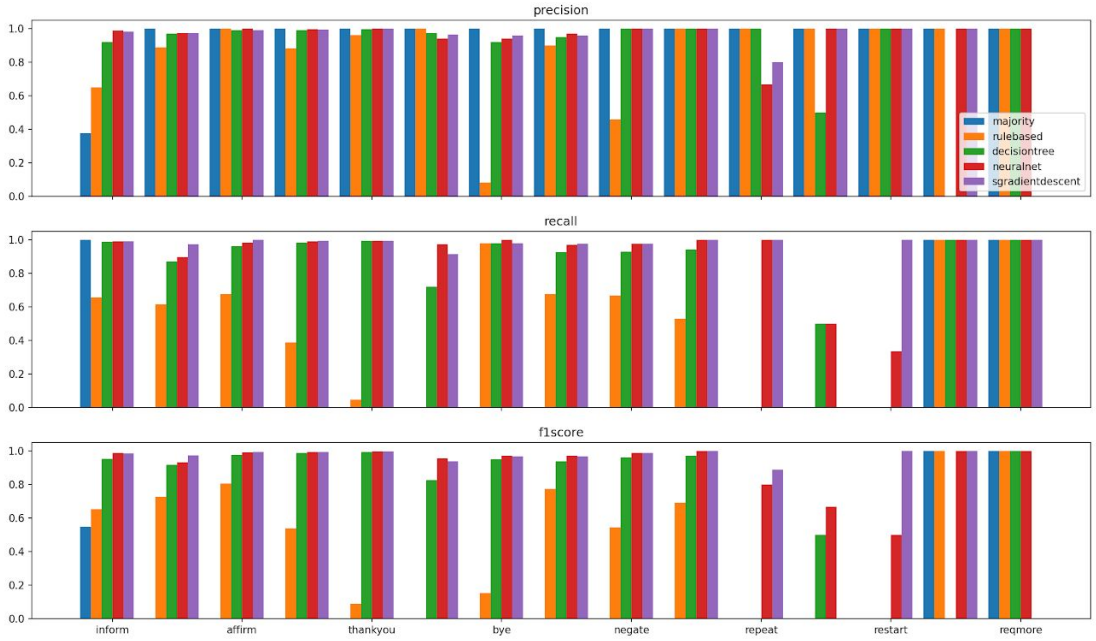
The f1 score seems - for the moment - to be the most adept at assessing the model's current performance. Though SGD has a higher score, the ffnw is the only model which is able to

achieve acceptable scores for every category, that is, there is no one label which has an f1 score of 0.

Difficult cases and system comparison:

Sentance/classifier	Majority	Rule based	Decision tree	Ff neural network	Stoch. Grad. descent
I dont want you to tell me about the chinese restaurant - Not a request	inform	inform	inform	Reqalts (it could be considered as a request to NOT do something though)	inform
Tahnk you - Misspelled thankyou	inform	Could not classify	inform	Thankyou	Thankyou
Bey - Misspelled bye	inform	Could not classify	inform	Null (it has no meaning to it)	inform
Could u pls tel me about the chinese restrnt - Bad spelling on req	inform	inform	reqalts	reqalts	reqalts
Mroe - Bad spelling on req	inform	Could not classify	inform	Null (it has no meaning to it)	inform
Phonenumber - Bad spelling on req	inform	Could not classify	inform	Null	inform
Adres - Bad spelling on req	inform	Could not classify	inform	Null (it has no meaning to it)	inform

The in-depth system comparison is below, but to sum up the difficult cases, it seems that no system can deal with one word sentences that are spelled badly, but when it comes to longer sentences with bad spelling, or maybe a few words, then the feed forward neural network, stochastic gradient descent and sometimes even the decision tree predict the right classes.



	Majority	**Rule-based**	**Decision Tree**	**SGD**	**Feedforward Neural Network**
General information	A majority classifier is rather simple. It just identifies the class most used and then assigns that label to every input.	Rules are manually defined for label assigning. An “if-then” sort of system.	Model that consists of the root node, internal nodes, branches and leaves. It has a recursive nature, i.e. the same strategy is used in the root node and all its internal nodes. It makes splits between nodes based on the split that results in the lowest costs (calculated by means of gini-index/ entropy, or other means). The leaves represent the classifications the decision tree made. Basically, it trains on the data by computing the best splits, and then with	This estimator implements regularized linear models with stochastic gradient descent (SGD). The gradient (slope of a straight line obtained by derivatives) of the loss (used to determine how well the initial line fits the data) is estimated each sample (by moving further along the line towards the negative direction of the gradient) at a time and the model is updated along the way with a learning rate. That means that the learning rate determines how far	Also known as multilayer perceptron. From an input x, a formula is approximated which should lead to correct output y, through training on the data. This is done through multiple layers, which incorporate non-linearity, known as hidden layers. The flow of data is forward, there is no looping through and within layers. Appropriate for supervised learning.

			the input it starts at the root node and then at every node the path for the input is decided, until it reaches a leaf-node. Then the input is labeled as the class that accompanies the leaf-node (based on training-data)	along the line is moved. SGD differs from gradient descent in that it replaces the actual gradient descent with an estimate thereof (calculated by randomly selected subsets of the data)	
Our implementation	Split the first word from the sentence (the label), identify the most occurring one with 'max()'. This is 'inform'. Assign this to every input.	Looking at the dataset, we try to identify common denominators for rules. Such as, if the sentence contains 'goodbye', we assign the 'bye' label.	With our implementation the splitter is set at: "best", that means every time the best split is chosen (the least amount of costs). To compute the costs we use entropy as our measure to compute the costs per split. Max_depth is set at 30, that means that the longest path from the root node to a leaf can be a maximum of 30.	In our model we use the *modified_huber* loss function. This is a modification of the huber loss function to be able to be used for classification. The huber loss function, & in our case the modified_huber, is a loss function that is less sensitive to outliers in the data when comparing it with other loss functions. L2 is used for penalties, since the other penalty functions are used for sparse solutions (uses less features in the model). Since, the difficulty with SGD is that it does not correctly classify minority classes the other penalty functions only move further away from correct classifications. Max_iter is still randomly chosen for now, but this is how many times it iterates with new data points. If we do more iterations we tend to have more classification criteria, but it might	We used the scikit implementation. As the program does not "understand" regular words, we use a vectoriser to convert the sentences to readable training data with accompanying labels. The program is trained on this vectorised set. For the solver we tried a few, Adam turned out the best results. We have yet to fine tune hyperparameters.

				at the same time lead to overfitting.	
Advantages	Very easy to implement, reasonable accuracy for such a simple algorithm,	Precise, if the rules are well defined it performs nicely. Can add precision to machine learning methods.	The interpretation of decision trees is easy. Cost to use the tree is low.	Easy to implement and efficiency	Incorporates non-linearity and is therefore more flexible and better at generalisation. It was the only model we used thus far that did not have any f1 scores of zero for any label. It can learn in real time.
Disadvantages	Unsophisticated, rigid, gets everything wrong that is not in the majority class, not well suited to new data if the statistics differ.	Very rigid. Does not do well with new data for which no specific rules are defined. Manually laborious to define rules.	For us the biggest disadvantage is the trade-off between on the one hand complexity of the mode (which leads to overfitting) which increases when the tree is bigger (due to more splits). On the other hand the smaller the model (for us regulated by the stopping-rule(decreases amount of splits): max_depth), the worse it is at classifying minority classes (e.g. reqmore, ack, deny).	Requires a lot of hyperparameters. Also, since only a subset is chosen from regular gradient descent, SGD might not learn minority classes (since these might not be selected in the subset)	Sensitivity to feature scaling, different random weight initialisations can lead to variations in validation accuracy, bit trickier to implement as it requires hyperparameter tuning (the number of iterations, the number of neurons [input aspects], the number of layers etc.), needs labelled data to train.