Theory

1. What is the difference between a rule-based system and a machine learning system?

   A rule-based system is a method of setting rules artificially to get results for several specific application scenarios while a machine learning system is the method of learning rules to predict results from training data by algorithm models automatically.

2. What is the difference between unsupervised and supervised learning?

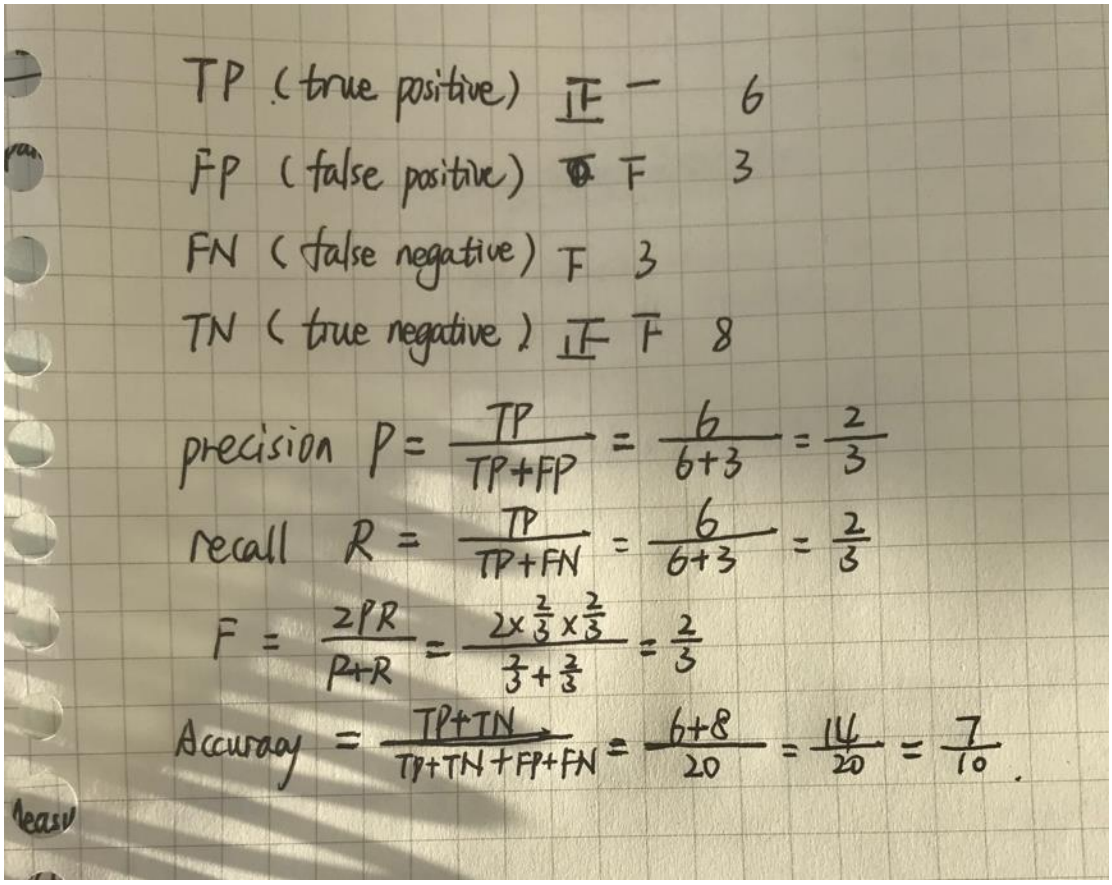
   Supervised learning needs pairs of features and targets of the training dataset to train the model and predict results for the testing dataset. Unsupervised learning does not have a training dataset but only one set of data, looking for rules., whose goal is not to generate a classification system, but to make decisions. The agent can motivate the correct behaviour and punish the wrong behaviour.

3. What do we mean when we say that a machine learning system is overfitting？

   Overfitting means almost all the features of the training data were learned, including local features and false features due to noise. Overfitting results in low "generalization" and the accuracy of the testing data.

Practice

1. Your algorithm gets the following results in a classification experiment. Please compute the precision, recall, f-measure and accuracy *manually* (without the help of your computer/Python, please provide all steps and formulas). Include the process to get to the final result.



2. You are given a dataset (named Wine data set) with different measured properties of different wines (dataset available in Learning Central). Your goal is to develop a machine learning model to predict the quality of an unseen wine given these properties. Train

two machine learning regression models and check their performance. Write, for each of the models, the main Python instructions to train and predict the labels (one line each, no need to include any data preprocessing) and the performance in the test set in terms of Root Mean Squared Error (RMSE)

Models:

```
lin_reg1 = SVR(kernel="linear", C=3, epsilon=1.5,gamma="auto")
lin_reg2=LinearRegression()
lin_reg3= Pipeline([
                    ("poly_features", PolynomialFeatures(degree=3,interaction_only=True)),
                    ("std_scaler", StandardScaler()),
                    ("regul_reg", lin_reg2),
                ])
plot_learning_curves(lin_reg1)
plot_learning_curves(lin_reg2)
plot_learning_curves(lin_reg3)
```

Train regression models and check performance using in terms of RMSE:

```
model.fit(X_train, y_train) # Train using different models
y_test_predict = model.predict(X_val) # predict results
Testscore=model.score(X_val,y_val) # compute score of model
TestMSE=mean_squared_error(y_val, y_test_predict) #compute MSE
print("test score:"+str(Testscore)) #print score of model
print("test RMSE:"+str(cmath.sqrt(TestMSE))) #print RMSE
```

	Score	RMSE
SVR with linear kernel	0.21	0.789
Linear Regression	0.24	0.774
Linear Regression After using Standardscaler() and PolynomialFeatures()	0.32	0.73

3. Train an SVM binary classifier using the Hateval dataset (available in Learning Central). The task consists of predicting whether a tweet represents hate speech or not. You can preprocess and choose the features freely. Evaluate the performance of your classifier in terms of accuracy using ten fold-cross validation. Write a table with the results of the classifier (accuracy, precision, recall and F-measure) in each of the folds and write a small summary (up

to 500 words) of how you preprocessed the data, chose the feature/s, and trained and evaluated your model.

Table 1. The performance of the classifier in 10 fold cross-validation

Average Accuracy: 0.767

	precision	recall	f1	accuracy
1	0.759032	0.760441	0.759699	0.768889
2	0.746116	0.750333	0.747062	0.750000
3	0.736743	0.735590	0.736142	0.747778
4	0.752314	0.756914	0.753437	0.756667
5	0.753298	0.758570	0.754744	0.758889
6	0.769067	0.768011	0.768503	0.773333
7	0.756119	0.751786	0.753339	0.758889
8	0.734555	0.728520	0.730232	0.736667
9	0.731099	0.734003	0.732331	0.741111
10	0.761527	0.762203	0.761850	0.766667

Preprocessing

Lemmatization

```
01. lemmatizer = nltk.stem.WordNetLemmatizer()
```

Removing stop words

```
01. stopwords=set(nltk.corpus.stopwords.words('english'))
02. stopwords.add(".")
03. stopwords.add(",")
04. stopwords.add("!")
05. stopwords.add(":")
06. stopwords.add("?")
```

Choice of features

Bag of words

```
# Function to get the list of tokens from a string
def get_list_tokens(string):
    sentence_split=nltk.tokenize.sent_tokenize(string)
    list_tokens=[]
    for sentence in sentence_split:
        list_tokens_sentence=nltk.tokenize.word_tokenize(sentence)
        for token in list_tokens_sentence:
            list_tokens.append(lemmatizer.lemmatize(token).lower())
    return list_tokens

# Function to get feature vectors
def get_vector_text(list_vocab,string):
    vector_text=np.zeros(len(list_vocab))
    list_tokens_string=get_list_tokens(string) ##Use function
    for i, word in enumerate(list_vocab):
        if word in list_tokens_string:
        #print (list_tokens_string.count(word))
            vector_text[i]=list_tokens_string.count(word) #Count words
    return vector_text


# Function to retrieve features based on bag of words
def get_vocabulary(training_set, num_features):
    vocabulary_inner=[]
    dict_word_frequency={}
    for instance in training_set:
        sentence_tokens=get_list_tokens(instance[0])##Use function
        for word in sentence_tokens:
            if word in stopwords: continue
            if word not in dict_word_frequency: dict_word_frequency[word]=1
            else: dict_word_frequency[word]+=1
    sorted_list = sorted(dict_word_frequency.items(), key=operator.itemgetter(1), reverse=True)[:num_features]
    for word,frequency in sorted_list:
        vocabulary_inner.append(word)
    return vocabulary_inner

X_train=[]
Y_train=[]
for instance in training_set:
    vector_instance=get_vector_text(vocabulary,instance[0])
    X_train.append(vector_instance)
    Y_train.append(instance[1])
```

Each token occurrence frequency is treated as a feature, then a collection of text documents is transformed into numerical feature vectors as input data.

Train the model

```
def train_svm_classifier(training_set, vocabulary):
    X_train=[]
    Y_train=[]
    for instance in training_set:
        vector_instance=get_vector_text(vocabulary,instance[0])
        X_train.append(vector_instance)
        Y_train.append(instance[1])
    #test_svm_function(X_train,Y_train) # Parameters Adjustment
    scaler = StandardScaler()
    #print(scaler)
    X_scaled = scaler.fit_transform(X_train) #StandScaler Training dataset
    #print(X_scaled)
    #test_svm_function(X_train,Y_train)
    #svm_clf=sklearn.svm.SVC(kernel="rbf",gamma=3.59) #train:99.7%, test:55.8%
    svm_clf=sklearn.svm.SVC(kernel="linear",C=1,gamma='auto') #train:85% test:76%
    #svm_clf=sklearn.svm.SVC(kernel="poly",degree=2,gamma='auto')#train:59% test 58%
    svm_clf.fit(X_scaled,Y_train)
    print("Train data score:"+str(svm_clf.score(X_scaled,Y_train)))

    return svm_clf
```

SVC models with linear, "RBF" and "poly" kernel provided by SVM are used. Also, methods of the learning curve and grid search are used to find the best parameters for each model. Function, as follows, are carried out for parameter adjustment.

```
#learning curve
def test_svm_function(X_train,Y_train):
    gamma_range=np.logspace(0,1,10)
    score=[]
    for i in gamma_range:
        svm_clf=SVC(kernel="rbf",gamma=i,cache_size=5000).fit(X_train,Y_train)
        score.append(svm_clf.score(X_train,Y_train))
    print(max(score),gamma_range[score.index(max(score))])
    plt.plot(gamma_range,score)
    return gamma_range[score.index(max(score))]

#grid search
def test_svm_function2(X_train,Y_train):
    gamma_range=np.logspace(-10,1,20)
    coef0_range=np.linspace(0,5,10)

    param_grid=dict(gamma=gamma_range,coef0=coef0_range)

    cv=StratifiedShuffleSplit(n_splits=10,test_size=0.3,random_state=420)
    grid=GridSearchCV(SVC(kernel="rbf"),param_grid=param_grid,cv=cv)
    grid.fit(X_train,Y_train)
    print("The best parameters are %s with a score of %0.5f"%(grid.best_params_,grid.best_score_))
```

Finally, SVC with a linear kernel is the best model among them and kept in codes.

Evaluation using 10-fold cross-validation

```
def Kfold():
    table=pd.DataFrame(columns=['precision','recall','f1','accuracy'])
    kf = KFold(n_splits=10)
    print (kf)
    random.shuffle(dataset_full)
    kf.get_n_splits(dataset_full)
    index=0
    for train_index, test_index in kf.split(dataset_full):
        train_set_fold=[]
        test_set_fold=[]
        accuracy_total=0.0
        index+=1
        for i,instance in enumerate(dataset_full):
            if i in train_index:
                train_set_fold.append(instance)
            else:
                test_set_fold.append(instance)

        vocabulary_fold=get_vocabulary(train_set_fold, 1000)
        svm_clf_fold=train_svm_classifier(train_set_fold, vocabulary_fold)
        X_test_fold=[]
        Y_test_fold=[]
        for instance in test_set_fold:
            vector_instance=get_vector_text(vocabulary_fold,instance[0])
            X_test_fold.append(vector_instance)
            Y_test_fold.append(instance[1])
        Y_test_fold_gold=np.asarray(Y_test_fold)
        X_test_fold=np.asarray(X_test_fold)
        Y_test_predictions_fold=svm_clf_fold.predict(X_test_fold)
        print ("Fold completed.")
        ### compute different value ###
        print("Test data score:"+str(svm_clf_fold.score(X_test_fold,Y_test_fold_gold)))
        accuracy_fold=accuracy_score(Y_test_fold_gold, Y_test_predictions_fold)
        accuracy_total+=accuracy_fold
        precision_fold=precision_score(Y_test_fold_gold, Y_test_predictions_fold, average='macro')
        recall_fold=recall_score(Y_test_fold_gold, Y_test_predictions_fold, average='macro')
        f1_fold=f1_score(Y_test_fold_gold, Y_test_predictions_fold, average='macro')
        ### draw the graph ###
        table=table.append(pd.DataFrame({'precision':precision_fold,'recall':recall_fold,'f1':f1_fold,'accuracy':accuracy_fold},index=[index]))
    average_accuracy=accuracy_total/10
    print ("\nAverage Accuracy: "+str(round(accuracy_fold,3)))
    print (table)
Kfold()
```

Firstly, the dataset is split into ten pairs of training dataset and testing dataset. Then vectorize these datasets using the same features based on word frequency. Finally, the model is evaluated by accuracy, precision, recall and f1 scores. Then average accuracy is computed by ten folds.

Table 1 shows the performance of the model and average accuracy: 0.767