COMP9417 - Machine Learning
Homework 2: Perceptrons, Kernels & `scikit-learn`

Introduction In this homework, we will explore perceptron learning in more depth. We will then dive further into some existing sklearn modules and compare their perforamance on simulated data.

Question 1. Diving Deeper into sklearn

In this question, we will consider the scikitlearn implementations of the following classifiers:

- Decision Trees
- Random Forest
- AdaBoost
- Logistic Regression
- Multilayer Perceptron (Neural Network)
- Support Vector Machine

You are required to compare the performance of the above models on a classifiation task. The following code loads in these classifiers and defines a function to simulate a toy dataset:

```
        import numpy as np
        import matplotlib.pyplot as plt
        from matplotlib.colors import ListedColormap
        import warnings
        warnings.simplefilter(action='ignore', category=FutureWarning)

        import time
        from sklearn.svm import SVC
        from sklearn.linear_model import LogisticRegression
        from sklearn.ensemble import AdaBoostClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.neural_network import MLPClassifier
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.datasets import make_classification

        def create_dataset(n=1250, nf=2, nr=0, ni=2, random_state=125):
            '''
            generate a new dataset with
            n: total number of samples
            nf: number of features
            nr: number of redundant features (these are linear combinatins of informative
    features)
            ni: number of informative features (ni + nr = nf must hold)
            random_state: set for reproducibility
```

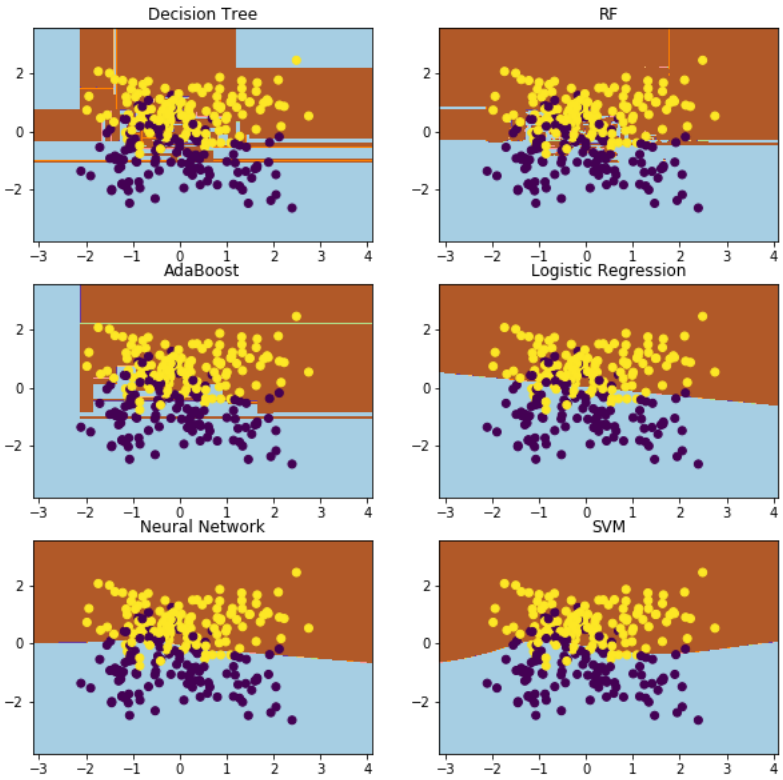
```
        '''
        X, y = make_classification( n_samples=n,
                                    n_features=nf,
                                    n_redundant=nr,
                                    n_informative=ni,
                                    random_state=random_state,
                                    n_clusters_per_class=2)
        rng = np.random.RandomState(2)
        X += 3 * rng.uniform(size = X.shape)
        X = StandardScaler().fit_transform(X)
        return X, y


```

(a) Generate a dataset using the default parameters of `create_dataset`. Then, randomly split the dataset into training set Xtrain and test set Xtest (use the sklearn `train_test_split` function with `random_state=45`), with 80 % of examples for training and 20 % for testing. Fit each of the models to the training data and then plot the decision boundaries of each of the classifiers (using default parameter settings) on the test set. If you prefer, you may use the following plotter function which plots the decision boundary and works for any sklearn model.

```
    def plotter(classifier, X, X_test, y_test, title, ax=None):
        # plot decision boundary for given classifier
        plot_step = 0.02
        x_min, x_max = X[:, 0].min() - 1, X[:,0].max() + 1
        y_min, y_max = X[:, 1].min() - 1, X[:,1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                             np.arange(y_min, y_max, plot_step))
        Z = classifier.predict(np.c_[xx.ravel(),yy.ravel()])
        Z = Z.reshape(xx.shape)
        if ax:
            ax.contourf(xx, yy, Z, cmap = plt.cm.Paired)
            ax.scatter(X_test[:, 0], X_test[:, 1], c = y_test)
            ax.set_title(title)
        else:
            plt.contourf(xx, yy, Z, cmap = plt.cm.Paired)
            plt.scatter(X_test[:, 0], X_test[:, 1], c = y_test)
            plt.title(title)

```

Note that you can use the same general approach for plotting a grid as used in Homework 1, and the plotter function supports an 'ax' argument. *What to submit: a single* 3×2 *plot, a print screen of the python code used, a copy of your python code in solutions.py.*

Solution:
The students should generate a plot that looks like:



. The code used to generate this plot is:

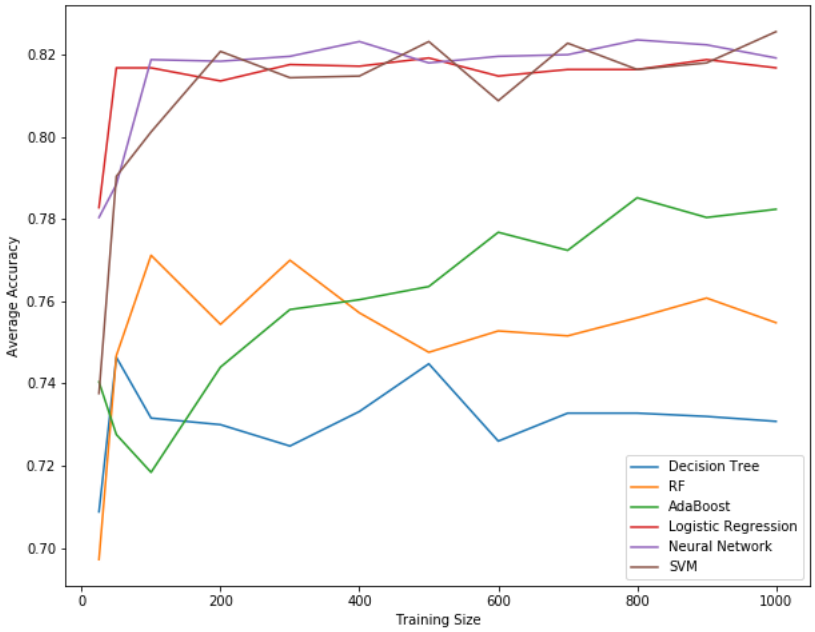
```
        X, y = create_dataset()
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=45)

        mods = [DecisionTreeClassifier(),
        RandomForestClassifier(),
        AdaBoostClassifier(),
        LogisticRegression(),
        MLPClassifier(),
        SVC()]

        titles = ["Decision Tree", "RF", "AdaBoost",
                "Logistic Regression", "Neural Network", "SVM"]

        fig, ax = plt.subplots(3,2, figsize=(10,10))
        for i, ax in enumerate(ax.flat):
            mod = mods[i]
```

```
            mod.fit(X_train, y_train)
            plotter(mod, X, X_test, y_test, titles[i], ax)
        plt.savefig("figures/boundaries.png")
        plt.show()

```

(b) Next, we will study how the performance of each of the clasifiers varies as you increase the size of the training set. Fix your training and test sets from part (a). Then, starting from a random subset (no need to set a seed) of your training set of size 50 (chosen with replacement), train your classification model, and compute the accuracy on the test set. Repeat this process for training set sizes of $[50, 100, 200, 300, \ldots, 1000]$. Repeat the experiment a total of 10 times for each classifier. Then, for each classifier, plot its average accuracy (achieved on the test set) for each training set size. Compare the accuracy across different algorithms in a **single** figure, and in 5 lines or less, discuss your observations: For the models covered in the course so far, use what you know about the bias-variance decomposition to inform your discussion. Which model you prefer for this task?

Please use the following color scheme for your plots: [Decision Tree, Random Forest, AdaBoost, Logistic Regression, Neural Network, SVM], and please include a legend in your plot. *What to submit: a discussion of your observation, a single plot, a print screen of the python code used, a copy of your python code in solutions.py.*

Solution:
The figure generated is



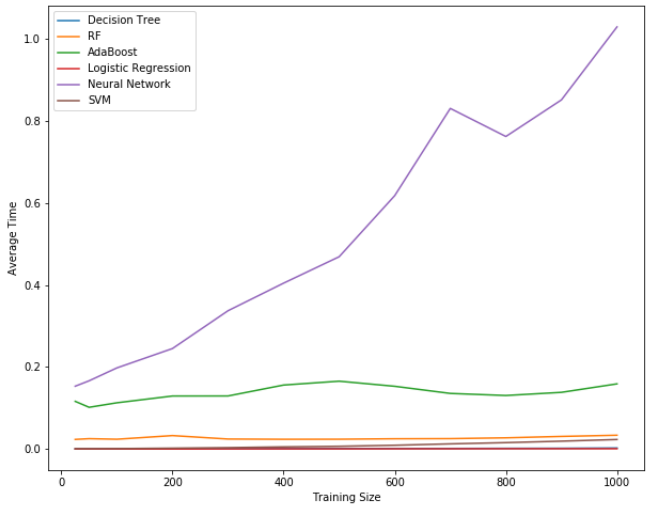
Discussion points can include:

- trees doing poorly and exhibiting high variance, RF seems to be overfitting
- as expected MLPs and SVMs do quite well without much tuning
- logistic regression is preferred here as it the simplest model and achieves consistently good results.

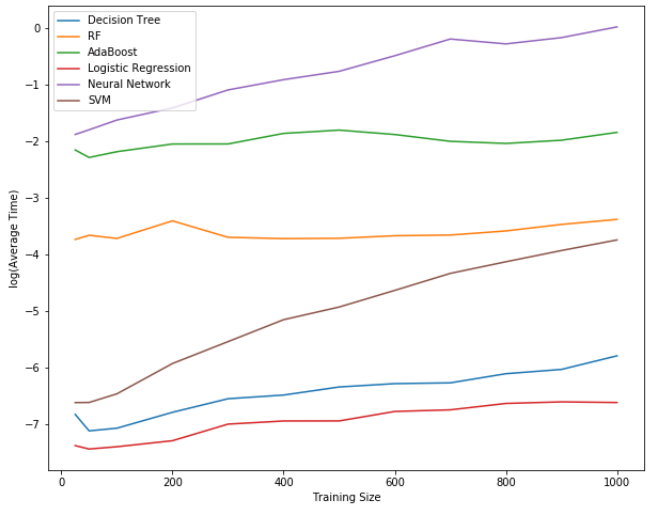
Full marks should use the decision surfaces in (a) to argue about overfitting of RFs and Adaboost relative to the other models. Code is presented at end of part (c).

(c) Using the time module, record the training time for each of the classifiers at each of the training set sizes. Plot the log of the average training time over the 10 trials of each classifier as a function of the training size (use log base e). You may add code for this section to your code in part (b). What do you observe? In 5 lines or less, discuss your observations. Use the same color scheme as in (b). *What to submit: a discussion of your observation, a single plot, a print screen of the python code used, a copy of your python code in solutions.py.*

Solution:
The figure generated is



of if log-scale;



Some basic comments about NNs taking more time as you increase the training size. Some discussion of the default MLP number of hidden layers would be good here but not necessary. Code used for both (b) and (c) is provided here:

```
    def train_mod(mod, X_train, y_train, X_test, y_test, train_size):
        idxs = np.random.choice(X_train.shape[0], size=train_size)
        Xp = X_train[idxs]
        yp = y_train[idxs]
        mod.fit(Xp, yp)
        return np.mean(mod.predict(X_test)==y_test)

    mod_mean_accuracies = np.zeros(shape=(6, 12))
    mod_mean_time = np.zeros(shape=(6, 12))
    train_sizes = [25,50] + [100 + 100*i for i in range(10)]
    for mod_idx, mod in enumerate(mods):
        mean_accuracies = np.zeros(shape=(10, 12))
        mean_times = np.zeros(shape=(10, 12))
        for i in range(10):
            for j, tr in enumerate(train_sizes):
                tstart = time.time()
                mean_accuracies[i, j] = train_mod(mod, X_train,
                                                  y_train, X_test,
                                                  y_test, tr)
                tend = time.time()
                mean_times[i,j] = tend - tstart
        mod_mean_accuracies[mod_idx,:] = mean_accuracies.mean(axis=0)
        mod_mean_time[mod_idx, :] = mean_times.mean(axis=0)

    fig_size = plt.rcParams["figure.figsize"]
    fig_size[0] = 10
    fig_size[1] = 8
    plt.rcParams["figure.figsize"] = fig_size

    for i in range(6):
        plt.plot(train_sizes, mod_mean_accuracies[i], label=titles[i])
        plt.xlabel("Training Size")
        plt.ylabel("Average Accuracy")
    plt.legend()
    plt.savefig("figures/Accuracy.png")
    plt.show()

    for i in range(6):
        plt.plot(train_sizes, mod_mean_time[i], label=titles[i])
        plt.xlabel("Training Size")
        plt.ylabel("Average Time")
    plt.legend()
    plt.savefig("figures/Timing.png")
    plt.show()


```

(d) We now focus on the `DecisionTreeClassifier` module. Use `create_dataset` from the previous question to generate a new dataset consisting of 2000 samples, 20 total features, 8 of which are informative, and use a random state of 25. Next, generate equally sized train/test data, using a random state of 15. Fit a deciscion tree classifier (using default parameters except for setting `random_state` to 0) and report both its train and test accuracy. *What to submit: train accuracy of your model, test accuracy of your model, a screen shot of your code, a copy of your python code in solutions.py.*

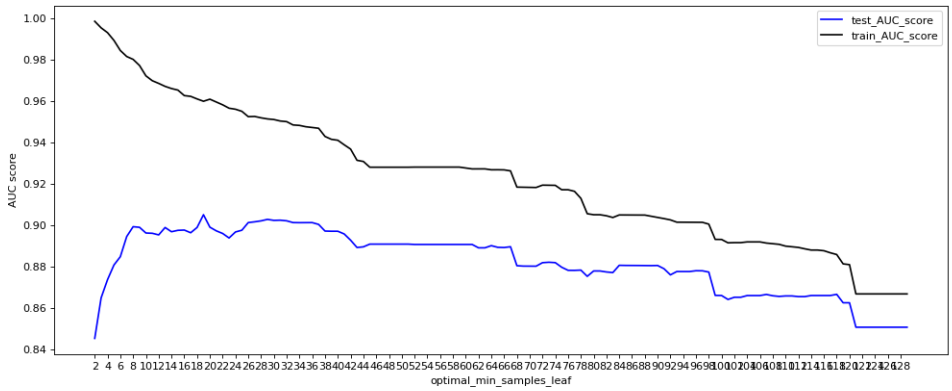
Solution:
We get:

train accuracy: 1.0, test accuracy: 0.814

```
import sklearn.metrics as metrics

X, y = create_dataset(n=2000, nf=20, nr=12, ni=8, random_state=25)

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.5,
                                                    random_state=15)

mod = DecisionTreeClassifier(random_state=0)
mod.fit(X_train, y_train)

train_accuracy = metrics.accuracy_score(y_train, mod.predict(X_train))
test_accuracy = metrics.accuracy_score(y_test, mod.predict(X_test))

print('Train Accuracy = ', train_accuracy , ', Test Accuracy = ', test_accuracy)
```

(e) Fit multiple decision trees with `min_samples_leaf` varying over $k = 2, 3, \ldots, 130$, and with `random_state` set to 0 For each iteration, compute the AUC score on both your train and test datasets from the previous part. Provide a plot of both train and test AUC over k. *What to submit: a single plot, a screen shot of your code, a copy of your python code in solutions.py.*

Solution:
We get the following plot:



The code used:

```
            training_scores = []
```

```
            testing_scores = []
            n_min = 2
            n_max = 130
            for n in range(n_min, n_max):
                mod = DecisionTreeClassifier(random_state=0,min_samples_leaf=n)
                mod.fit(X_train, y_train)
                testing_scores.append(metrics.roc_auc_score(y_test, mod.predict_proba(
    X_test)[:,1]))
                training_scores.append(metrics.roc_auc_score(y_train, mod.
    predict_proba(X_train)[:,1]))

            plt.figure(figsize=(15, 6), dpi=80, facecolor='w', edgecolor='k')
            plt.plot(range(n_min,n_max), testing_scores, label='test_AUC_score', color
    ='blue')
            plt.plot(range(n_min,n_max), training_scores, label='train_AUC_score',
    color='black')
            plt.xlabel('optimal_min_samples_leaf')
            plt.ylabel('AUC score')
            plt.xticks(np.arange(n_min, n_max, step=2))
            plt.legend()
            plt.savefig("figures/AUC_scores.png")
            plt.show()

```

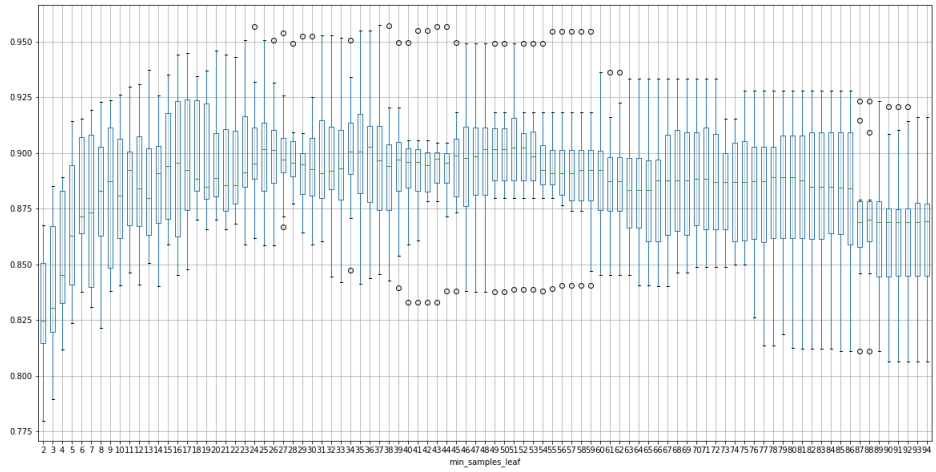
(f) In this part, you will perform cross validation over the min_samples_leaf hyperparameter from scratch (**Do not use existing cross validation implementations in this question, doing so will result in a mark of zero.**) For `min_samples_leaf` ranging from $k = 2$ to $k = 95$, run 10-fold cross validation (i.e. split the training data into 10 folds, fit a decision tree with `min_samples_leaf` set to k (and `random_state` set to 0) on 9 of those folds, and record the auc score on the 10-th, repeating this process 10 times). For this question, your 10 folds must be created by taking the first fold to be the first 100 rows, the second fold to be the next 100 rows, and so on.

Produce a plot displaying your results: the x-axis should be `min_samples_leaf`, and for each k plot a Box-plot over the 10 CV scores. Further, report the value of k that results in the highest CV score, re-fit the Decision tree (with `random_state` set to 0) with this choice and report both train and test accuracy. *What to submit: a single plot, train accuracy of your model, test accuracy of your model, a screen shot of your code, a copy of your python code in solutions.py.*

Solution:
The optimal choice of the paraeter is $k = 24$. We get:

$$\text{train accuracy: } 0.881, \qquad \text{test accuracy: } 0.825$$

We get the following plot:




```
import pandas as pd
n_min = 2
n_max = 95
K = 10

idxs = np.zeros(shape=(K,100))
for i in range(K):
    idxs[i] = np.arange(i*100, (i+1)*100)


scores = np.zeros(shape=(n_max-n_min, K))

for n in range(n_max-n_min):
    for k in range(K):

        # get idxs for train/test
        cur_test_idx = np.int32(idxs[k])
        cur_train_idx = np.int32(np.delete(idxs, k, axis=0).reshape(-1))

        # fit model
        mod = DecisionTreeClassifier(random_state=0, min_samples_leaf=n+2)
        mod.fit(X_train[cur_train_idx], y_train[cur_train_idx])

        auc_s = metrics.roc_auc_score(y_train[cur_test_idx],
                                      mod.predict_proba(X_train[cur_test_idx])[:,1])
        scores[n, k] = auc_s

df = pd.DataFrame(scores, columns = ['s1','s2','s3','s4','s5','s6', 's7', 's8', '
s9', 's10'])
df['n'] = np.arange(n_min, n_max)
tdf = df.set_index('n').T
tdf.boxplot(figsize=(20,10))
plt.xlabel('min_samples_leaf')
plt.savefig('figures/cv_scores.png')
```

```
    plt.show()

    kmax = scores.mean(axis=1).argmax()+2

    mod = DecisionTreeClassifier(min_samples_leaf=kmax).fit(X_train, y_train)
    train_accuracy = metrics.accuracy_score(y_train, mod.predict(X_train))
    test_accuracy = metrics.accuracy_score(y_test, mod.predict(X_test))

    print('Train Accuracy = ', train_accuracy , ', Test Accuracy = ', test_accuracy)
```

(g) Finally, we will use scikit-learn's built in `GridSearchCV` to repeat the analysis in the previous part. Report the optimal `min_samples_leaf` chosen by this process. Re-fit the Decision tree (with `random_state` set to 0) with this choice and report both train and test accuracy. Explain why the results might differ between your implementation and sklearn's. *What to submit: train accuracy of your model, test accuracy of your model, a screen shot of your code, a copy of your python code in solutions.py, some comments to address the results.*

Solution:

The optimal choice of the paraeter is $k = 28$. We get:

$$\text{train accuracy: } 0.878, \qquad \text{test accuracy: } 0.828$$

Assuming that your implementation is correct, the difference (which is quite small) between the two approaches is down to different cross validation splits being used, and the high variance of decision trees.

```
            from sklearn.model_selection import GridSearchCV
            param_grid = {'min_samples_leaf': np.arange(n_min, n_max, 1)}

            grid_tree = GridSearchCV(DecisionTreeClassifier(random_state=0),
    param_grid, cv=10, scoring='roc_auc')
            grid_tree.fit(X_train, y_train)

            print(grid_tree.best_params_['min_samples_leaf'])

            cv_best = grid_tree.best_params_['min_samples_leaf']

            mod = DecisionTreeClassifier(min_samples_leaf=cv_best).fit(X_train,
    y_train)
            train_accuracy = metrics.accuracy_score(y_train, mod.predict(X_train))
            test_accuracy = metrics.accuracy_score(y_test, mod.predict(X_test))

            print('Train Accuracy = ', train_accuracy , ', Test Accuracy = ',
    test_accuracy)
```

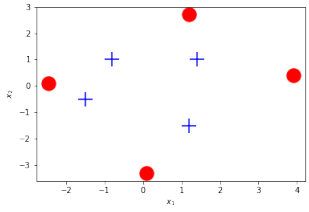
Question 2. (Perceptron Learning & Features)

This question requires you to refer to the following training data for parts (a)-(c). You are only permitted to make use of numpy and matplotlib. You are not permitted to make use of any existing numpy implementations of perceptrons (if they exist).

x_1	x_2	y
-0.8	1	1
3.9	0.4	-1
1.4	1	1
0.1	-3.3	-1
1.2	2.7	-1
-2.45	0.1	-1
-1.5	-0.5	1
1.2	-1.5	1

(a) Plot the data. Recall that the polynomial kernel is defined as $k(x, y) = (m + x^T y)^d$ for $m \in \{0, 1, 2, \dots\}$ and $d \in \{1, 2, \dots\}$. Each such kernel corresponds to a feature representation of the original data. Find the simplest polynomial kernel for which this data becomes linearly separable (**note:** simplest in this case is defined as the polynomial kernel with the smallest values of **both** m and d). *What to submit: a single plot, a value for m, a value for d.*

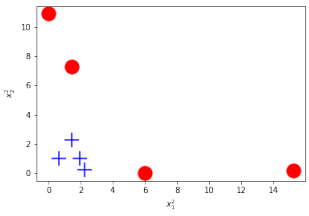
Solution:
The plot of the data is:



The simplest polynomial kernel is $m = 0$, $d = 2$, i.e. $k(x, y) = (x^T y)^2$

(b) The optimal kernel found in (a) induces a feature representation in $\mathbb{R}^p$ for some integer p determined by your choice of kernel, write down this feature representation. Then, choose a subset of two coordinates from this p-dimensional vector and plot the transformed data. Is your transformed data linearly separable? For example, for vector $(w, x, y, z)^T \in \mathbb{R}^4$, a subset of two coordinates could be the first two coordinates (w, x), or the first and third coordinates (w, y), etc.). *What to submit: a single plot, a description of the feature vector, and an answer to the question of linear separability.*

Solution:
The feature vector is $\phi([x_1, x_2]^T) = [x_1^2, x_2^2, \sqrt{2}x_1x_2]$. Plotting the first two coordinates:



(c) Train a perceptron on the transformed data (using your feature transformation in (b)) with initial weight vector $w^{(0)} = \mathbf{1}_{p+1}$ (the vector of ones in $\mathbb{R}^{p+1}$). Use a learning rate of $\eta = 0.2$. Note: this should be done in numpy. Provide a table outlining all updates of the weight vector, and the iteration number at which the update occured. State the final learned perceptron and the number of iterations until convergence. Demonstrate that your perceptron correctly classifies each of the points. You may use the following table as a template for presenting your results:

Iteration No.	w_0	w_1	$\dots$	w_p
0	1	1	$\dots$	1
first update iteration	$1+\delta_0$	$1+\delta_1$	$\dots$	$1+\delta_p$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

where δ_j is the update for w_j computed at the first iteration. To demonstrate that your perceptron classifies each point correctly, use the following table:

x_i	$\phi(x_i)$	$y_i\phi^T(x_i)w^*$
$[-0.8, 1]^T$	$\phi([-0.8, 1]^T)$	$r_1 > 0$
$\vdots$	$\vdots$	$\vdots$
$[1.2, -1.5]^T$	$\phi([1.2, -1.5]^T)$	$r_8 > 0$

where $r_i = y_i\phi^T(x_i)w^*$ should be positive if your perceptron has converged. *What to submit: the final weight vector, a table of iterations, a table showing that your model has converged, a print screen of the python code used, a copy of the python code used in your solutions.py file.*

Solution:

The final weights vector should be

$$w^{(59)} = [3, -0.62, -0.654, 0.07793276]$$

The code used is

```
import numpy as np
import matplotlib.pyplot as plt

# data
X = np.array([
    [-0.8,1],
    [3.9,0.4],
    [1.4,1],
    [0.1,-3.3],
    [1.2,2.7],
```

```
            [-2.45, 0.1],
            [-1.5,-0.5],
            [1.2,-1.5]
        ])

        y = np.array([1,-1,1,-1,-1,-1,1,1])

        plt.scatter(X[:,0][y==1], X[:,1][y==1], color="blue", marker="+", s=300)
        plt.scatter(X[:,0][y==-1], X[:,1][y==-1], color="red", marker="o", s=300)
        plt.xlabel("$x_1$")
        plt.ylabel("$x_2$")
        plt.savefig("figures/perceptron_a.png")
        plt.show()

        # transform X
        def phi(x):
            return(np.array([x[0]**2, x[1]**2, np.sqrt(2)*x[0]*x[1]]))

        phiX = np.apply_along_axis(phi, 1, X)

        plt.scatter(phiX[:,0][y==1], phiX[:,1][y==1], color="blue", marker="+", s=300)
        plt.scatter(phiX[:,0][y==-1], phiX[:,1][y==-1], color="red", marker="o", s
    =300)
        plt.xlabel("$x_1^2$")
        plt.ylabel("$x_2^2$")
        plt.savefig("figures/perceptron_b.png")
        plt.show()


        # initialise perceptron
        w = np.zeros(shape=(100, 4))
        w = np.concatenate((np.arange(100).reshape(-1,1), w), axis=1)
        w[0,1:] = np.array([1,1,1,1])
        # add bias to feature vecs
        phiX = np.concatenate((np.ones(8).reshape(-1,1), phiX), axis=1)

        idx = 0
        ctr = 10
        eta = 0.2
        conv = 0        # keep track of convergence
        iteration_no = 0
        while ctr:
            for i in range(X.shape[0]):
                iteration_no += 1
                conv+=1
                if (y[i] * w[idx, 1:] @ phiX[i] < 0):
                    conv=0     # reset convergence counter
                    w[idx+1, 0] = iteration_no
                    w[idx+1, 1:] = w[idx, 1:] + y[i] * phiX[i] * eta
                    idx +=1
            if conv>=8:
                wstar = w[idx,1:]
                break
            ctr -= 1

        # check correct
        for i in range(8):
            print(y[i] * phiX[i] @ wstar)
```

```
        # store
        np.savetxt("perceptron.csv", w[:idx+1], delimiter=',')
```

The following table shows the iterations:

0	1	2	3	4
0.00	1.00	1.00	1.00	1.00
2.00	0.80	−2.04	0.97	0.56
3.00	1.00	−1.65	1.17	0.95
4.00	0.80	−1.65	−1.01	1.05
7.00	1.00	−1.20	−0.96	1.26
8.00	1.20	−0.91	−0.51	0.75
9.00	1.40	−0.79	−0.31	0.52
13.00	1.20	−1.07	−1.77	−0.39
15.00	1.40	−0.62	−1.72	−0.18
16.00	1.60	−0.34	−1.27	−0.69
19.00	1.80	$5.6 \cdot 10^{-2}$	−1.07	−0.29
22.00	1.60	−1.14	−1.07	−0.22
23.00	1.80	−0.69	−1.02	$-1.12 \cdot 10^{-2}$
24.00	2.00	−0.41	−0.57	−0.52
27.00	2.20	$-1.45 \cdot 10^{-2}$	−0.37	−0.12
30.00	2.00	−1.22	−0.37	$-5.5 \cdot 10^{-2}$
31.00	2.20	−0.77	−0.32	0.16
32.00	2.40	−0.48	0.13	−0.35
36.00	2.20	−0.48	−2.05	−0.26
40.00	2.40	−0.19	−1.60	−0.77
43.00	2.60	0.20	−1.40	−0.37
46.00	2.40	−1.00	−1.40	−0.30
47.00	2.60	−0.55	−1.35	$-9.04 \cdot 10^{-2}$
48.00	2.80	−0.26	−0.90	−0.60
54.00	2.60	−1.46	−0.90	−0.53
55.00	2.80	−1.01	−0.85	−0.32
59.00	3.00	−0.62	−0.65	$7.79 \cdot 10^{-2}$

m=1, d=2 Some students will have incorrectly taken $m = 1$ in part (a). They can still get full marks here, their feature vector is then

$$\phi([x_1, x_2]^T) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2]$$

Note that there is no longer a need to add a bias term when training the perceptron. The final weight vector is:

$$w^{(79)} = [4.8, 0.4060303, -0.04651804, -0.465, -0.742, -0.34067446]$$

The following table shows the iterations:

0	1	2	3	4	5	6
0.00	1.00	1.00	1.00	1.00	1.00	1.00
2.00	0.80	-0.10	0.89	-2.04	0.97	0.56
3.00	1.00	0.29	1.17	-1.65	1.17	0.95
4.00	0.80	0.26	2.10	-1.65	-1.01	1.05
5.00	0.60	$-7.48 \cdot 10^{-2}$	1.34	-1.94	-2.47	0.13
7.00	0.80	-0.50	1.20	-1.49	-2.42	0.34
8.00	1.00	-0.16	0.77	-1.20	-1.97	-0.17
9.00	1.20	-0.39	1.06	-1.07	-1.77	-0.39
11.00	1.40	$1.01 \cdot 10^{-2}$	1.34	-0.68	-1.57	$4.39 \cdot 10^{-3}$
15.00	1.60	-0.41	1.20	-0.23	-1.52	0.22
16.00	1.80	$-7.48 \cdot 10^{-2}$	0.77	$5.6 \cdot 10^{-2}$	-1.07	-0.29
18.00	1.60	-1.18	0.66	-2.99	-1.10	-0.73
19.00	1.80	-0.78	0.94	-2.59	-0.90	-0.34
23.00	2.00	-1.21	0.80	-2.14	-0.85	-0.13
24.00	2.20	-0.87	0.38	-1.86	-0.40	-0.63
27.00	2.40	-0.47	0.66	-1.46	-0.20	-0.24
31.00	2.60	-0.90	0.52	-1.01	-0.15	$-2.67 \cdot 10^{-2}$
32.00	2.80	-0.56	$9.49 \cdot 10^{-2}$	-0.73	0.30	-0.54
35.00	3.00	-0.16	0.38	-0.33	0.50	-0.14
36.00	2.80	-0.19	1.31	-0.34	-1.68	$-4.65 \cdot 10^{-2}$
38.00	2.60	0.51	1.28	-1.54	-1.68	$2.28 \cdot 10^{-2}$
39.00	2.80	$8.08 \cdot 10^{-2}$	1.14	-1.09	-1.63	0.23
40.00	3.00	0.42	0.72	-0.80	-1.18	-0.27
47.00	3.20	$-4.09 \cdot 10^{-3}$	0.58	-0.35	-1.13	$-6.21 \cdot 10^{-2}$
48.00	3.40	0.34	0.15	$-6.05 \cdot 10^{-2}$	-0.68	-0.57
50.00	3.20	-0.77	$3.83 \cdot 10^{-2}$	-3.10	-0.71	-1.01
51.00	3.40	-0.37	0.32	-2.71	-0.51	-0.62
55.00	3.60	-0.80	0.18	-2.26	-0.46	-0.40
56.00	3.80	-0.46	-0.24	-1.97	$-1.2 \cdot 10^{-2}$	-0.91
59.00	4.00	$-6.07 \cdot 10^{-2}$	$3.83 \cdot 10^{-2}$	-1.58	0.19	-0.52
60.00	3.80	$-8.89 \cdot 10^{-2}$	0.97	-1.58	-1.99	-0.42
63.00	4.00	-0.51	0.83	-1.13	-1.94	-0.21
64.00	4.20	-0.17	0.41	-0.84	-1.49	-0.72
67.00	4.40	0.22	0.69	-0.45	-1.29	-0.33
70.00	4.20	0.92	0.66	-1.65	-1.29	-0.26
71.00	4.40	0.49	0.52	-1.20	-1.24	$-4.37 \cdot 10^{-2}$
72.00	4.60	0.83	$9.49 \cdot 10^{-2}$	-0.92	-0.79	-0.55
79.00	4.80	0.41	$-4.65 \cdot 10^{-2}$	-0.47	-0.74	-0.34

(d) Let $x, y \in \mathbb{R}^n$ (i.e. x and y are n dimensional vectors), and consider the following kernel:

$$k(x, y) = \prod_{i=1}^{n} (1 + x_i y_i).$$

What feature representation does this kernel induce? **Hint: try the cases when $n = 2$, $n = 3$, then figure out the pattern for larger n.** *What to submit: a solution can either be typed or handwritten (take a*

photo and upload to the pdf). All working must be shown for full marks.

Solution:
This is the 'all-subsets' kernel and induces the all-subsets embedding: $\phi : x \mapsto (\phi_A(x))_{A \subset \{1,\dots,n\}}$. For $n = 3$ say, we have

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_1x_2 \\ x_1x_3 \\ x_2x_3 \\ x_1x_2x_3 \end{bmatrix}$$

and for general n, $\phi_j(x) = x_1^{j_1} x_2^{j_2} \dots x_n^{j_n}$, where $j = (j_1, \dots, j_n) \in \{0, 1\}^n$.

Points Allocation There are a total of 15 marks, the available marks are:

- Question 1 a): 1 mark
- Question 1 b): 1.5 marks
- Question 1 c): 2 marks
- Question 1 d): 1 mark
- Question 1 e): 1 mark
- Question 1 f): 2 marks
- Question 1 g): 1 marks
- Question 2 a): 1 marks
- Question 2 b): 1 marks
- Question 2 c): 2.5 marks
- Question 2 d): 1 marks

What to Submit

- A single PDF file which contains solutions to each question. For each question, provide your solution in the form of text and requested plots. For any question in which you use code, provide a copy of your code at the bottom of the relevant section.
- **.py file(s) containing all code you used for the project, which should be provided in a seperate .zip file.** This code must match the code provided in the report.
- You may be deducted points for not following these instructions.
- You may be deducted points for poorly presented/formatted work. Please be neat and make your solutions clear.

- You cannot submit a python notebook, this will receive a mark of zero. This does not stop you from developing your code in a notebook and then copying it into a .py file though.

When and Where to Submit

- Due date: Monday **March 29th**, 2021 by **5:00pm**.
- Late submission incur a penalty of 10% per day for the first 5 days and 100% for more than 5 days.
- Submission has to be done through Moodle.

Final Reminder: You are required to submit one PDF file, AND also to submit the Python file(s) with all your code as a separate .zip file.