Benjamin Ye
CS/CNS/EE 156a: Learning Systems (Fall 2023)
December 1, 2023

Final Exam

Problem	Answer	Problem	Answer
1	[e]	11	[c]
2	[d]	12	[c]
3	[d]	13	[a]
4	[d]	14	[e]
5	[a]	15	[d]
6	[b]	16	[d]
7	[d]	17	[c]
8	[b]	18	[a]
9	[e]	19	[b]
10	[a]	20	[c]

Nonlinear Transforms

1. The polynomial transform of order $Q = 10$ applied to $\mathcal{X}$ of dimension $d = 2$ resuilts in a $\mathcal{Z}$ space of what dimensionality (not counting the constant coordinate $x_0 = 1$ or $z_0 = 1$)?

   Answer: [e] None of the above

   A second-order polynomial transformation gives $d = 5$:

   $$\Phi_2 : \mathbf{x} = (1, x_1, x_2) \rightarrow (1, x_1, x_2, x_1^2, x_1x_2, x_2^2)$$

   A third-order polynomial transformation gives $d = 9$:

   $$\Phi_3 : \mathbf{x} = (1, x_1, x_2) \rightarrow (1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3)$$

   A fourth-order polynomial transformation gives $d = 14$ (from Homework 5 Problem 3).

   The examples above show that each additional order Q adds $Q + 1$ terms. By continuing this pattern for higher-order polynomial transformations, the dimensionality for order Q is given by

   $$d(Q) = \sum_{q=1}^{Q} q + 1$$

   Therefore, for $Q = 10$, the $\mathcal{Z}$ space has dimensionality

   $$d(10) = \sum_{q=1}^{10} q + 1 = 65$$

Bias and Variance

2. Recall that the average hypothesis $\bar{g}$ was based on training the same model $\mathcal{H}$ on different data sets $\mathcal{D}$ to get $g^{(\mathcal{D})} \in \mathcal{H}$, and taking the expected value of $g^{(\mathcal{D})}$ with respect to $\mathcal{D}$ to get $\bar{g}$. Which of the following models $\mathcal{H}$ could result in $\bar{g} \notin \mathcal{H}$?

   Answer: [d] $\mathcal{H}$ is the logistic regression model

   Choices [a] through [c] cannot give $\bar{g} \notin \mathcal{H}$:

   [a] The average of a singleton hypothesis is itself, so $\bar{g} \in \mathcal{H}$.

   [b] The average of constant, real-valued hypotheses is a constant, real-valued number, so $\bar{g} \in \mathcal{H}$.

   [c] In the linear regression model, a hypothesis has the form of a linear combination $w_0 + \mathbf{w}^\top \mathbf{x}$. The average of these linear combinations is another linear combination, so $\bar{g} \in \mathcal{H}$.

   However, in the logistic regression model, a hypothesis has the form of a logistic (sigmoidal) function $1/(1 + \exp(-w_0 - \mathbf{w}^\top \mathbf{x}))$. The average of these logistic functions is not necessarily another logistic function, so $\bar{g} \notin \mathcal{H}$ could be true for [d].

   For example, consider the simple hypothesis set containing only

$$g_1 = \frac{1}{1 + \exp(1 - x)}, \quad g_2 = \frac{1}{1 + \exp(1 - 2x)}$$

   The average is

$$\bar{g} = \frac{2 + \exp(1 - x) + \exp(1 - 2x)}{2(1 + \exp(1 - x))(1 + \exp(1 - 2x))}$$

   which cannot be reduced to a logistic function.

Overfitting

3. Which of the following statements is false?

   Answer: [d] We can always determine if there is overfitting by comparing the values of $(E_{\text{out}} - E_{\text{in}})$.

   Overfitting occurs when a hypothesis with the lowest E_{in} is selected despite its higher E_{out}.

   Choices [a] through [c] are all true:

   [a] There must be at least two hypotheses with different E_{in} values to have overfitting.

   [b] There must be at least two hypotheses with different E_{out} values to have overfitting.

   [c] For overfitting to occur, the optimal and overfitted hypotheses must have different $E_{\text{out}} - E_{\text{in}}$ values. For example, consider two hypotheses with in-sample and out-of-sample error $\mathbf{E}_1 = (0.1,\ 0.3)$ and $\mathbf{E}_2 = (0.2,\ 0.2)$, with the first hypothesis selected for its lower E_{in} value. If the two hypotheses have the same $E_{\text{out}} - E_{\text{in}}$ values, then E_{out} for the second hypothesis must be 0.4, which would then mean that no overfitting occurred.

   Now, let's change the errors of the first hypothesis in the example in [c] to $\mathbf{E}_1 = (0.3,\ 0.5)$. The $E_{\text{out}} - E_{\text{in}}$ values for the old and new hypotheses are the same, but there is no longer overfitting. Therefore, the magnitudes of the $E_{\text{out}} - E_{\text{in}}$ values cannot be used to determine if there is overfitting, and [d] is false.

4. Which of the following statements is true?

   Answer: [d] Stochastic noise does not depend on the hypothesis set.

   Choices [a], [b], [c], and [e] are all false:

   [a] Deterministic and stochastic noise can be present simultaneously; the data set can have inherent fluctuations or measurement errors, and certain parts of the target function may be impossible to reproduce using the chosen model.

   [b] Deterministic noise is caused by the inability of the hypothesis set to model a target function (e.g., hypothesis set of linear combinations used to fit a quadratic target function).

   [c] Same explanation as [b].

   [e] Stochastic noise is dependent on the target distribution, as a distribution with a lower variance will have less stochastic noise.

   Since stochastic noise is a property of only the input data, [d] is true.

Regularization

5. The regularized weight $\mathbf{w}_{\text{reg}}$ is a solution to:

$$\text{minimize} \quad \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n - y_n)^2 \quad \text{subject to} \quad \mathbf{w}^{\mathsf{T}}\Gamma^{\mathsf{T}}\Gamma\mathbf{w} \leq C,$$

where Γ is a Tikhonov matrix. If $\mathbf{w}_{\text{lin}}^{\mathsf{T}}\Gamma^{\mathsf{T}}\Gamma\mathbf{w}_{\text{lin}} \leq C$, where $\mathbf{w}_{\text{lin}}$ is the linear regression solution, then what is $\mathbf{w}_{\text{reg}}$?

Answer: [a] $\mathbf{w}_{\text{reg}} = \mathbf{w}_{\text{lin}}$

As the linear regression solution $\mathbf{w}_{\text{lin}}$ minimizes the squared error and satisfies the same constraint used to determine $\mathbf{w}_{\text{reg}}$, it is equal to the regularized weight $\mathbf{w}_{\text{reg}}$.

6. Soft-order constraints that regularize polynomial models can be

Answer: [b] translated into augmented error

Soft-order constraints prevent overfitting by encouraging the weights to be small, but not necessarily zero like hard constraints, and does not change the order of the polynomial. This means that

[a] is false by definition,

[c] is false since the VC dimension does not change despite the effectively smaller model with regularization, and

[d] is false since E_{in} is expected to increase for better generalization (lower E_{out}).

A soft-order constraint yields the optimization problem

$$\min_{\mathbf{w}} E_{\text{in}}(\mathbf{w}) \quad \text{subject to} \quad \mathbf{w}^{\mathsf{T}}\mathbf{w} \leq C$$

For $\mathbf{w}_{\text{reg}}$ to be optimal,

$$\nabla E_{\text{in}}(\mathbf{w}_{\text{reg}}) = -2\lambda_C\mathbf{w}_{\text{reg}}$$

By grouping $-2\lambda_C$ into the undetermined variable λ_C, $\mathbf{w}_{\text{reg}}$ satisfies

$$\nabla(E_{\text{in}}(\mathbf{w}) + \lambda_C\mathbf{w}^{\mathsf{T}}\mathbf{w})|_{\mathbf{w}=\mathbf{w}_{\text{reg}}} = \mathbf{0}$$

That is, $\mathbf{w}_{\text{reg}}$ locally minimizes $E_{\text{in}}(\mathbf{w}) + \lambda_C\mathbf{w}^{\mathsf{T}}\mathbf{w}$, which is the augmented error where the second term represents a penalty term. Therefore, [b] is true.

Regularized Linear Regression

We are going to experiment with linear regression for classification on the processed U.S. Postal Service Zip Code data set from Homework 8. Download the data (extracted features of intensity and symmetry) for training and testing:

`http://www.amlbook.com/data/zip/features.train`

`http://www.amlbook.com/data/zip/features.test`

(The format of each row is: **`digit, intensity, symmetry`**.) We will train two types of binary classifiers; one-versus-one (one digit is class $+1$ and another digit is class -1, with the rest of the digits disregarded), and one-versus-all (one digit is class $+1$ and the rest of the digits are class -1). When evaluating E_{in} and E_{out} of the resulting classifier, use binary classification error. Implement the regularized least-squares linear regression for classification that minimizes

$$\frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n - y_n)^2 + \frac{\lambda}{N}\mathbf{w}^{\mathsf{T}}\mathbf{w}$$

where $\mathbf{w}$ includes w_0.

7. Set $\lambda = 1$ and dop not apply a feature transform (i.e., use $\mathbf{z} = \mathbf{x} = (1, x_1, x_2)$). Which among the following classifiers has the lowest E_{in}?

   Answer: [d] 8 versus all

8. Now, apply a feature transform $\mathbf{z} = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$, and set $\lambda = 1$. Which among the following classifiers has the lowest E_{out}?

   Answer: [b] 1 versus all

9. If we compare using the transform versus not using it, and applying that to "0 versus all" through "9 versus all", which of the following statements is correct for $\lambda = 1$?

   Answer: [e] The transform improves the out-of-sample performance of "5 versus all", but by less than 5%.

   The transform improves the out-of-sample error from 0.079721 to 0.079223, a 0.6% decrease, for the "5 versus all" classifier.

10. Train the "1 versus 5" classifier with $\mathbf{z} = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$ with $\lambda = 0.01$ and $\lambda = 1$. Which of the following statements is correct?

    Answer: [a] Overfitting occurs (from $\lambda = 1$ to $\lambda = 0.01$).

    From $\lambda = 1$ to $\lambda = 0.01$, E_{in} decreases from 0.005125 to 0.004484, but E_{out} increases from 0.025943 to 0.028302. This signifies overfitting since the $\lambda = 0.01$ hypothesis will be selected due for its lower in-sample error despite its worse out-of-sample performance.

(The sample program output and Python 3 source code are on the following pages. ⟶)

Sample program output

```
[Final Exam Problems 7–9]
Linear regression with regularization (lambda=1):
classifier     E_in    E_out  transform E_in  transform E_out
 0 vs. all 0.109313 0.115097        0.102318         0.106627
 1 vs. all 0.015224 0.022422        0.012344         0.021923
 2 vs. all 0.100261 0.098655        0.100261         0.098655
 3 vs. all 0.090248 0.082711        0.090248         0.082711
 4 vs. all 0.089425 0.099651        0.089425         0.099651
 5 vs. all 0.076258 0.079721        0.076258         0.079223
 6 vs. all 0.091071 0.084704        0.091071         0.084704
 7 vs. all 0.088465 0.073244        0.088465         0.073244
 8 vs. all 0.074338 0.082711        0.074338         0.082711
 9 vs. all 0.088328 0.088191        0.088328         0.088191

[Final Exam Problem 10]
Linear regression with regularization and transform for 1 vs. 5 classifier:
 lambda  in-sample error  out-of-sample error
   0.01         0.004484             0.028302
   1.00         0.005125             0.025943
```

Python 3 source code

```
from pathlib import Path

import numpy as np
import pandas as pd
import requests

DATA_DIR = Path(__file__).parent / "data"
rng = np.random.default_rng()

class LinearRegression:
    def __init__(
            self, *, vf=None, regularization=None, transform=None, noise=None,
            rng=None, seed=None, **kwargs):
        self.rng = np.random.default_rng(seed) if rng is None else rng
        self.set_parameters(vf=vf, regularization=regularization,
                            transform=transform, noise=noise, **kwargs)

    def get_error(self, x, y):
        if self.transform:
            x = self.transform(x)
        if self.noise:
            N = x.shape[0]
            index = self.rng.choice(N, round(self.noise[0] * N), False)
            y[index] = self.noise[1](y[index])

        if self.vf is not None and self.w is not None:
            return self.vf(self.w, x, y)
```

```
    def set_parameters(
            self, *, vf=None, regularization=None, transform=None, noise=None,
            update=False, **kwargs):
        self._reg_params = {}
        if update:
            self.noise = noise or self.noise
            self.regularization = regularization or self.regularization
            if self.regularization == "weight_decay" \
                    and "weight_decay_lambda" in kwargs:
                self._reg_params["lambda"] = kwargs["weight_decay_lambda"]
            self.transform = transform or self.transform
            self.vf = vf or self.vf
        else:
            self.noise = noise
            self.regularization = regularization
            if regularization == "weight_decay":
                self._reg_params["lambda"] = kwargs["weight_decay_lambda"]
            self.transform = transform
            self.vf = vf

    def train(self, x, y):
        if self.transform:
            x = self.transform(x)
        if self.noise:
            N = x.shape[0]
            index = self.rng.choice(N, round(self.noise[0] * N), False)
            y[index] = self.noise[1](y[index])
        if self.regularization is None:
            self.w = np.linalg.pinv(x) @ y
        elif self.regularization == "weight_decay":
            self.w = np.linalg.inv(
                x.T @ x
                + self._reg_params["lambda"] * np.eye(x.shape[1], dtype=float)
            ) @ x.T @ y
        if self.vf is not None:
            return self.vf(self.w, x, y)

def validate_binary(w, x, y):
    return np.count_nonzero(np.sign(x @ w) != y, axis=0) / x.shape[0]

if __name__ == "__main__":
    DATA_DIR.mkdir(exist_ok=True)
    data = {}
    for dataset in ["train", "test"]:
        file = f"features.{dataset}"
        if not (DATA_DIR / file).exists():
            r = requests.get(f"http://www.amlbook.com/data/zip/{file}")
            with open(DATA_DIR / file, "wb") as f:
                f.write(r.content)
        data[dataset] = np.loadtxt(DATA_DIR / file)
    weight_decay_lambda = 1
    transform = lambda x: np.hstack((x, x[:, 1:2] * x[:, 2:], x[:, 1:2] ** 2,
                                     x[:, 2:] ** 2))
```

```
reg = LinearRegression(vf=validate_binary, regularization="weight_decay",
                       weight_decay_lambda=weight_decay_lambda, rng=rng)
reg_transform = LinearRegression(vf=validate_binary,
                                 regularization="weight_decay",
                                 transform=transform,
                                 weight_decay_lambda=weight_decay_lambda,
                                 rng=rng)
df = pd.DataFrame(columns=["classifier", "E_in", "E_out",
                           "transform E_in", "transform E_out"])
for digit in range(10):
    x_train = np.hstack((np.ones((len(data["train"]), 1), dtype=float),
                         data["train"][:, 1:]))
    y_train = 2 * (data["train"][:, 0] == digit) - 1
    x_test = np.hstack((np.ones((len(data["test"]), 1), dtype=float),
                        data["test"][:, 1:]))
    y_test = 2 * (data["test"][:, 0] == digit) - 1
    E_in = reg.train(x_train, y_train)
    E_in_transform = reg_transform.train(x_train, y_train)
    df.loc[digit] = (f"{digit} vs. all", E_in,
                     reg.get_error(x_test, y_test), E_in_transform,
                     reg_transform.get_error(x_test, y_test))
print("\n[Final Exam Problems 7–9]\n"
      "Linear regression with regularization "
      f"(lambda={weight_decay_lambda}):\n", df.to_string(index=False),
      sep="")

subset_train = data["train"][np.isin(data["train"][:, 0], (1, 5))]
x_train = np.hstack((np.ones((subset_train.shape[0], 1), dtype=float),
                     subset_train[:, 1:]))
y_train = (subset_train[:, 0] == 1).astype(int) - (subset_train[:, 0] == 5)
subset_test = data["test"][np.isin(data["test"][:, 0], (1, 5))]
x_test = np.hstack((np.ones((subset_test.shape[0], 1), dtype=float),
                    subset_test[:, 1:]))
y_test = (subset_test[:, 0] == 1).astype(int) - (subset_test[:, 0] == 5)
df = pd.DataFrame(columns=["lambda", "in-sample error",
                           "out-of-sample error"])
for weight_decay_lambda in (0.01, 1):
    reg_transform.set_parameters(weight_decay_lambda=weight_decay_lambda,
                                 update=True)
    E_in = reg_transform.train(x_train, y_train)
    df.loc[len(df)] = (weight_decay_lambda, E_in,
                       reg_transform.get_error(x_test, y_test))
print("\n[Final Exam Problem 10]\n"
      "Linear regression with regularization and transform for "
      "1 vs. 5 classifier:\n", df.to_string(index=False), sep="")
```

Support Vector Machines

11. Consider the following training set generated from a target function $f : \mathcal{X} \to \{-1, +1\}$, where $\mathcal{X} = \mathbb{R}^2$:

$$\mathbf{x}_1 = (1, 0), y_1 = -1, \quad \mathbf{x}_2 = (0, 1), y_2 = -1, \quad \mathbf{x}_3 = (0, -1), y_3 = -1$$

$$\mathbf{x}_4 = (-1, 0), y_1 = +1, \quad \mathbf{x}_2 = (0, 2), y_2 = +1, \quad \mathbf{x}_3 = (0, -2), y_3 = +1$$

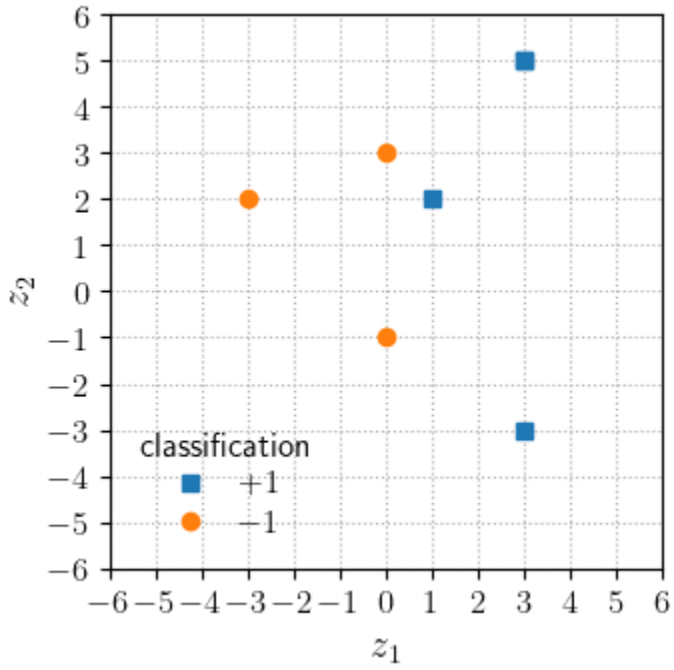
$$\mathbf{x}_7 = (-2, 0), y_7 = +1$$

Transform this training set into another two-dimensional space $\mathcal{Z}$:

$$z_1 = x_2^2 - 2x_1 - 1, \quad z_2 = x_1^2 - 2x_2 + 1$$

Using geometry (not quadratic programming), what values of $\mathbf{w}$ (without w_0) and b specify the separating plane $\mathbf{w}^\mathsf{T}\mathbf{z} + b = 0$ that maximizes the margin in the $\mathcal{Z}$ space? The values of w_1, w_2, and b are:

Answer: [c] $1, 0, -0.5$



In $\mathcal{Z}$ space, the separating plane should be placed between the two groups of blue squares and orange circles above. The ideal placement is at $z_1 = 0.5$ and any z_2 ($\mathbf{z} = (0.5, z_2)$ is a point on the plane). This arrangement maximizes the margin between the plane and the three closest points at $(0, -1)$, $(0, 3)$, and $(1, 2)$, with each point having a Euclidean distance of 0.5 from the plane.

The form of $\mathbf{z}$ suggests that $w_1 = 1$ and $w_2 = 0$ (since the z_2 value is irrelevant), and b can be found using

$$(1 \quad 0)^\mathsf{T}(0.5 \quad z_2) + b = 0 \longrightarrow b = -0.5$$

12. Consider the same training set of the previous problem, but instead of explicitly transforming the input space $\mathcal{X}$, apply the hard-margin SVM algorithm with the kernel

$$K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^\top \mathbf{x}')^2$$

(which corresponds to a second-order polynomial transformation). Set up the expression for $\mathcal{L}(\alpha_1, \ldots, \alpha_7)$ and solve for the optimal $\alpha_1, \ldots, \alpha_7$ (numerically, using a quadratic programming package). The number of support vectors you get is in what range?

Answer: [c] 4–5

Derivation

The `cvxopt` Python quadratic programming package can be used to implement the hard margin SVM algorithm.

The `cvxopt.solvers.qp()` function solves

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{x}^\top P\mathbf{x} + \mathbf{q}^\top \mathbf{x} \\ \text{subject to} \quad & G\mathbf{x} \leq \mathbf{h}, \;\; A\mathbf{x} = \mathbf{b} \end{aligned}$$

The dual problem to the hard margin SVM primal is

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\boldsymbol{\alpha}^\top Q\boldsymbol{\alpha} - \mathbf{e}^\top \boldsymbol{\alpha} \\ \text{subject to} \quad & \mathbf{y}^\top \boldsymbol{\alpha} = 0, \;\; 0 \leq \alpha_i \leq C, \;\; i = 1, \ldots, n \end{aligned}$$

where $\mathbf{e}$ is a vector of all ones, $Q = (\mathbf{y} \otimes \mathbf{y})K(\mathbf{x}, \mathbf{x}')$ is a positive semidefinite matrix, and $\boldsymbol{\alpha}$ is a vector containing the dual coefficients, each of which is bounded by $C = \infty$ (naturally).

By mapping the dual problem to the `cvxopt.solvers.qp()` function,

$$P = Q = (\mathbf{y} \otimes \mathbf{y})K(\mathbf{x}, \mathbf{x}'), \quad \mathbf{q} = \mathbf{e}, \quad G = -\mathbf{I}, \quad \mathbf{h} = \mathbf{0}, \quad \mathbf{x} = \boldsymbol{\alpha}, \quad A = \mathbf{y}^\top, \quad \mathbf{b} = 0$$

As a check, the hard margin SVM implementation in `sklearn.svm.SVC` is also fitted using the training set to verify that the number of support vectors is correct.

Sample program output

```
[Final Exam Problem 11]
The second-order polynomial hard margin support vector machine (SVM) uses
  - cvxopt.solvers.qp: 5 support vectors;
  - sklearn.svm.SVC: 5 support vectors.
```

(The Python 3 source code is on the following pages. ⟶)

Python 3 source code

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm

mpl.rcParams.update(
    {
        "axes.labelsize": 14,
        "figure.autolayout": True,
        "figure.figsize": (4.875, 3.65625),
        "font.size": 12,
        "legend.columnspacing": 1,
        "legend.edgecolor": "1",
        "legend.framealpha": 0,
        "legend.fontsize": 12,
        "legend.handlelength": 1.25,
        "legend.labelspacing": 0.25,
        "xtick.labelsize": 12,
        "ytick.labelsize": 12,
        "text.usetex": True
    }
)

if __name__ == "__main__":
    x = np.array(((1, 0), (0, 1), (0, -1), (-1, 0), (0, 2), (0, -2), (-2, 0)),
                 dtype=float)
    y = np.array((-1, -1, -1, 1, 1, 1, 1), dtype=float)
    z = np.hstack((x[:, 1:] ** 2 - 2 * x[:, :1] - 1,
                   x[:, :1] ** 2 - 2 * x[:, 1:] + 1))

    ticks = np.arange(-6, 7)
    _, ax = plt.subplots()
    ax.plot(*z[y == 1].T, "s", label="$+1$")
    ax.plot(*z[y == -1].T, "o", label="$-1$")
    ax.grid(ls=":")
    ax.legend(title="classification", loc="lower left")
    ax.set_aspect("equal", "box")
    ax.set_xlabel("$z_1$")
    ax.set_xlim(-6, 6)
    ax.set_xticks(ticks)
    ax.set_ylabel("$z_2$")
    ax.set_ylim(-6, 6)
    ax.set_yticks(ticks)
    plt.show()

    solvers.options['show_progress'] = False
    solution = solvers.qp(matrix(np.outer(y, y) * (1 + x @ x.T) ** 2),
                          matrix(-np.ones((x.shape[0], 1))),
                          matrix(-np.eye(x.shape[0])),
                          matrix(np.zeros(x.shape[0])),
                          matrix(y[None]),
                          matrix(0.0))
```

```
clf = svm.SVC(C=np.finfo(float).max, kernel="poly", degree=2, gamma=1,
              coef0=1)
clf.fit(x, y)

print("\n[Final Exam Problem 11]\n"
      "The second-order polynomial hard margin support vector "
      "machine (SVM) uses\n  - cvxopt.solvers.qp: "
      f"{(~np.isclose(solution['x'], 0)).sum()} support vectors;\n"
      f"  - sklearn.svm.SVC: {clf.n_support_.sum()} support vectors.")
```

Radial Basis Functions

We experiment with the RBF model, both in regular form (Lloyd + pseudo-inverse) with K centers:

$$\text{sign}\left(\sum_{k=1}^{K} w_k \exp(-\gamma\|\mathbf{x}-\mu_k\|^2)+b\right)$$

(notice that there is a bias term), and in kernel form (using the RBF kernel in hard-margin SVM):

$$\text{sign}\left(\sum_{\alpha_n>0} \alpha_n y_n \exp(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2)+b\right).$$

The input space is $\mathcal{X}=[-1,\,1]\times[-1,\,1]$ with uniform probability distribution, and the target is

$$f(\mathbf{x})=\text{sgn}(x_2-x_1+0.25\sin(\pi x_1))$$

which is slightly nonlinear in the $\mathcal{X}$ space. In each run, generate 100 training points at random using this target, and apply both forms of RBF to these training points. Here are some guidelines:

- Repeat the experiment for as many runs as needed to get the answer to be stable (statistically away from flipping to the closest competing answer).
- In case a data set is not separable in the "$\mathcal{Z}$ space" by the RBF kernel using hard-margin SVM, discard the run but keep track of how often this happens, if ever.
- When you use Lloyd's algorithm, initialize the centers to random points in $\mathcal{X}$ and iterate until there is no change from iteration to iteration. If a cluster becomes empty, discard the run and repeat.

13. For $\gamma=1.5$, how often do you get a data set that is not separable by the RBF kernel (using hard-margin SVM)? Hint: Run the hard-margin SVM, then check if the solution has $E_{\text{in}}\neq 0$.

    Answer: [a] $\leq 5\%$ of the time

14. If we use $K=9$ for regular RBF and take $\gamma=1.5$, how often does the kernel form beat the regular form (excluding runs mentioned in Problem 13 and runs with empty clusters, if any) in terms of E_{out}?

    Answer: [e] $> 75\%$ of the time

15. If we use $K=12$ for regular RBF and take $\gamma=1.5$, how often does the kernel form beat the regular form (excluding runs mentioned in Problem 13 and runs with empty clusters, if any) in terms of E_{out}?

    Answer: [d] $> 60\%$ but $\leq 90\%$ of the time

16. Now we focus on regular RBF only, with $\gamma = 1.5$. If we go from $K = 9$ clusters to $K = 12$ clusters (only 9 and 12), which of the following 5 cases happens most often in your runs (excluding runs with empty clusters, if any)? Up or down means strictly so.

    Answer: [d] Both E_{in} and E_{out} go down.

17. For regular RBF with $K = 9$, if we go from $\gamma = 1.5$ to $\gamma = 2$ (only 1.5 and 2), which of the following 5 cases happens most often in your runs (excluding runs with empty clusters, if any)? Up or down means strictly so.

    Answer: [c] Both E_{in} and E_{out} go up.

18. What is the percentage of time that regular RBF achieves $E_{\text{in}} = 0$ with $K = 9$ and $\gamma = 1.5$ (excluding runs with empty clusters, if any)?

    Answer: [a] $\leq 10\%$ of the time

Sample output

```
[Final Exam Problems 13–18]
Radial basis function (RBF) model (100 runs):
 gamma    K  nonseparable  outperform  E_in=0   E_in   E_out
   1.5  9.0           0.0        0.83    0.04 0.0273 0.04678
   1.5 12.0           0.0        0.84    0.12 0.0185 0.04045
   2.0  9.0           0.0        0.86    0.06 0.0296 0.05223
```

(The Python 3 source code is on the following pages. ⟶)

Python 3 source code

```
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.cluster import k_means

rng = np.random.default_rng()

class RBFRegular:
    def __init__(self, gamma, K, *, vf) -> None:
        self.set_parameters(gamma, K, vf=vf)

    def get_error(self, x, y):
        if self.vf is not None and self.w is not None:
            return self.vf(self.w, self.get_phi(x, self.centers), y)

    def get_phi(self, x, centers):
        return np.hstack((
            np.ones((x.shape[0], 1), dtype=float),
            np.exp(-self.gamma
                   * np.linalg.norm((x[:, None] - centers), axis=2) ** 2)
        ))

    def set_parameters(self, gamma, K, *, vf=None, update=False) :
        if update:
            self.gamma = gamma or self.gamma
            self.K = K or self.K
            self.vf = vf or self.vf
        else:
            self.gamma = gamma
            self.K = K
            self.vf = vf

    def train(self, x: np.ndarray[float], y: np.ndarray[float]) -> float:
        self.centers = k_means(x, self.K, n_init="auto")[0]
        phi = self.get_phi(x, self.centers)
        self.w = np.linalg.pinv(phi) @ y
        if self.vf is not None:
            return self.vf(self.w, phi, y)

def generate_data(
        N, f, d=2, lb=-1.0, ub=1.0, *, bias=False, rng=None, seed=None):
    if rng is None:
        rng = np.random.default_rng(seed)
    x = rng.uniform(lb, ub, (N, d))
    if bias:
        x = np.hstack((np.ones((N, 1)), x))
    return x, f(x)

def validate_binary(w, x, y):
    return np.count_nonzero(np.sign(x @ w) != y, axis=0) / x.shape[0]
```

```
def target_function_final_exam(x):
    f = lambda x: np.sign(np.diff(x[:, -2:], axis=1)[:, 0]
                          + 0.25 * np.sin(np.pi * x[:, -1]))
    return f if x is None else f(x)

if __name__ == "__main__":
    N_runs = 100
    N_train = 100
    N_test = 1_000
    columns = ["gamma", "K", "nonseparable", "outperform",
               "E_in=0", "E_in", "E_out"]
    df = pd.DataFrame(columns=columns)
    for gamma, K in [(1.5, 9), (1.5, 12), (2, 9)]:
        clf = svm.SVC(C=np.finfo(float).max, gamma=gamma)
        rbf = RBFRegular(gamma, K, vf=validate_binary)
        counters = np.zeros(5, dtype=float)
        for _ in range(N_runs):
            x_train, y_train = generate_data(N_train,
                                             target_function_final_exam,
                                             rng=rng)
            x_test, y_test = generate_data(N_test,
                                           target_function_final_exam,
                                           rng=rng)
            clf.fit(x_train, y_train)
            if not np.isclose(clf.score(x_train, y_train), 1):
                counters[0] += 1
                continue
            E_in = rbf.train(x_train, y_train)
            E_out = rbf.get_error(x_test, y_test)
            counters[1:5] += (1 - clf.score(x_test, y_test) < E_out,
                              E_in == 0, E_in, E_out)
        counters /= N_runs
        counters[:3] *= 100
        df.loc[len(df)] = gamma, K, *counters
    print("\n[Final Exam Problems 13–18]\n"
          f"Radial basis function (RBF) model ({N_runs:,} runs):\n",
          df.to_string(index=False), sep="")
```

Bayesian Priors

19. Let $f \in [0, 1]$ be the unknown probability of getting a heart attack for people in a certain population. Notice that f is just a constant, not a function, for simplicity. We want to model f using a hypothesis $h \in [0, 1]$. Before we see any data, we assume that $P(h = f)$ is uniform over $h \in [0, 1]$ (the prior). We pick one person from the population, and it turns out that he or she had a heart attack. Which of the following is true about the posterior probability that $h = f$ given this sample point?

    Answer: [b] The posterior increases linearly over $[0, 1]$.

    Using Bayes' theorem, the posterior $P(h = f \mid \mathcal{D})$ can be related to the prior $P(h = f)$ via

    $$P(h = f \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid h = f)P(h = f)}{P(\mathcal{D})} \propto P(\mathcal{D} \mid h = f)P(h = f)$$

    $P(\mathcal{D})$ is expected to be a constant probability, akin to one used in a binomial distribution. The randomly chosen person either has or has not had a heart attack.

    $P(\mathcal{D} \mid h = f)$ is the probability that a randomly chosen person has had a heart attack given the likelihood $h = f$ of getting a heart attack in that population. Intuitively, this probability increases linearly with $h = f$; the higher the chance of getting a heart attack, the more likely the randomly selected person has had one.

    From the problem statement, it is known that $P(h = f)$ is a uniform distribution, i.e., constant across $[0, 1]$.

    Multiplying a linearly increasing distribution by a uniform distribution gives back a linearly increasing distribution. Therefore, the posterior is expected to increase linearly over $[0, 1]$.

Aggregation

20. Given two learned hypotheses g_1 and g_2, we construct the aggregate hypothesis g given by $g(\mathbf{x}) = 0.5\big(g_1(\mathbf{x}) + g_2(\mathbf{x})\big)$ for all $\mathbf{x} \in \mathcal{X}$. If we use the mean-squared error, which of the following statements is true?

Answer: [c] $E_{\text{out}}(g)$ cannot be worse than the average of $E_{\text{out}}(g_1)$ and $E_{\text{out}}(g_2)$.

Choices [a], [b], and [d] are all false:

[a] Consider the case where $E_{\text{out}}(g_1) < E_{\text{out}}(g_2)$, or that g_2 is further away from the target function f than g_1 is, with both g_1 and g_2 overestimating f. By taking their average to get g, the resulting hypothesis will model f less accurately and have a higher mean-squared error than g_1.

[b] Same explanation as [a].

[d] Consider the case where g_1 underestimates the target function while g_2 overestimates it. The averaged hypothesis g could then be much closer to the target function and result in $E_{\text{out}}(g)$ being much lower than both $E_{\text{out}}(g_1)$ and $E_{\text{out}}(g_2)$.

Using the definition of the mean-squared error,

$$E_{\text{out}}(g) = \mathbb{E}[(f - g)^2]$$

it can be shown that the fully expanded mean-squared error of g is

$$E_{\text{out}}(g) = \mathbb{E}\left[\left(f - \frac{1}{2}(g_1 + g_2)\right)^2\right] = \mathbb{E}\left[f^2 - fg_1 - fg_2 + \frac{1}{4}g_1^2 + \frac{1}{2}g_1g_2 + \frac{1}{4}g_2\right]$$

The average of the mean-squared errors of g_1 and g_2 is

$$\begin{aligned}
\frac{1}{2}\big(E_{\text{out}}(g_1) + E_{\text{out}}(g_2)\big) &= \frac{1}{2}(\mathbb{E}[(f - g_1)^2] + \mathbb{E}[(f - g_2)^2]) \\
&= \frac{1}{2}(\mathbb{E}[f^2 - 2fg_1 + g_1^2] + \mathbb{E}[f^2 - 2fg_2 + g_2^2]) \\
&= \mathbb{E}\left[f^2 - fg_1 - fg_2 + \frac{1}{2}g_1^2 + \frac{1}{2}g_2^2\right] \\
&= \mathbb{E}\left[\left(f^2 - fg_1 - fg_2 + \frac{1}{4}g_1^2 + \frac{1}{2}g_1g_2 + \frac{1}{4}g_2^2\right) + \left(\frac{1}{4}g_1^2 - \frac{1}{2}g_1g_2 + \frac{1}{4}g_2^2\right)\right] \\
&= E_{\text{out}}(g) + \frac{1}{4}\mathbb{E}[(g_1 - g_2)^2]
\end{aligned}$$

The second term in the last line is greater than or equal to zero since it is squared. Therefore, $E_{\text{out}}(g)$ is always smaller than the average of $E_{\text{out}}(g_1)$ and $E_{\text{out}}(g_2)$, so [c] is true.